Universitas Pertamina

MANUAL BOOK

Benchmarking and Performance Measurement of Neural Network Models with Two Hidden Layers for Time Series Data : Case Study of RNN, LSTM and GRU-based Structure

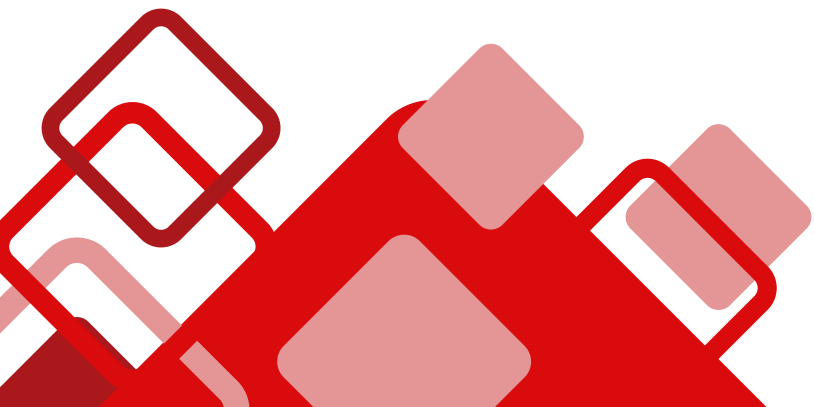
Researcher

1. Dr. Ariana Yunita
2. Dr. Tasmi
3. Dr. Muhammad Zaki Almuzakki

2025

Benchmarking and Performance Measurement of Neural Network Models with Two Hidden Layers for Time Series Data : Case Study of RNN, LSTM and GRU-based Structure

Acknowledgement:
We acknowledge funding from the 2023 Universitas Pertamina – Universiti Teknologi Petronas Collaborative Research. We also extend our gratitude to our research assistants, MHD Iqbal Ramadhan and Kevin Adam Prasetya, for their valuable contributions to the technical aspects of this work.



Manual Book

Menu dan Cara Penggunaan

Menyiapkan *workspace* yang dapat menjalankan *file* dengan ekstensi .ipynb. Anda dapat menggunakan *Google Colab (Colaboratory)* untuk menjalankan *file* dengan ekstensi .ipynb, dengan langkah-langkah berikut :

1. Buka *Google Colab* : Buka perambanan web yang ada di device Anda dan pergi ke *Google Colab.*
2. *Log In :*

   Pastikan Anda sudah masuk menggunakan akun *Google* Anda. Jika belum, Anda perlu memasukkan akun Anda terlebih dahulu untuk menggunakan *Google Colab*.
3. Buat *notebook* baru atau *import* dari *file* dari *device* Anda :

   Anda dapat membuat *notebook* baru dengan mengklik "*File*">"*New Notebook*" di *Google Colab*. Jika Anda sudah memiliki *file* .ipynb yang ingin dijalankan, Anda dapat mengimportnya dengan mengklik "*File*">"*Upload Notebook*".
4. Jalankan Program :

   Dalam *notebook colab*, Anda dapat menjalankan program dengan beberapa cara:

   4.1. Menjalankan *cell* secara individual :

   Klik pada *cell code* yang ingin Anda jalankan, dan tekan tombol "*Shift*"+"*Enter*" atau klik tombol "*Run*" yang ada disamping *cell code*.

   4.2. Menjalankan semua cell :

   Klik "*Runtime*" di atas, kempudian pilih "*Run All*" untuk menjalankan semua *cell code* dalam *notebook* secara otomatis.
5. Mengunggah Dataset (*Optional*) :

   Dataset sudah disediakan, apabila memiliki dataset lain, unggah dataset baru dan sesuaikan dengan nama *file*.

Benchmarking and Performance Measurement of Neural Networks Models With Two Hidden Layers For Time-Series Data: Case Study of RNN, LSTM, and GRU-based Structure

1. Import Libraries

List of libraries that will be used.

In [1]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import csv
import random
```

Using Utilites of Data Visualization

In [2]:
```
def plot_series(x, y, format="-", start=0, end=None,
                title=None, xlabel=None, ylabel=None, legend=None ):
    plt.figure(figsize=(10, 6))
    if type(y) is tuple:
      for y_curr in y:
        plt.plot(x[start:end], y_curr[start:end], format)
    else:
      plt.plot(x[start:end], y[start:end], format)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    if legend:
      plt.legend(legend)
    plt.title(title)
    plt.grid(True)
    plt.show()
```

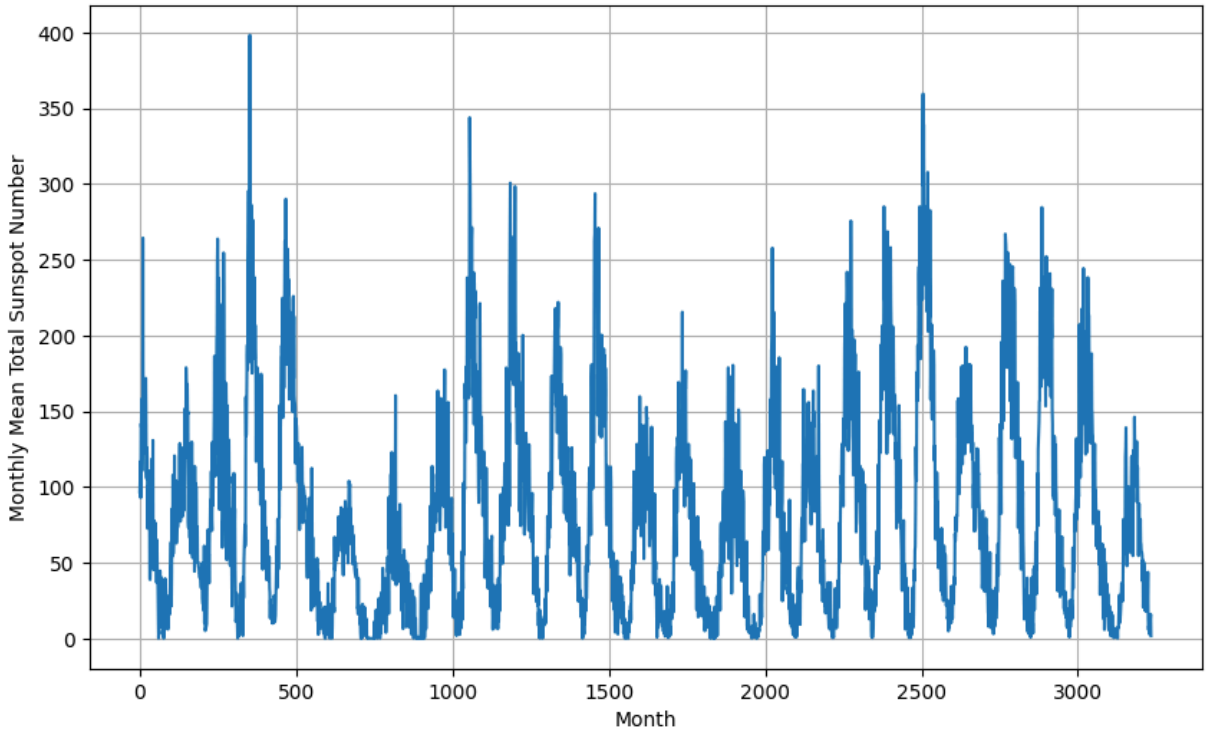
2. Get DATA

Take Data from Upstream (Local, Gdrive, Github, etc).

In [ ]:
```
!wget https://storage.googleapis.com/tensorflow-1-public/course4/Sunspots.csv
```

Data Visualize

In [4]:
```
time_step = []
sunspots = []

with open('./Sunspots.csv') as csvfile:
  reader = csv.reader(csvfile, delimiter=',')
  next(reader)

  for row in reader:
    time_step.append(int(row[0]))
    sunspots.append(float(row[2]))

time = np.array(time_step)
series = np.array(sunspots)

plot_series(time, series, xlabel='Month', ylabel='Monthly Mean Total Sunspot Number')
```



3. Data Prerocessing

Analysis and Cleaning Data -> Extract, Transform, Load.

Split Data

In [5]:
```
indices_to_change = np.where((series >= 0) & (series <= 1))
for index in indices_to_change[0]:
  series[index] = random.random() * 10

split_time = int(len(time)*0.8)

time_train = time[:split_time]
x_train = series[:split_time]

time_valid = time[split_time:]
x_valid = series[split_time:]
```

Prepare Features and Labels

In [6]:
```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
    dataset = dataset.map(lambda window: (window[:-1], window[-1]))
    dataset = dataset.shuffle(shuffle_buffer)
    dataset = dataset.batch(batch_size).prefetch(1)
    return dataset

window_size = 30
batch_size = 32
shuffle_buffer_size = 1000

train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)
```

4. Model Architechture

Using library from Scikit-learn, Tensorflow(?).

RNN Model

In [7]:
```
model_RNN = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                        activation="relu",
                        padding='causal',
                        input_shape=[window_size, 1]),
  tf.keras.layers.SimpleRNN(64, return_sequences=True),
  tf.keras.layers.SimpleRNN(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_RNN.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential mode
ls, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 30, 64)	256
simple_rnn (SimpleRNN)	(None, 30, 64)	8,256
simple_rnn_1 (SimpleRNN)	(None, 64)	8,256
dense (Dense)	(None, 1)	65
lambda (Lambda)	(None, 1)	0

Total params: 16,833 (65.75 KB)

Trainable params: 16,833 (65.75 KB)

Non-trainable params: 0 (0.00 B)

LSTM Model

In [8]:
```
model_LSTM = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                        activation="relu",
                        padding='causal',
                        input_shape=[window_size, 1]),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.LSTM(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_LSTM.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 30, 64)	256
lstm (LSTM)	(None, 30, 64)	33,024
lstm_1 (LSTM)	(None, 64)	33,024
dense_1 (Dense)	(None, 1)	65
lambda_1 (Lambda)	(None, 1)	0

Total params: 66,369 (259.25 KB)

Trainable params: 66,369 (259.25 KB)

GRU Model

In [9]:
```
model_GRU = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                        activation="relu",
                        padding='causal',
                        input_shape=[window_size, 1]),
  tf.keras.layers.GRU(64, return_sequences=True),
  tf.keras.layers.GRU(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_GRU.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv1d_2 (Conv1D)	(None, 30, 64)	256
gru (GRU)	(None, 30, 64)	24,960
gru_1 (GRU)	(None, 64)	24,960
dense_2 (Dense)	(None, 1)	65
lambda_2 (Lambda)	(None, 1)	0

Total params: 50,241 (196.25 KB)

Trainable params: 50,241 (196.25 KB)

Non-trainable params: 0 (0.00 B)

RNN-LSTM Model

In [10]:
```
model_RNN_LSTM = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                         activation="relu",
                         padding='causal',
                         input_shape=[window_size, 1]),
  tf.keras.layers.SimpleRNN(64, return_sequences=True),
  tf.keras.layers.LSTM(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
```

```
])
model_RNN_LSTM.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv1d_3 (Conv1D)	(None, 30, 64)	256
simple_rnn_2 (SimpleRNN)	(None, 30, 64)	8,256
lstm_2 (LSTM)	(None, 64)	33,024
dense_3 (Dense)	(None, 1)	65
lambda_3 (Lambda)	(None, 1)	0

Total params: 41,601 (162.50 KB)

Trainable params: 41,601 (162.50 KB)

Non-trainable params: 0 (0.00 B)

RNN-GRU Model

In [11]:
```
model_RNN_GRU = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                         activation="relu",
                         padding='causal',
                         input_shape=[window_size, 1]),
  tf.keras.layers.SimpleRNN(64, return_sequences=True),
  tf.keras.layers.GRU(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_RNN_GRU.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv1d_4 (Conv1D)	(None, 30, 64)	256
simple_rnn_3 (SimpleRNN)	(None, 30, 64)	8,256
gru_2 (GRU)	(None, 64)	24,960
dense_4 (Dense)	(None, 1)	65
lambda_4 (Lambda)	(None, 1)	0

Total params: 33,537 (131.00 KB)

Trainable params: 33,537 (131.00 KB)

Non-trainable params: 0 (0.00 B)

LSTM-RNN

In [12]:
```
model_LSTM_RNN = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                         activation="relu",
                         padding='causal',
```

```
                      input_shape=[window_size, 1]),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.SimpleRNN(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_LSTM_RNN.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
conv1d_5 (Conv1D)	(None, 30, 64)	256
lstm_3 (LSTM)	(None, 30, 64)	33,024
simple_rnn_4 (SimpleRNN)	(None, 64)	8,256
dense_5 (Dense)	(None, 1)	65
lambda_5 (Lambda)	(None, 1)	0

Total params: 41,601 (162.50 KB)

Trainable params: 41,601 (162.50 KB)

Non-trainable params: 0 (0.00 B)

LSTM-GRU Model

In [13]:
```
model_LSTM_GRU = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                      strides=1,
                      activation="relu",
                      padding='causal',
                      input_shape=[window_size, 1]),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.GRU(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_LSTM_GRU.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
conv1d_6 (Conv1D)	(None, 30, 64)	256
lstm_4 (LSTM)	(None, 30, 64)	33,024
gru_3 (GRU)	(None, 64)	24,960
dense_6 (Dense)	(None, 1)	65
lambda_6 (Lambda)	(None, 1)	0

Total params: 58,305 (227.75 KB)

Trainable params: 58,305 (227.75 KB)

Non-trainable params: 0 (0.00 B)

GRU-RNN Model

In [14]:
```
model_GRU_RNN = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                        activation="relu",
                        padding='causal',
                        input_shape=[window_size, 1]),
  tf.keras.layers.GRU(64, return_sequences=True),
  tf.keras.layers.SimpleRNN(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_GRU_RNN.summary()
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
conv1d_7 (Conv1D)	(None, 30, 64)	256
gru_4 (GRU)	(None, 30, 64)	24,960
simple_rnn_5 (SimpleRNN)	(None, 64)	8,256
dense_7 (Dense)	(None, 1)	65
lambda_7 (Lambda)	(None, 1)	0

Total params: 33,537 (131.00 KB)

Trainable params: 33,537 (131.00 KB)

Non-trainable params: 0 (0.00 B)

GRU-LSTM Model

In [15]:
```
model_GRU_LSTM = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                        activation="relu",
                        padding='causal',
                        input_shape=[window_size, 1]),
  tf.keras.layers.GRU(64, return_sequences=True),
  tf.keras.layers.LSTM(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_GRU_LSTM  .summary()
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
conv1d_8 (Conv1D)	(None, 30, 64)	256
gru_5 (GRU)	(None, 30, 64)	24,960
lstm_5 (LSTM)	(None, 64)	33,024
dense_8 (Dense)	(None, 1)	65
lambda_8 (Lambda)	(None, 1)	0

Total params: 58,305 (227.75 KB)

Trainable params: 58,305 (227.75 KB)

Non-trainable params: 0 (0.00 B)

5. Model Training

In [16]:
```
tf.keras.backend.clear_session()
learning_rate = 8e-7
```

In [ ]:
```
optimizer_RNN = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_RNN.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_RNN,
              metrics=["mae", "mse", "mape"])
history_RNN = model_RNN.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_LSTM = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_LSTM.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_LSTM,
              metrics=["mae", "mse", "mape"])
history_LSTM = model_LSTM.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_GRU= tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_GRU.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_GRU,
              metrics=["mae", "mse", "mape"])
history_GRU = model_GRU.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_RNN_LSTM = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_RNN_LSTM.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_RNN_LSTM,
              metrics=["mae", "mse", "mape"])
history_RNN_LSTM = model_RNN_LSTM.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_RNN_GRU = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_RNN_GRU.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_RNN_GRU,
              metrics=["mae", "mse", "mape"])
history_RNN_GRU = model_RNN_GRU.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_LSTM_RNN = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_LSTM_RNN.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_LSTM_RNN,
              metrics=["mae", "mse", "mape"])
history_LSTM_RNN = model_LSTM_RNN.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_LSTM_GRU = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_LSTM_GRU.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_LSTM_GRU,
              metrics=["mae", "mse", "mape"])
history_LSTM_GRU = model_LSTM_GRU.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_GRU_RNN= tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_GRU_RNN.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_GRU_RNN,
              metrics=["mae", "mse", "mape"])
history_GRU_RNN = model_GRU_RNN.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_GRU_LSTM= tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_GRU_LSTM.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_GRU_LSTM,
              metrics=["mae", "mse", "mape"])
history_GRU_LSTM = model_GRU_LSTM.fit(train_set,epochs=100)
```

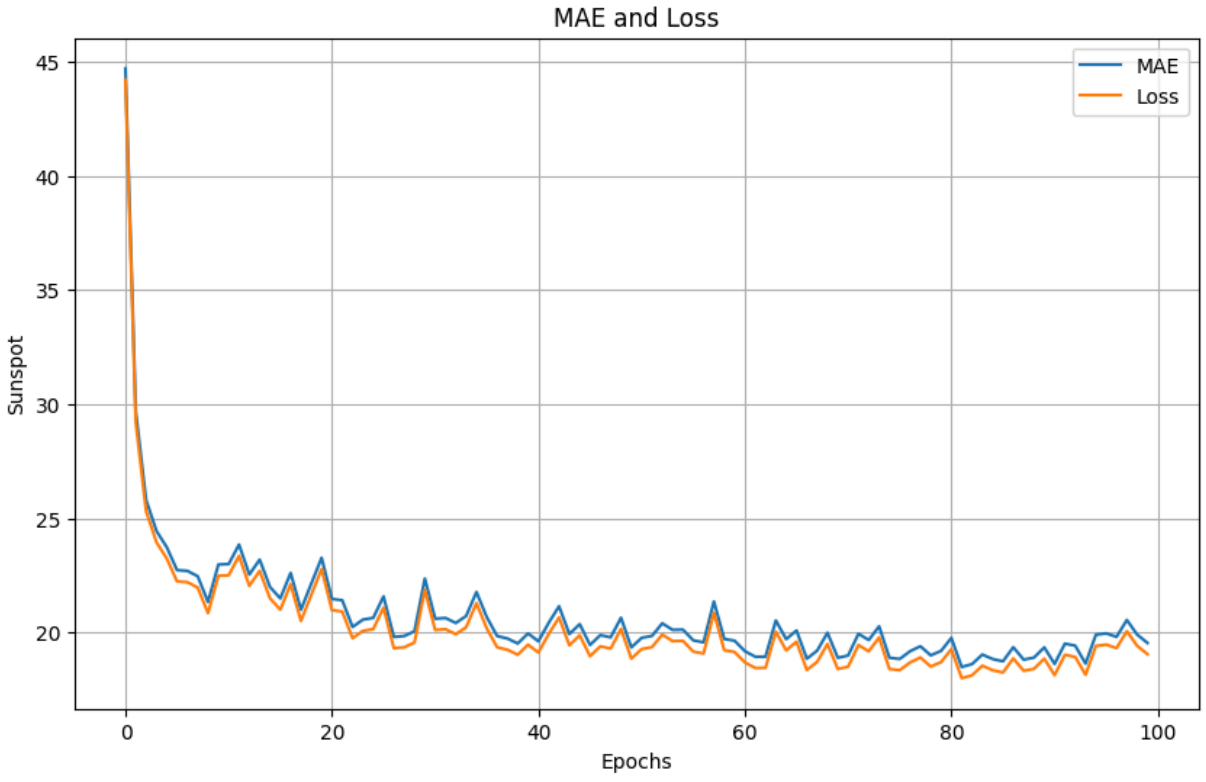
6. Model Evaluation

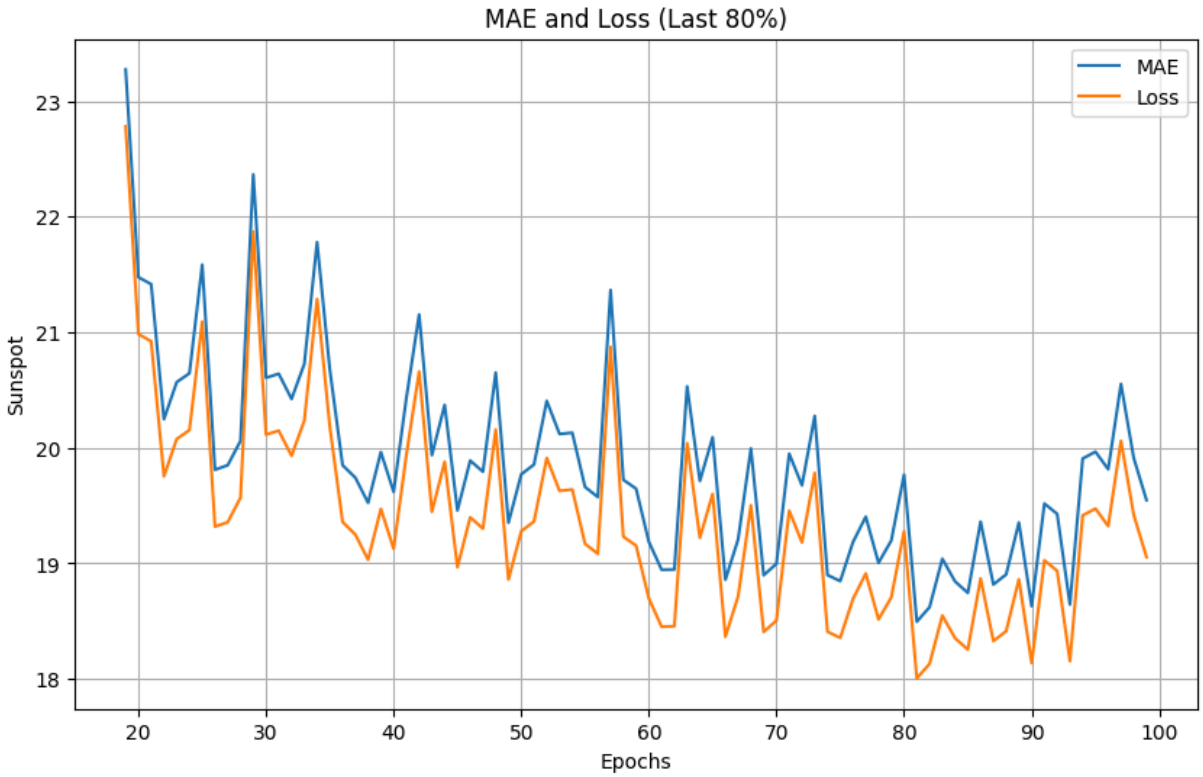
Loss evaluation using Scatter Plot

In [26]:
```
# Plot Function
def visualize_evaluation(history):
  mae=history.history['mae']
  loss=history.history['loss']

  epochs=range(len(loss))

  plot_series(
      x=epochs,
      y=(mae, loss),
      title='MAE and Loss',
      xlabel='Epochs',
      ylabel='Sunspot',
      legend=['MAE', 'Loss'])

  zoom_split = int(epochs[-1] * 0.2)
  epochs_zoom = epochs[zoom_split:]
  mae_zoom = mae[zoom_split:]
  loss_zoom = loss[zoom_split:]

  plot_series(
      x=epochs_zoom,
      y=(mae_zoom, loss_zoom),
      title='MAE and Loss (Last 80%)',
      xlabel='Epochs',
      ylabel='Sunspot',
      legend=['MAE', 'Loss'])
```

RNN Evaluation

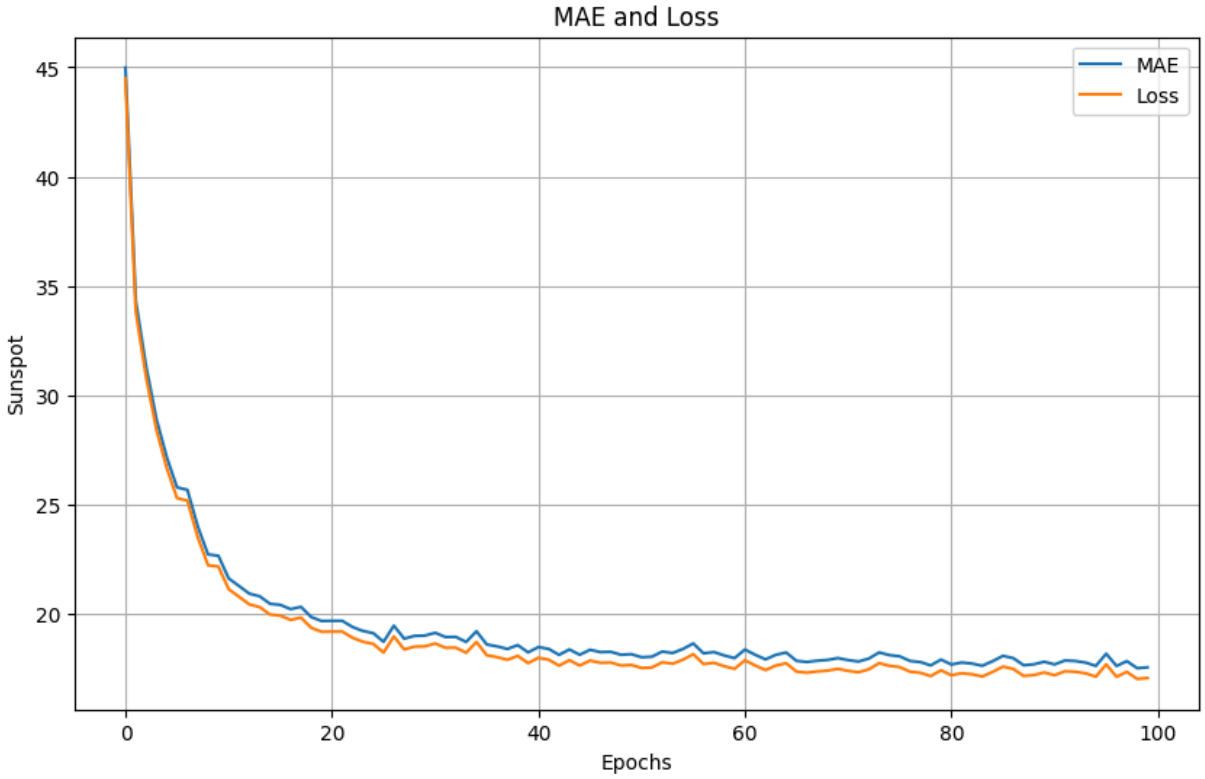
In [27]:
```
visualize_evaluation(history_RNN)
```

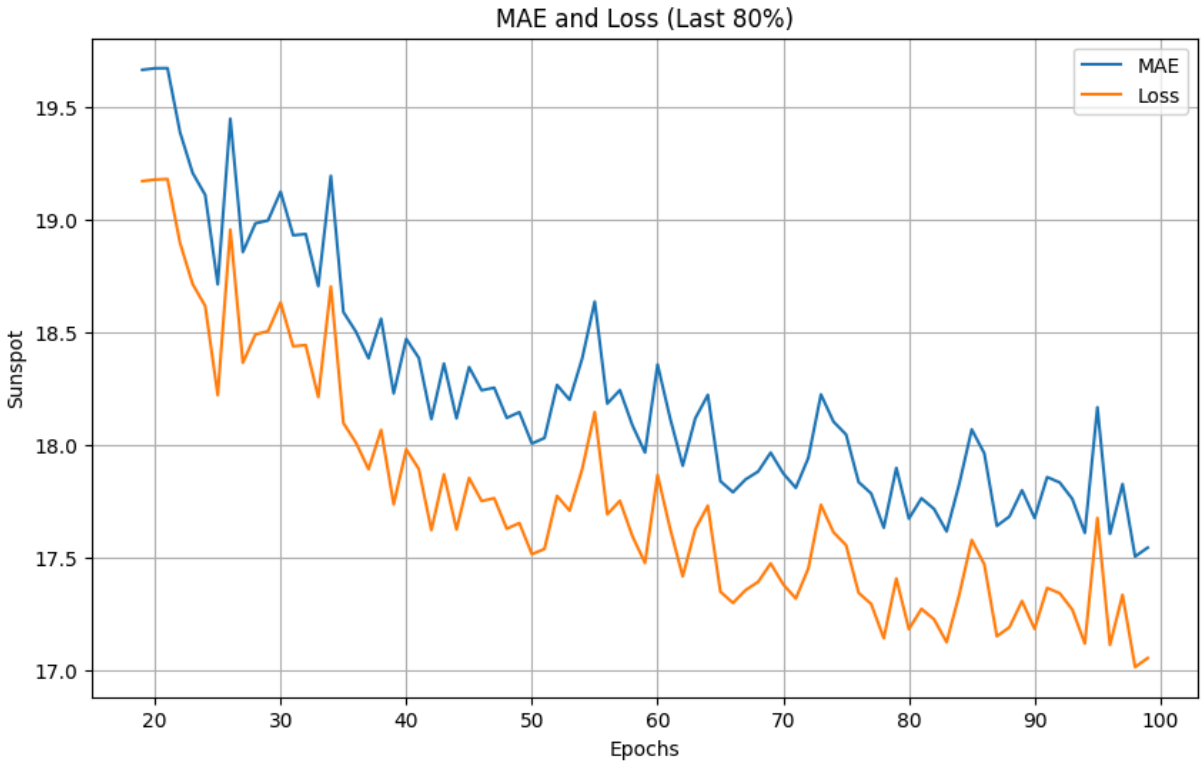




LSTM Evaluation

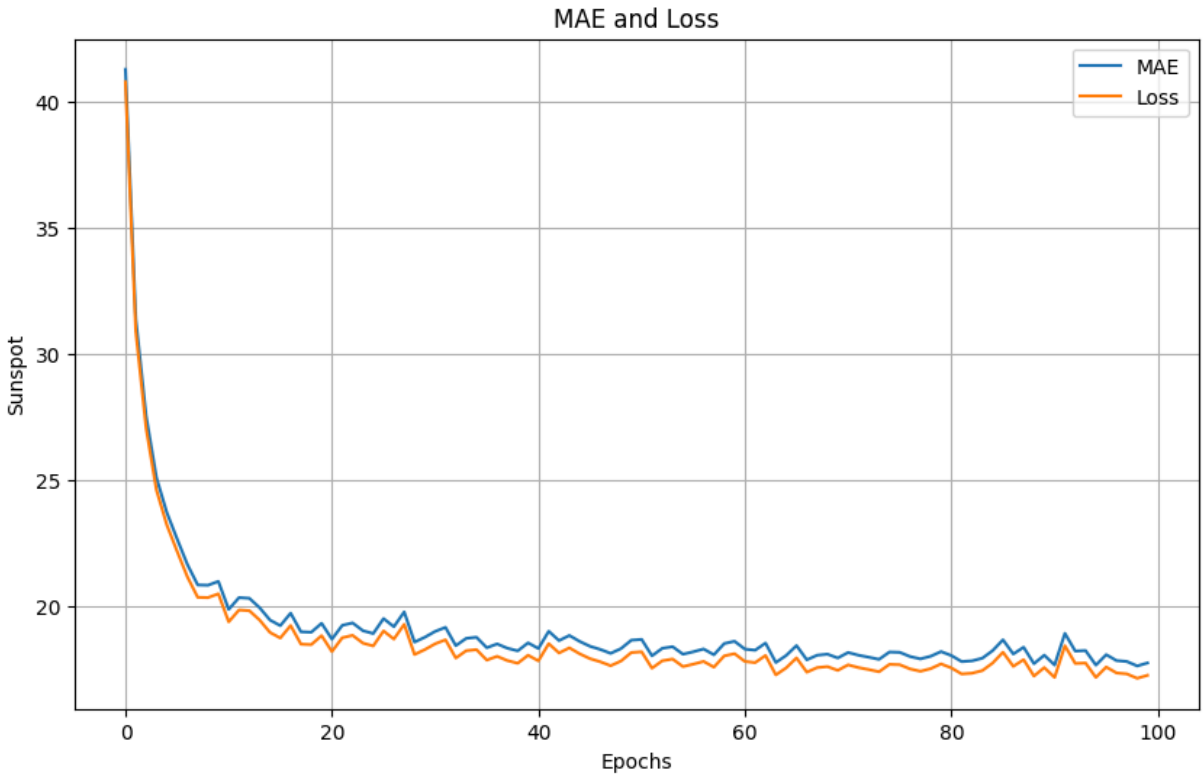
In [28]:
```
visualize_evaluation(history_LSTM)
```





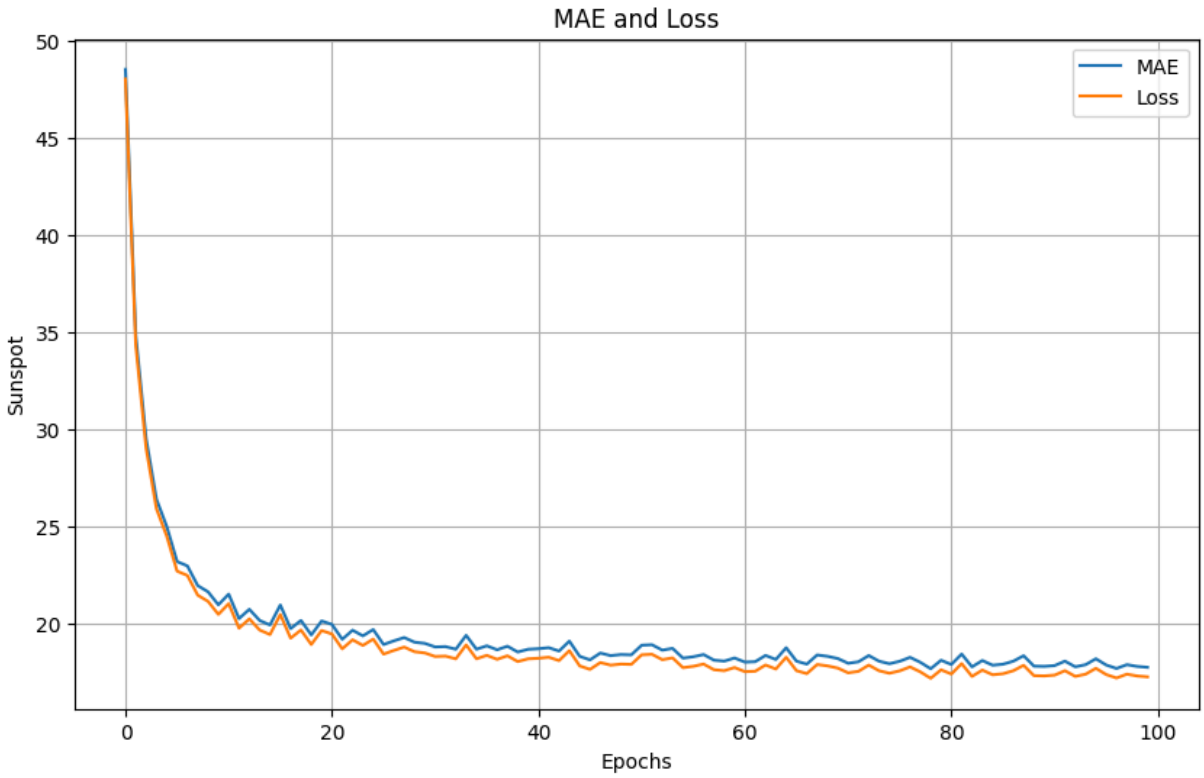
GRU Evaluation

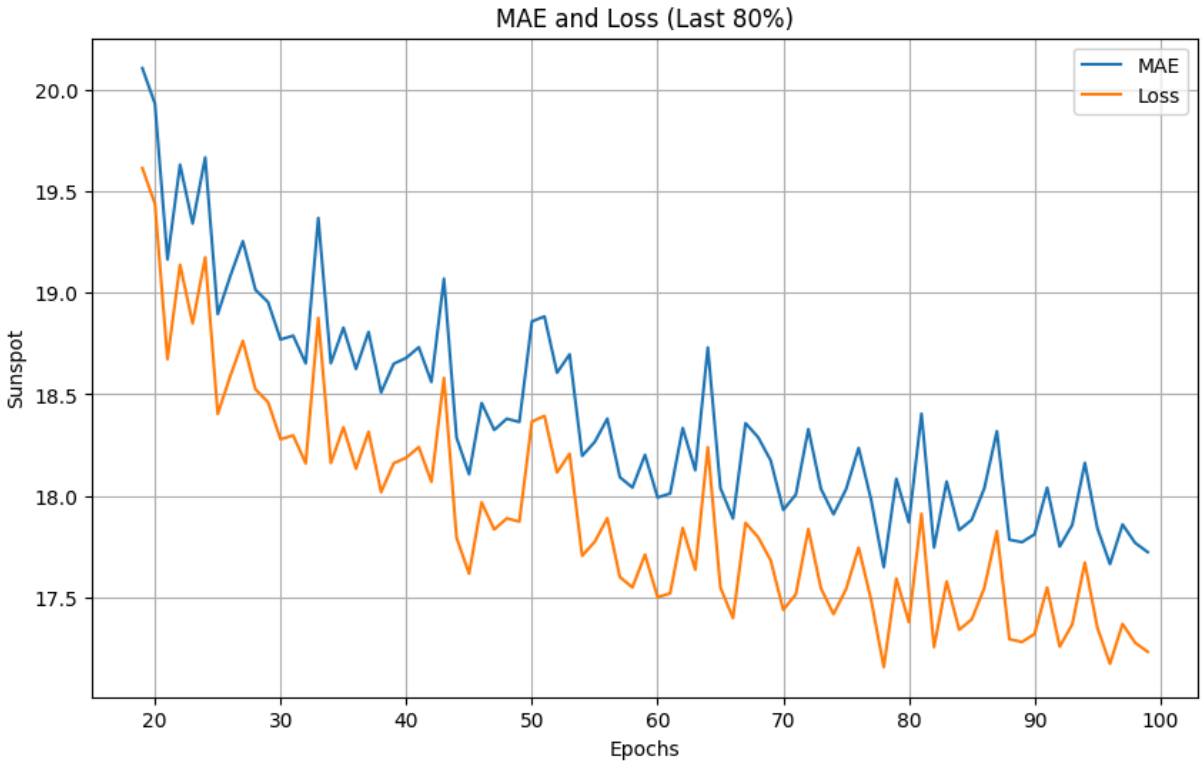
In [29]:
```
visualize_evaluation(history_GRU)
```



RNN-LSTM Evaluation

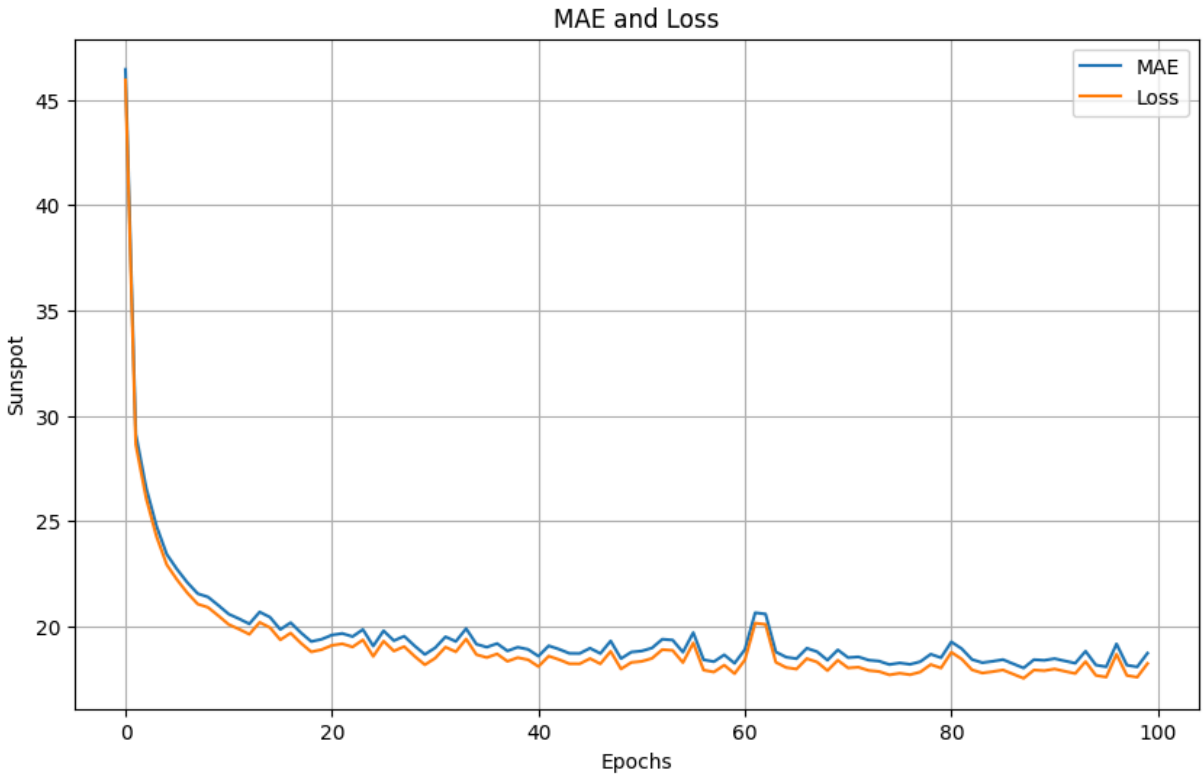
In [30]:
```
visualize_evaluation(history_RNN_LSTM)
```

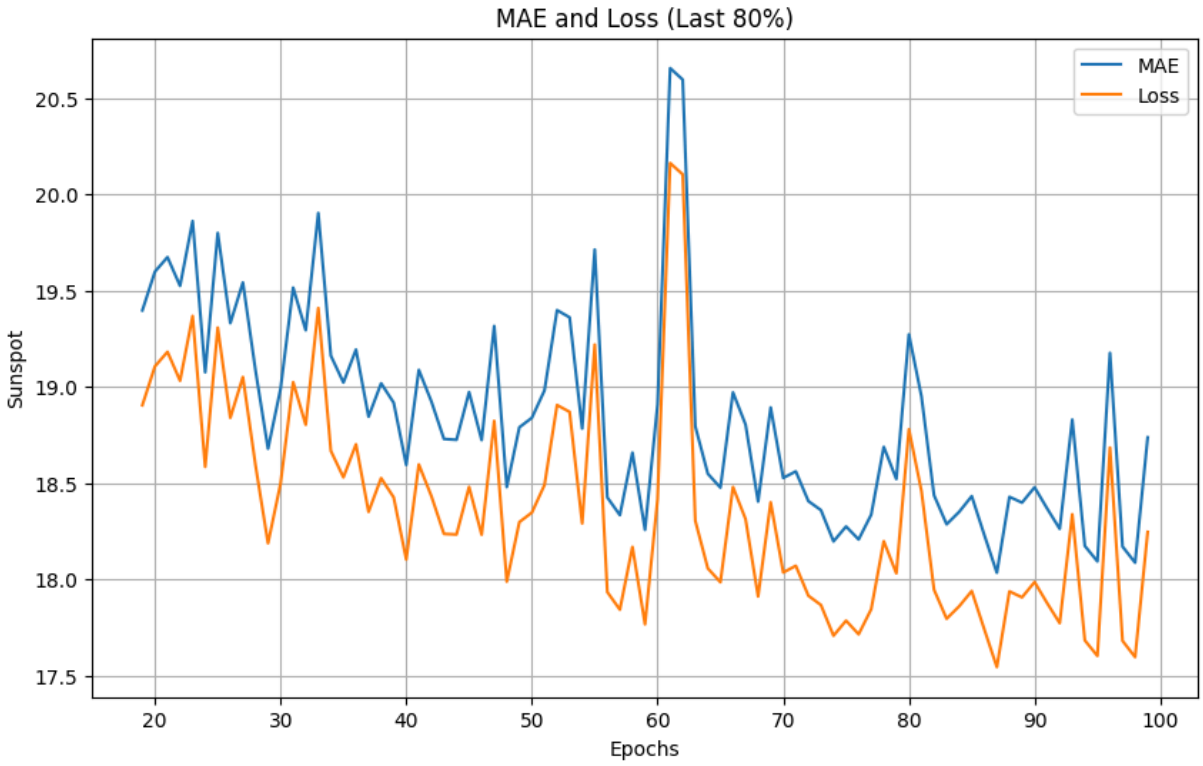




RNN-GRU Evaluation

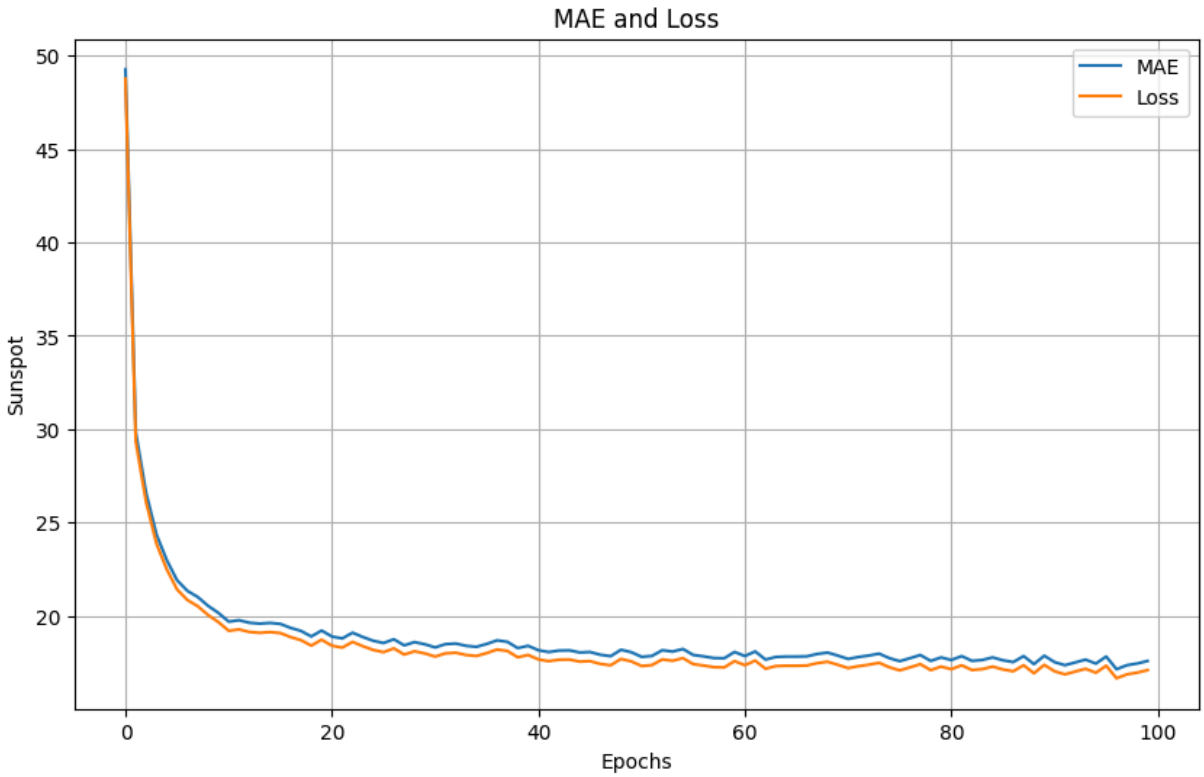
In [31]:
```
visualize_evaluation(history_RNN_GRU)
```

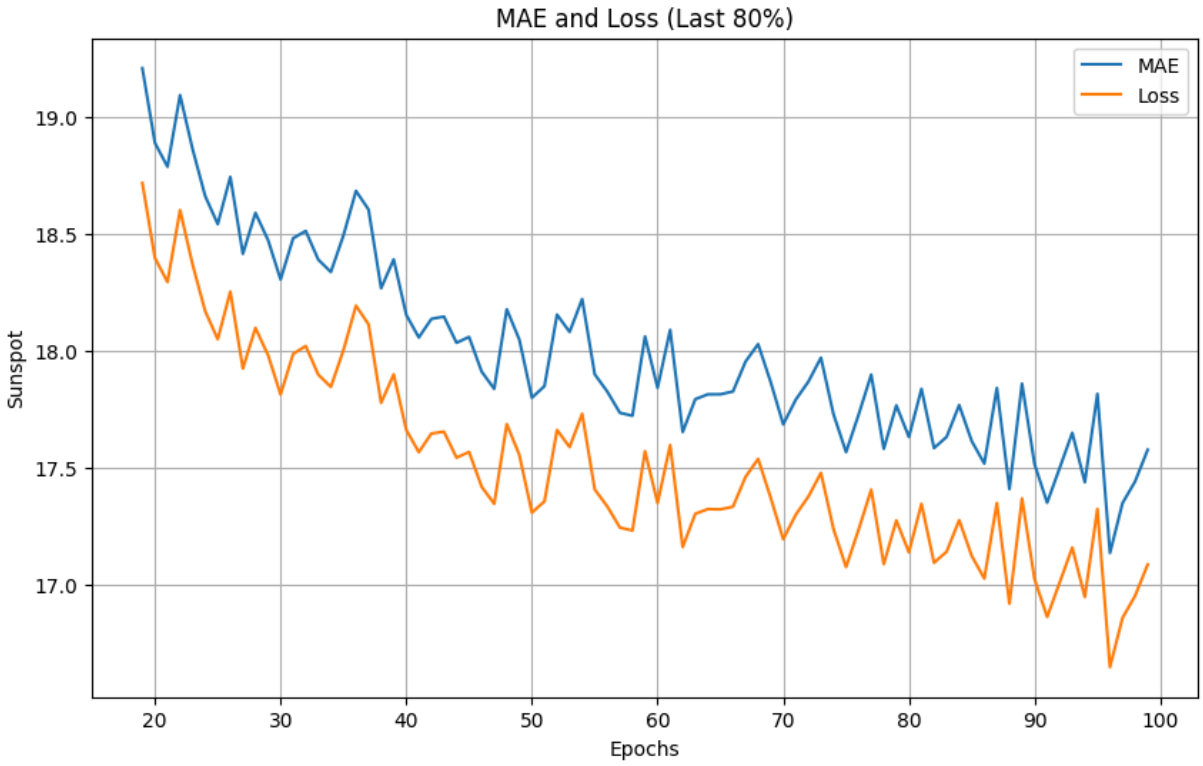




LSTM-RNN Evaluation

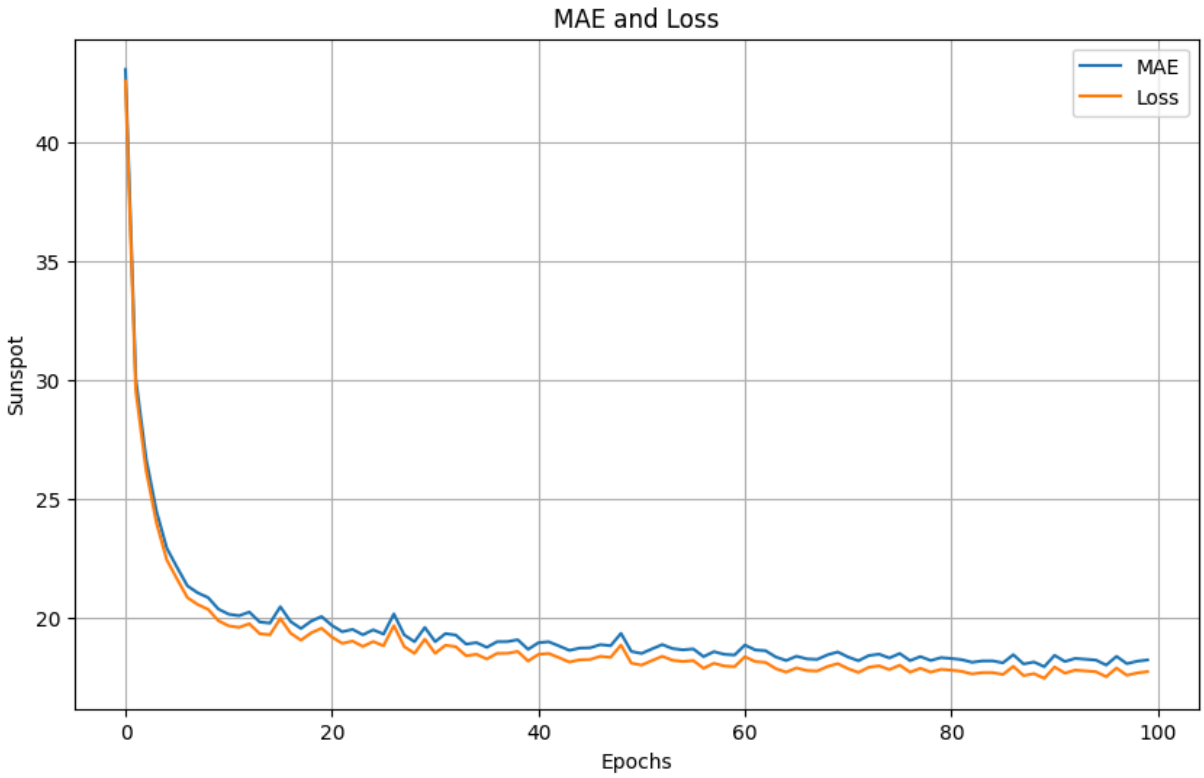
In [32]:
```
visualize_evaluation(history_LSTM_RNN)
```





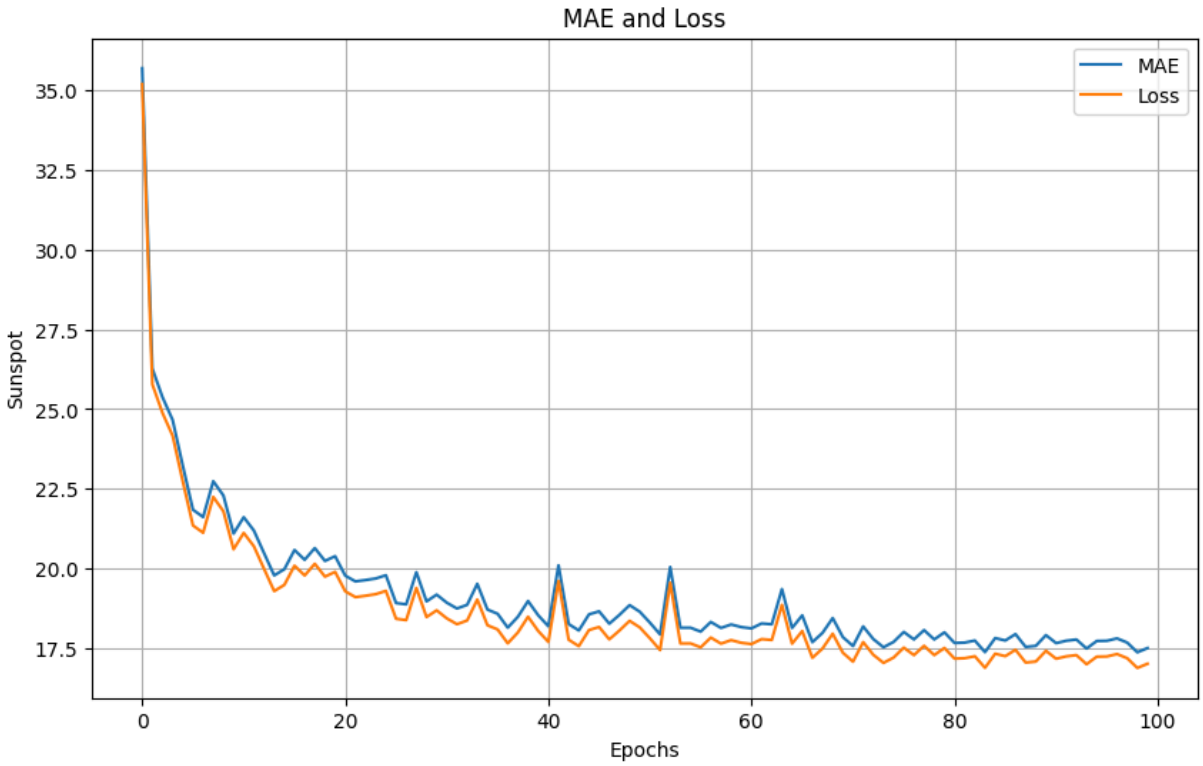
LSTM-GRU Evaluation

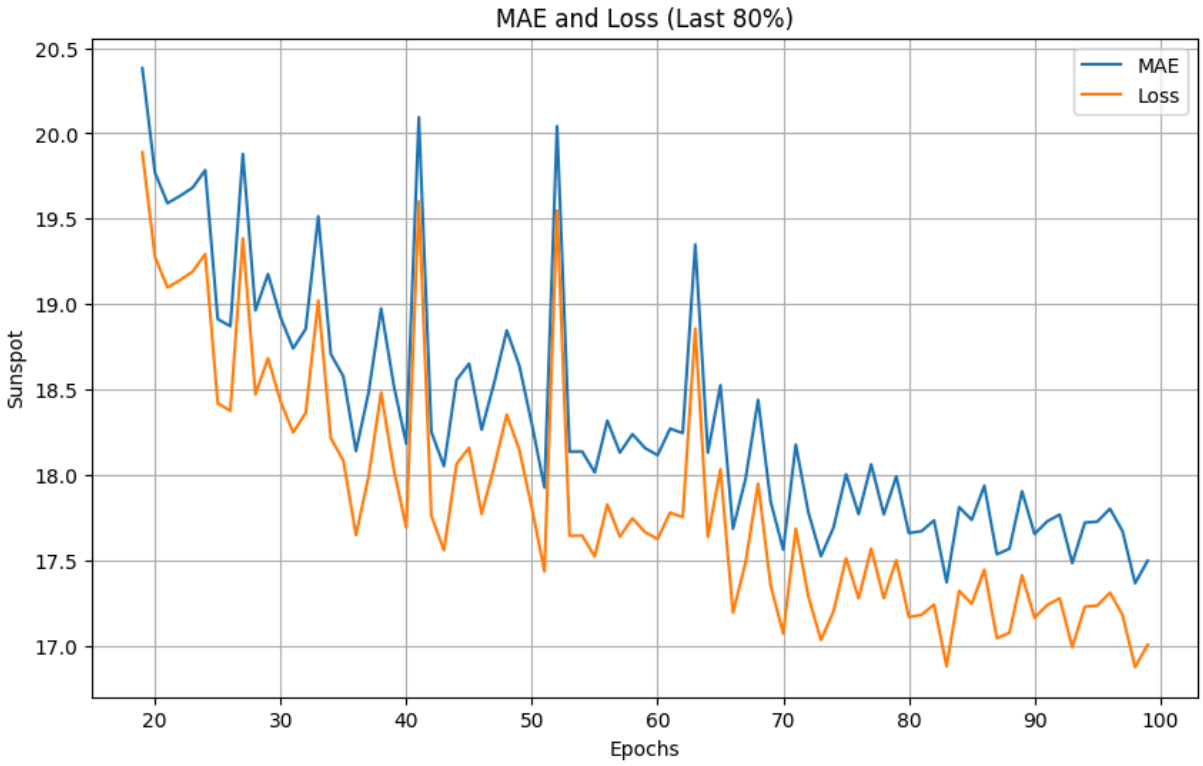
In [33]:
```
visualize_evaluation(history_LSTM_GRU)
```



GRU-RNN Evaluation

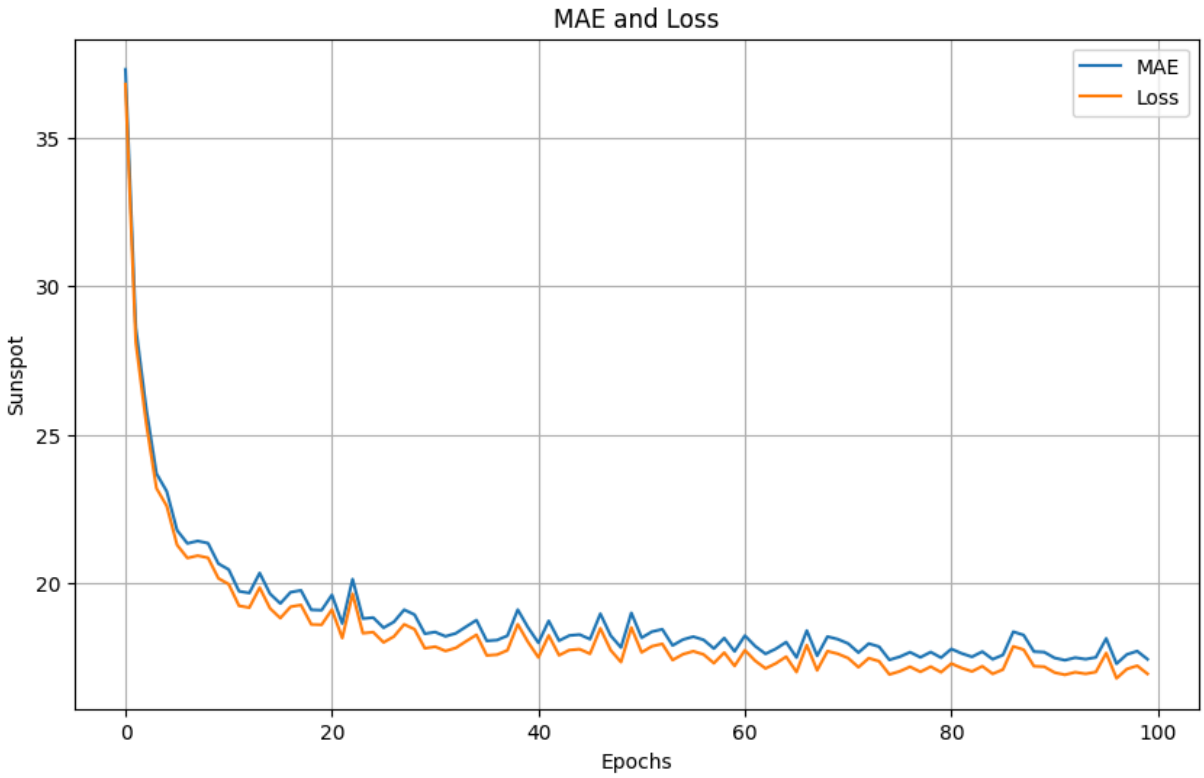
In [34]:
```
visualize_evaluation(history_GRU_RNN)
```

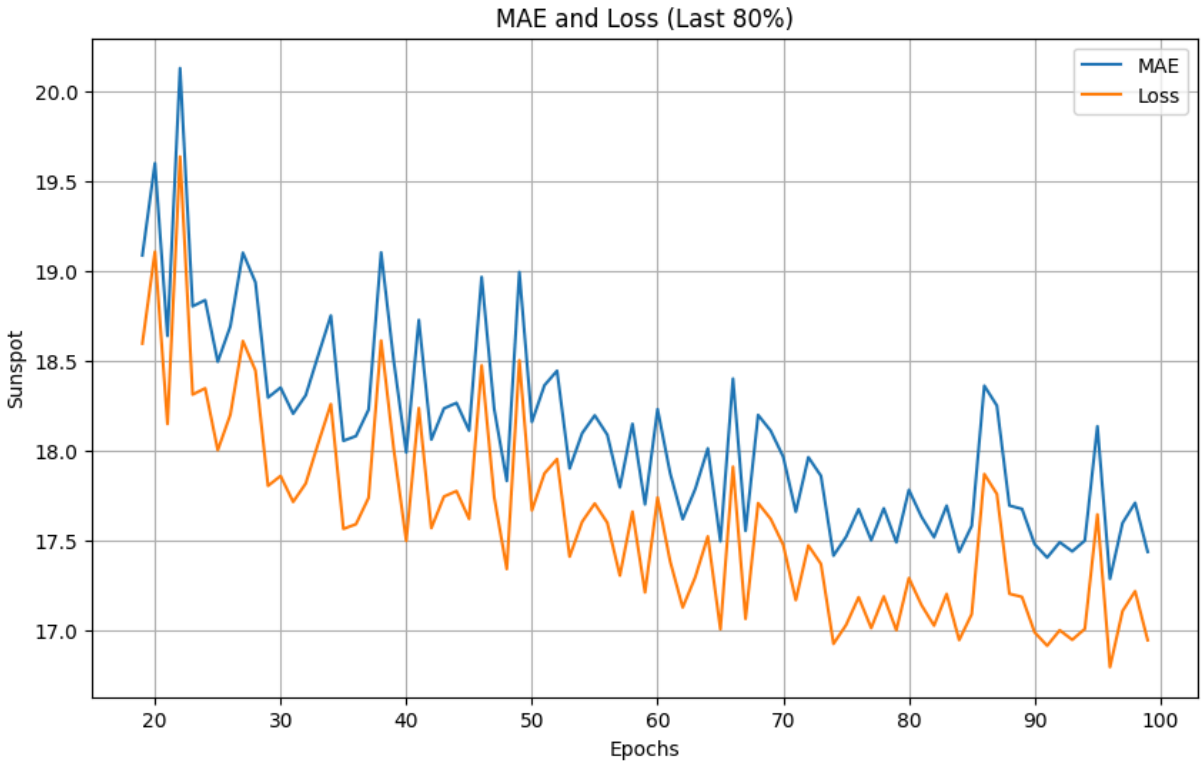




GRU-LSTM Evaluation

In [36]:
```
visualize_evaluation(history_GRU_LSTM)
```





7. Model Prediction

Analysis outcome of clustering for whale transaction characteristic

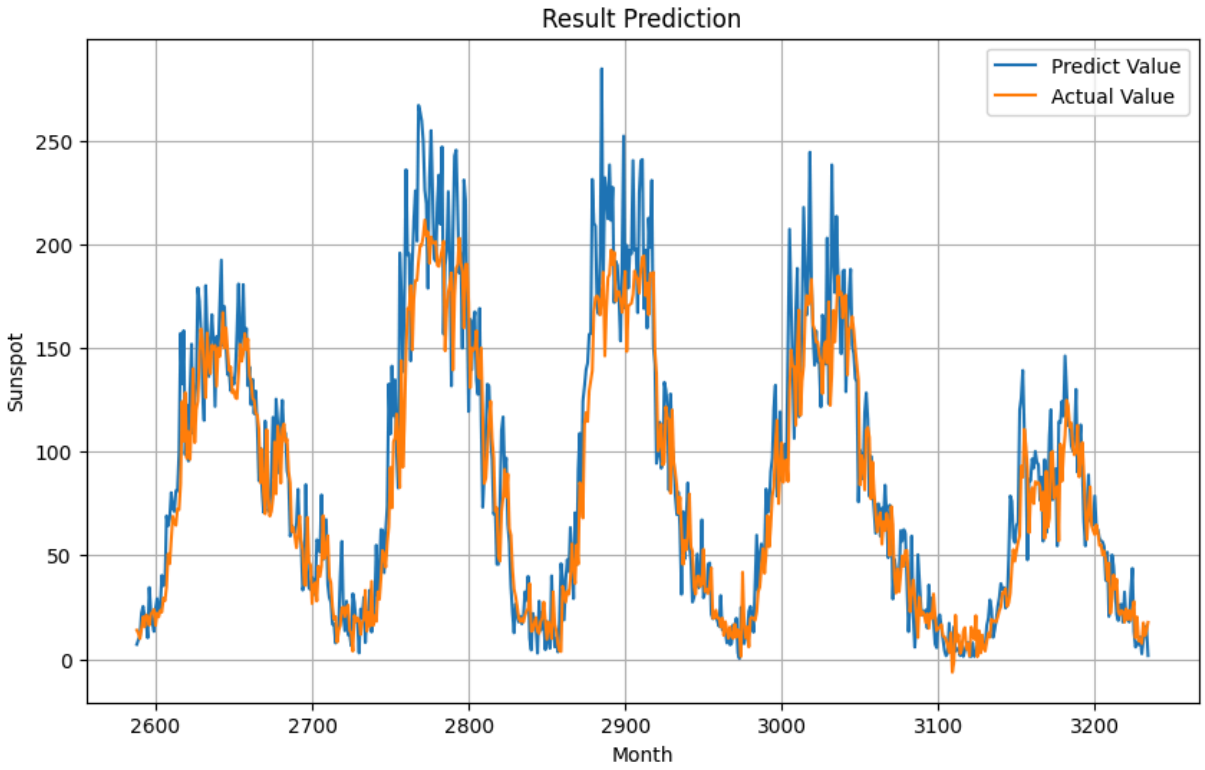
In [37]:
```
def model_forecast(model, series, window_size, batch_size):
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda w: w.batch(window_size))
    dataset = dataset.batch(batch_size).prefetch(1)
    forecast = model.predict(dataset)
    return forecast
```

In [38]:
```
def predict(model):
  forecast_series = series[split_time-window_size:-1]
  forecast = model_forecast(model, forecast_series, window_size, batch_size)
  results = forecast.squeeze()
  plot_series(time_valid, (x_valid, results, ), title='Result Prediction',
  xlabel='Month',
  ylabel='Sunspot',
  legend=['Predict Value', 'Actual Value'])
```

RNN Prediction

In [39]:
```
predict(model_RNN)
```

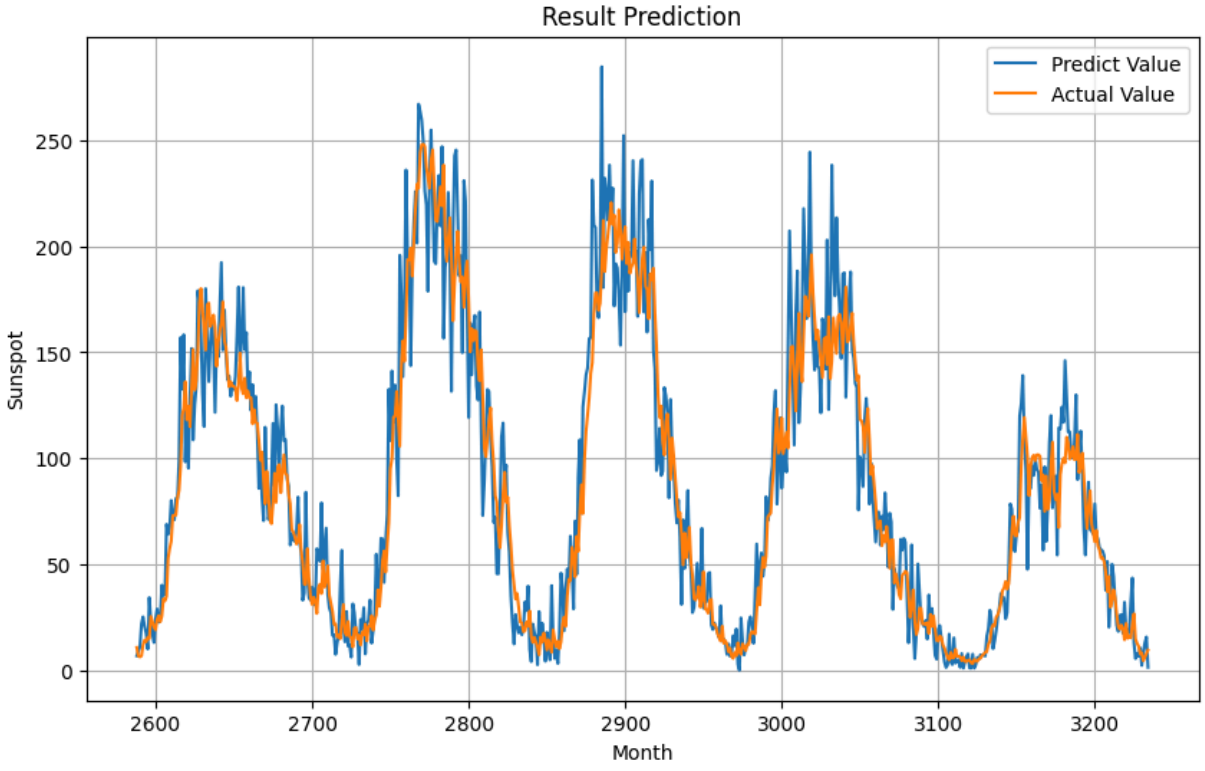
```
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step
```



LSTM Prediction

In [40]:
```
predict(model_LSTM)
```

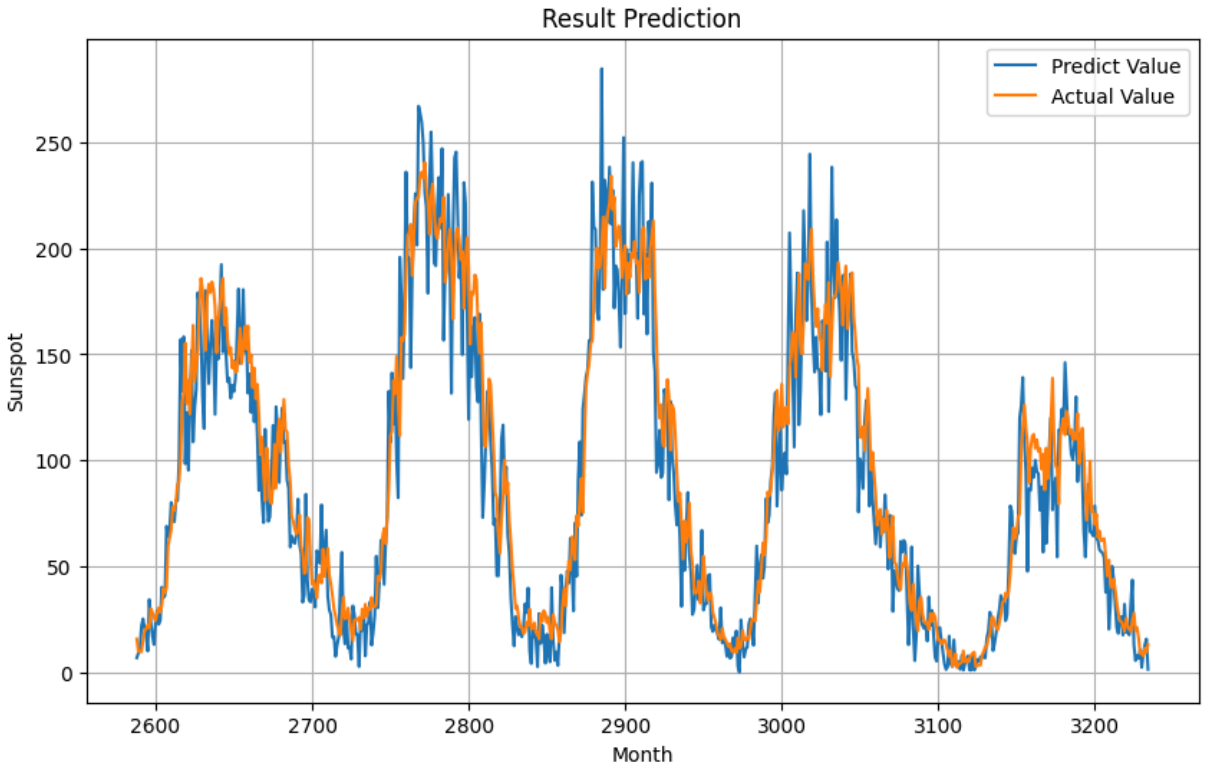
```
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step
```



GRU Prediction

In [41]:
```
predict(model_GRU)
```

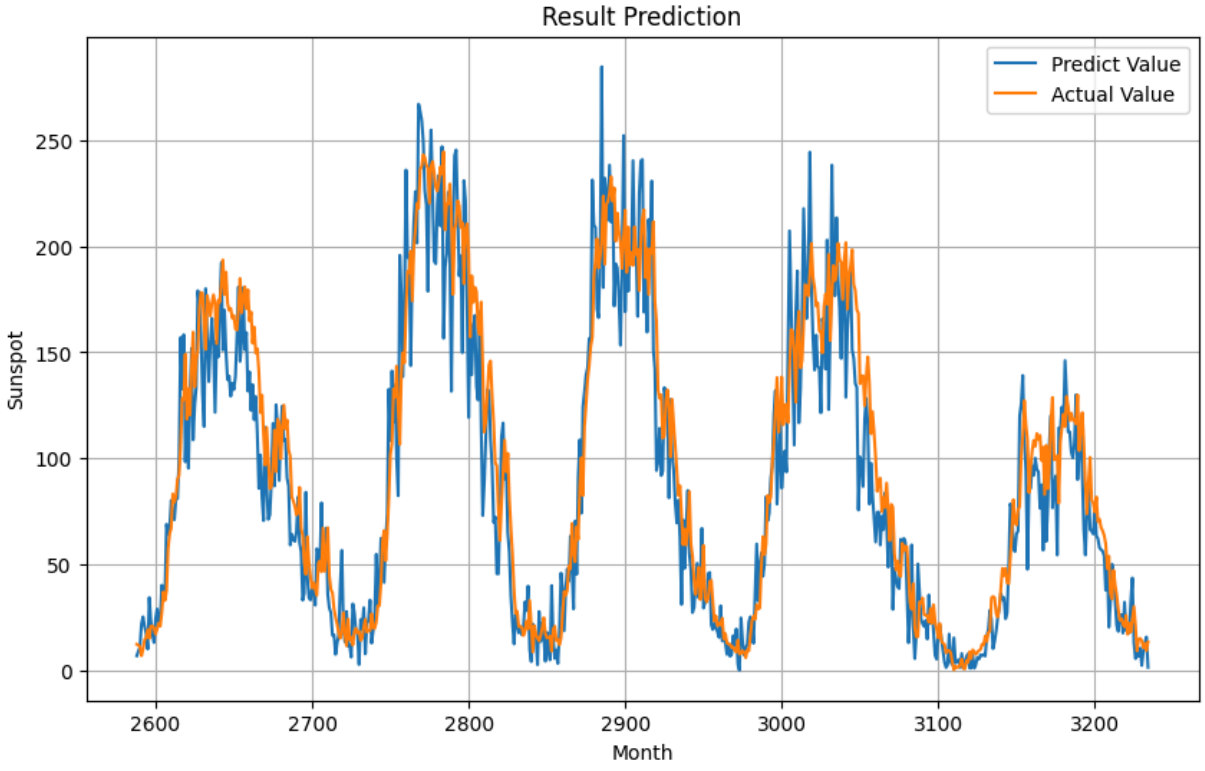
```
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step
```



RNN-LSTM Prediction

In [42]:
```
predict(model_RNN_LSTM)
```

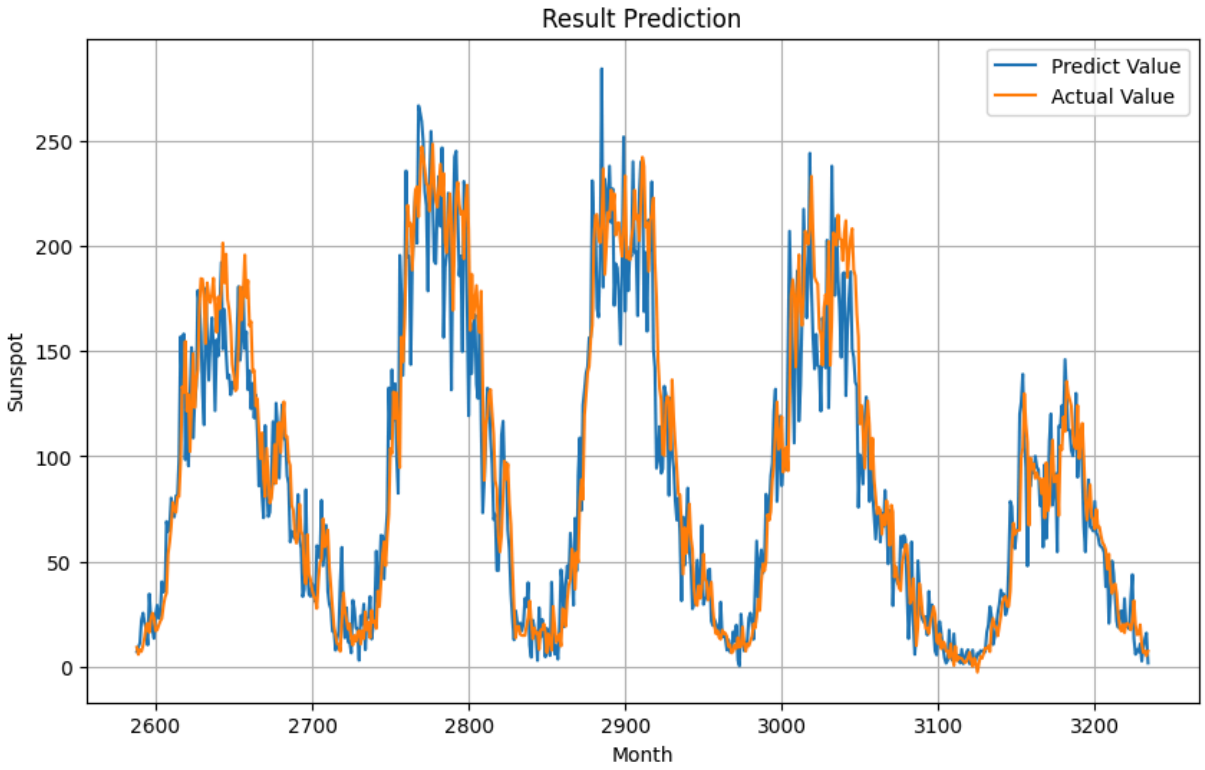
```
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step
```



RNN-GRU Prediction

In [43]:
```
predict(model_RNN_GRU)
```

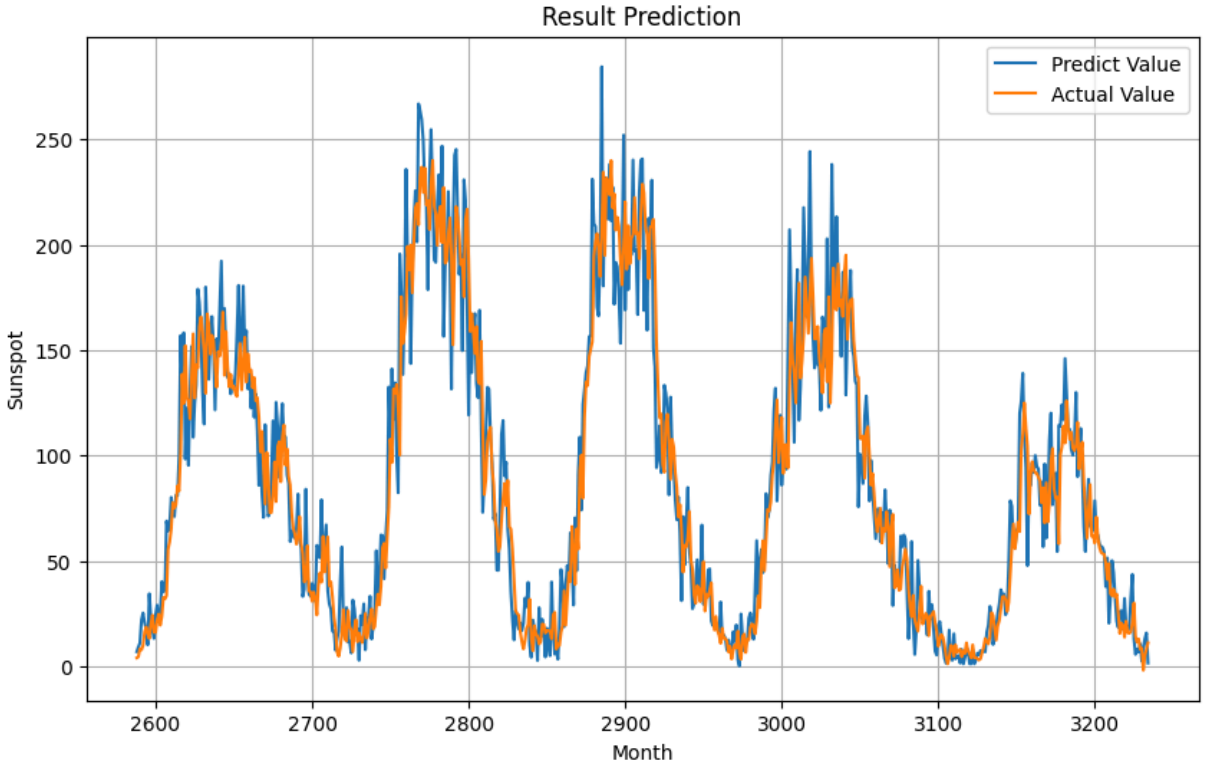
```
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step
```



LSTM-RNN Prediction

In [44]:
```
predict(model_LSTM_RNN)
```

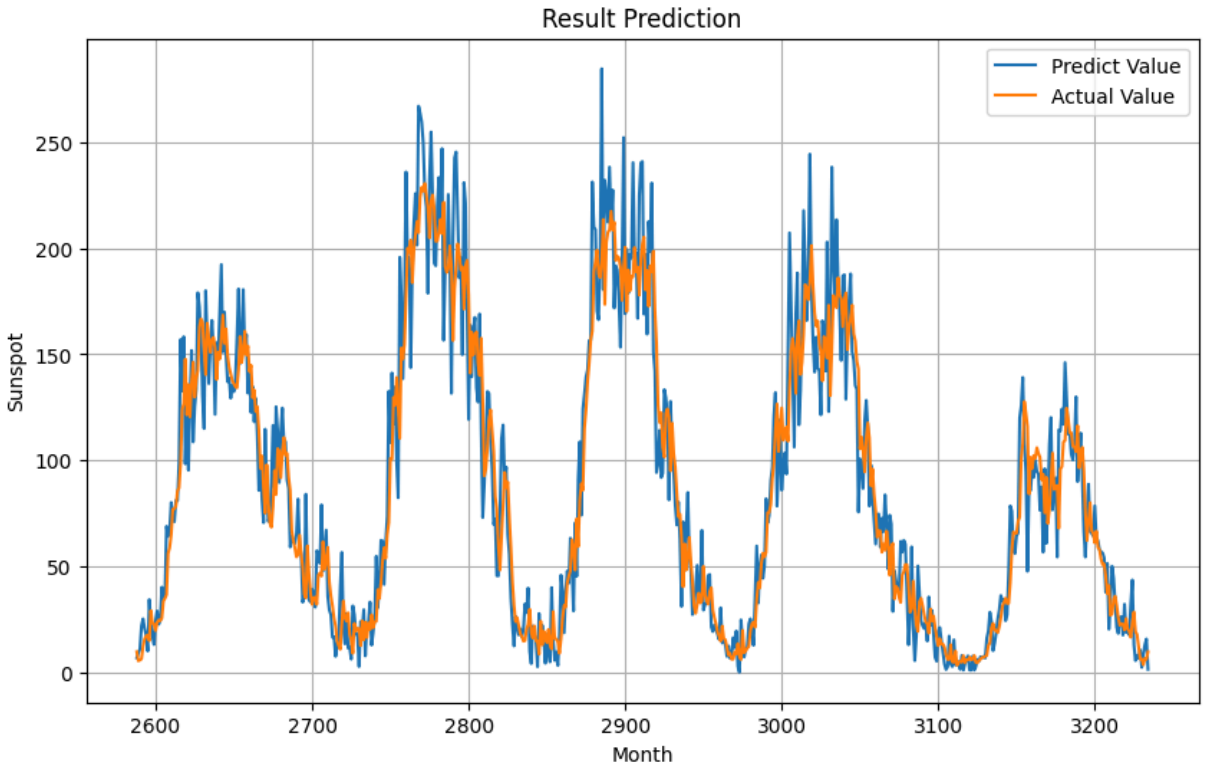
```
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step
```



LSTM-GRU Prediction

In [45]:
```
predict(model_LSTM_GRU)
```

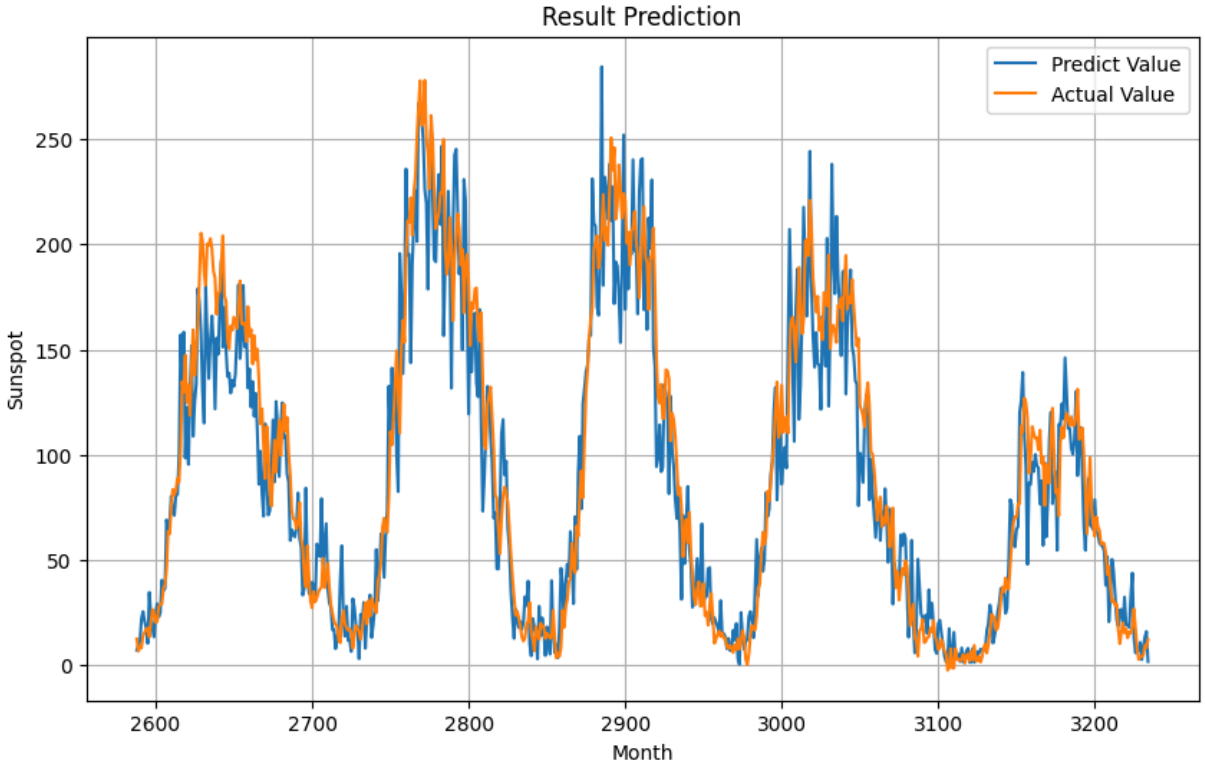
```
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step
```



GRU-RNN Prediction

In [46]:
```
predict(model_GRU_RNN)
```

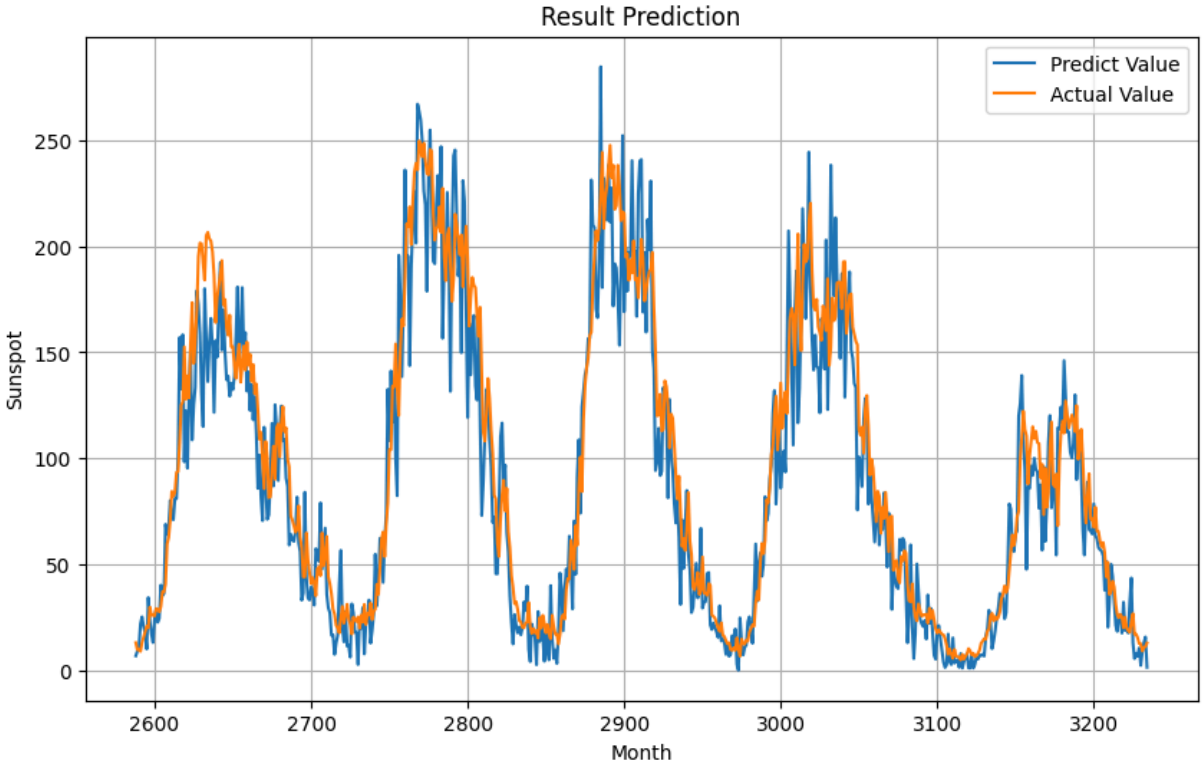
```
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step
```



GRU-LSTM Prediction

In [47]:
```
predict(model_GRU_LSTM)
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step
```


Result Prediction
Predict Value
Actual Value
Sunspot
Month
0
50
100
150
200
250
2600
2700
2800
2900
3000
3100
3200

Confident Interval

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import files
import time
```

Donwload Data

In [ ]:
```
!gdown 1tPJuX6Fe-PXCxzLE-xw9jxnWwUErcA4F
!gdown 1wFyMh0P9bT08fdaB1AoVv6tDvPSuYY8d
!gdown 1HmdOvr2q03rC4w7qEAdNqYWjJK7gXhHD
!gdown 1JkwqKnS0ubAns2XJH_A8lHbOE8-ThOhx
!gdown 1JnE3zsGCpFG9eFK4c9meG7Ox0zF7p5Ao
!gdown 10s6qULX0pCdZsXqiVVaqHoyUuOdlzzls
!gdown 1cG7s3oX6jHgKcTdfAJ8J1O1XKTq38KeN
!gdown 1CKUWag2iDSmHlYdaYiGIFd6_E8zjli7c
!gdown 1p-EagHjp7NyEVGNXM3mStH5QAXGpN-gI
```

In [ ]:
```
!gdown 1IZId_e5nCMviLsGv9aM3nyaDQmEqYjLq
!gdown 13H7TwY8WZLF6Xd9I0ZZtS1orAGhKM4lC
!gdown 1nyVHyU8D7vRyMBXsQ4xQxcGaw1uSiCIK
!gdown 1fnLwDBh8FfROEK9JJsJsUMWb4_FST0Mh
!gdown 1kXFAubzTacey_WrduvpAkysM8N9iLw6F
!gdown 1Kyt7pNxSQpPjQCRxbMzqUNuYjv8qASnM
!gdown 1B4fj3dEbEeByqF1vUs-r2bQNJ2jFScou
!gdown 1_-e4A9LH13DVTbNLz-cFbunZo1iuYMHB
!gdown 11Mr2-77o-UwP5uxI3n5VwcJlWZac8-hx
```

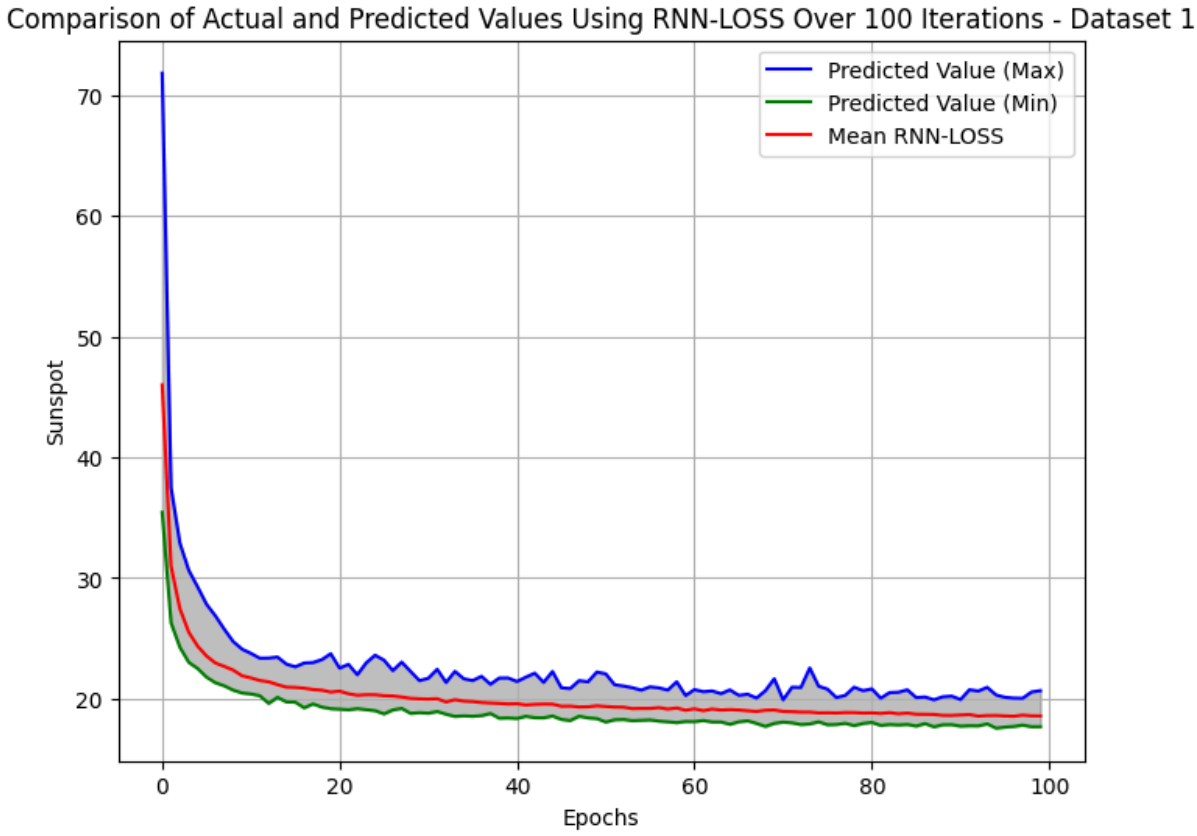
Viz

In [ ]:
```
from re import X
def shade_plot(temp, x_valid='0', line='MAE', predict=''):
  import matplotlib.pyplot as plt

  if predict == '':
    redline = f'Mean {line}'
    axis = 'Epochs'
  else:
    redline = 'Actual Value'
    axis = 'Days'

  #SORT
  temp = temp.apply(lambda x: x.sort_values().reset_index(drop=True))

  #CUT 2.5%
  n = int(len(temp) * 0.025)
  temp = temp.iloc[n:-n]

  batas_atas = temp.max().tolist()
  batas_bawah = temp.min().tolist()
  rata = temp.mean().tolist()

  # Replace these lists with your actual data
  # Dataset 1
  x1 = [i for i in range(0,len(batas_atas))]
  y1 = batas_atas

  # Dataset 2
  x2 = [i for i in range(0,len(batas_bawah))]
  y2 = batas_bawah

  if(x_valid=='0'):
```

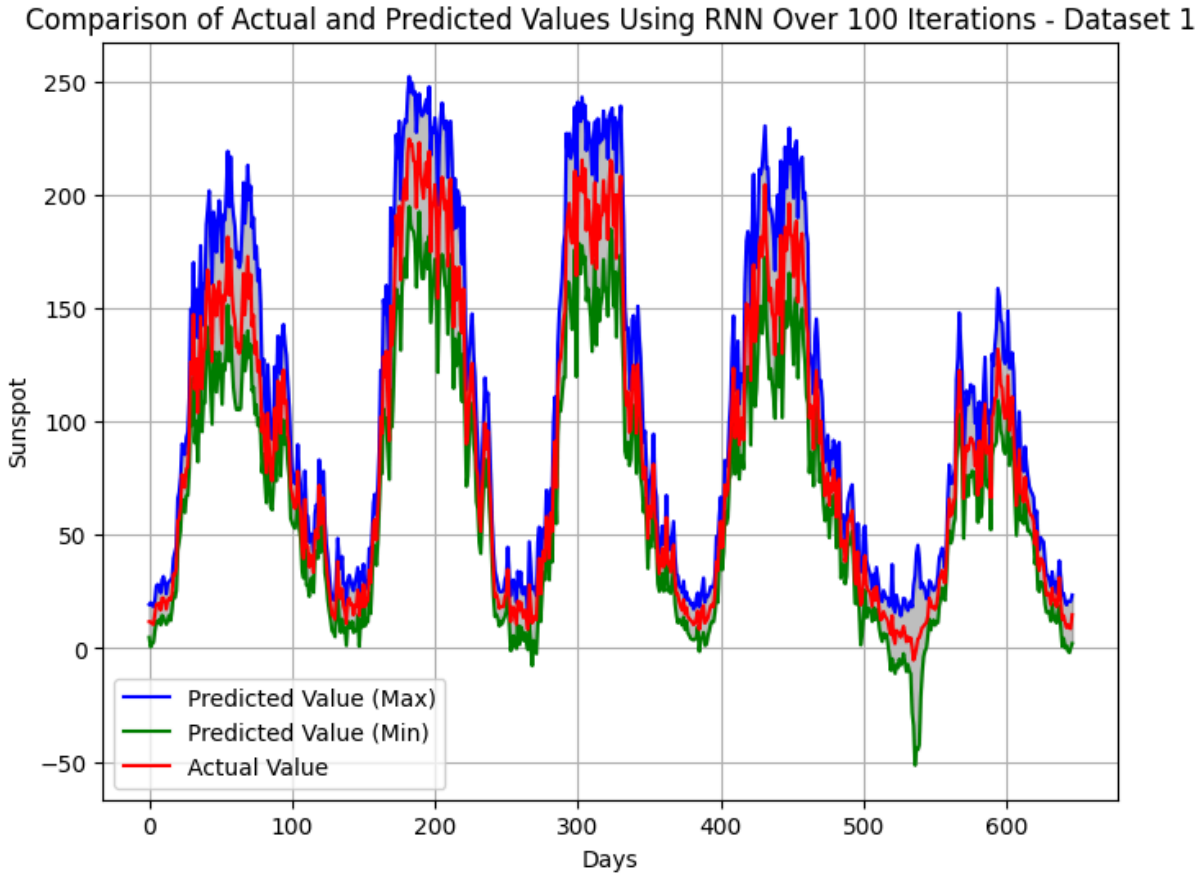
```
    # Dataset 3
    x3 = [i for i in range(0,len(rata))]
    y3 = rata
  else:
    x3 = [i for i in range(0,len(x_valid))]
    y3 = x_valid

  # Create the plot
  plt.figure(figsize=(8, 6))

  # Plot the first linear dataset
  plt.plot(x1, y1, label=f'Predicted Value (Max)', color='blue')

  # Plot the second linear dataset
  plt.plot(x2, y2, label=f'Predicted Value (Min)', color='green')

  plt.plot(x3, y3, label=f'{redline}', color='red')

  # Create a mask for the shaded area where y1 >= y2
  mask = [y1_val >= y2_val for y1_val, y2_val in zip(y1, y2)]

  # Shade the area between the two datasets
  plt.fill_between(x1, y1, y2, where=mask, interpolate=True, color='gray', alpha=0.5)

  # Add labels and legend
  plt.xlabel(f'{axis}')
  plt.ylabel('Sunspot')
  plt.title(f'Comparison of Actual and Predicted Values Using {line} Over 100 Iterations - Data
  plt.legend()
  plt.savefig(f"Predicted_Actual_Value_Dataset1_{line}.pdf", dpi=300)
  files.download(f"Predicted_Actual_Value_Dataset1_{line}.pdf")
  # Show the plot
  plt.grid(True)
  plt.show()
```

RNN

In [ ]:
```
temp = pd.read_excel('RNN_LOSS.xlsx')
shade_plot(temp, line = 'RNN-LOSS')
```

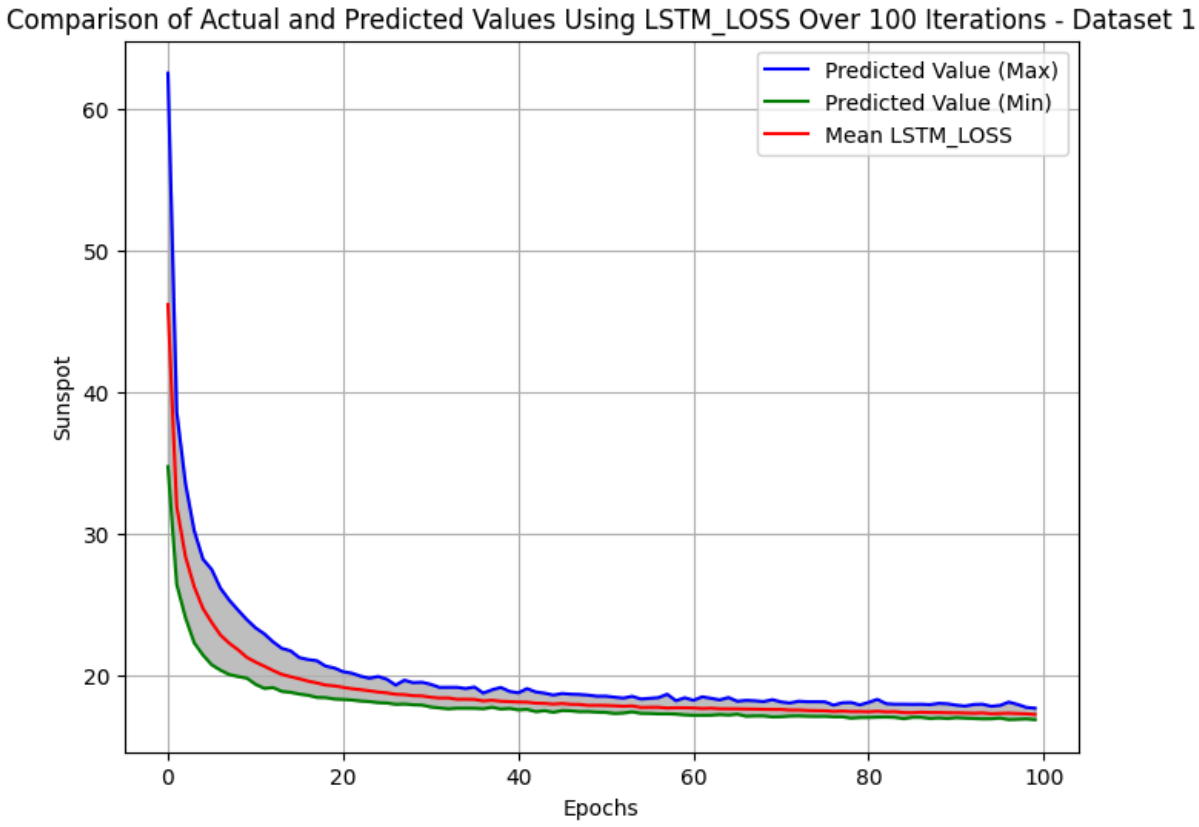


In [ ]:
```
temp = pd.read_excel('RNN_PREDICT.xlsx')
shade_plot(temp, line='RNN', predict='YES')
```

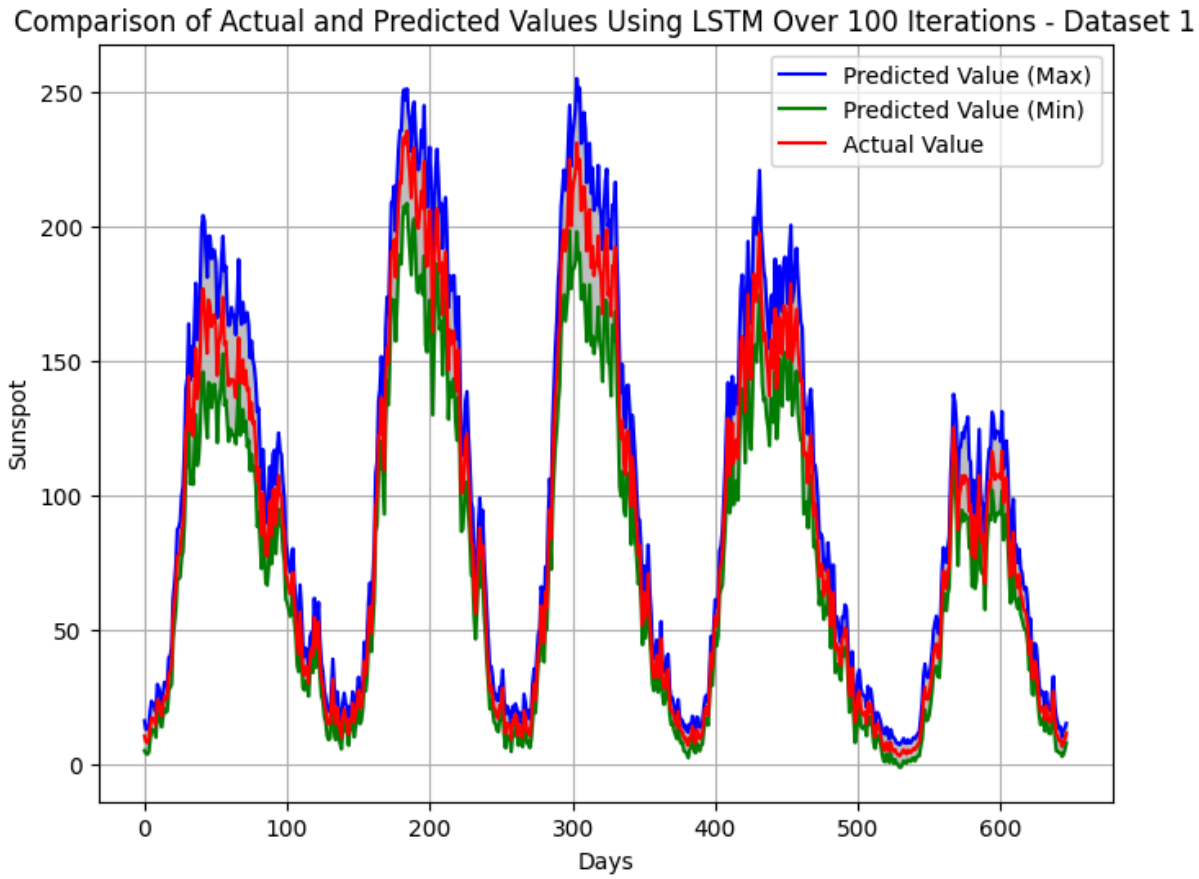


LSTM

In [ ]:
```
temp = pd.read_excel('LSTM_LOSS.xlsx')
shade_plot(temp, line = 'LSTM_LOSS')
```

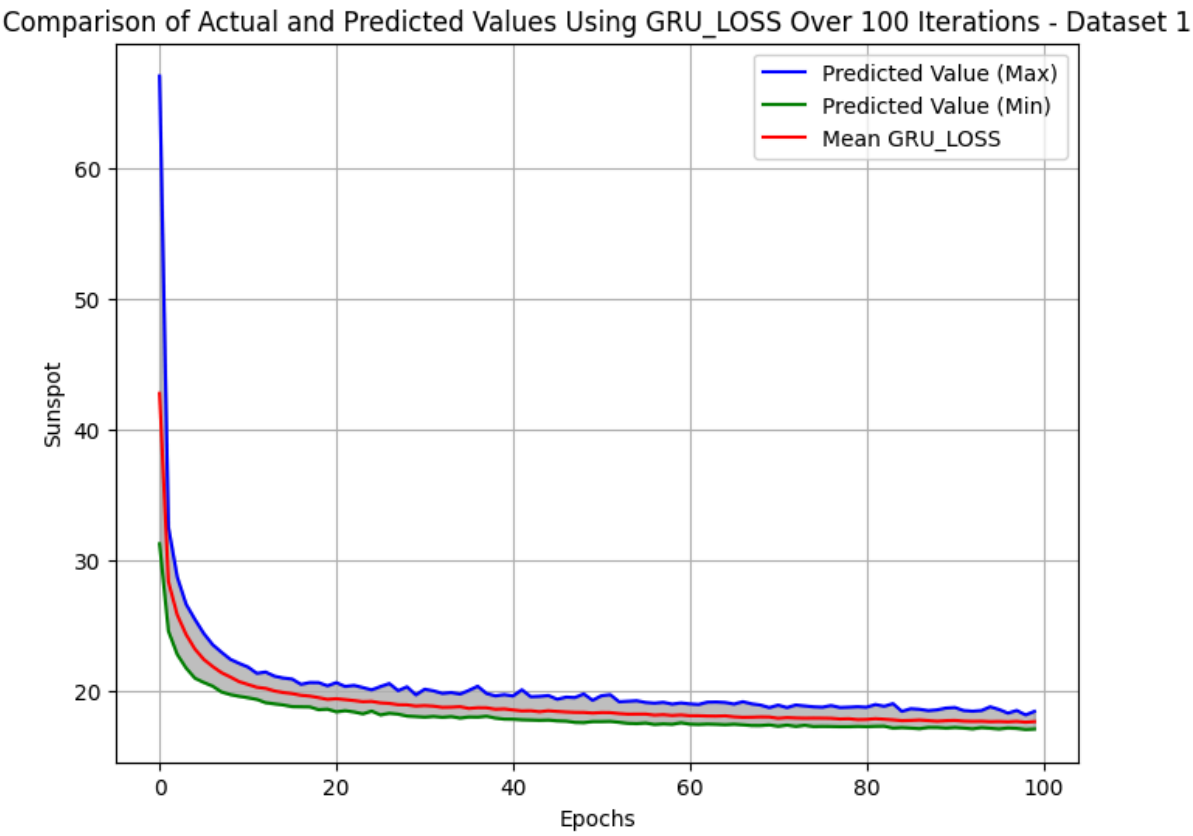


In [ ]:
```
temp = pd.read_excel('LSTM_PREDICT.xlsx')
shade_plot(temp, line='LSTM', predict='YES')
```

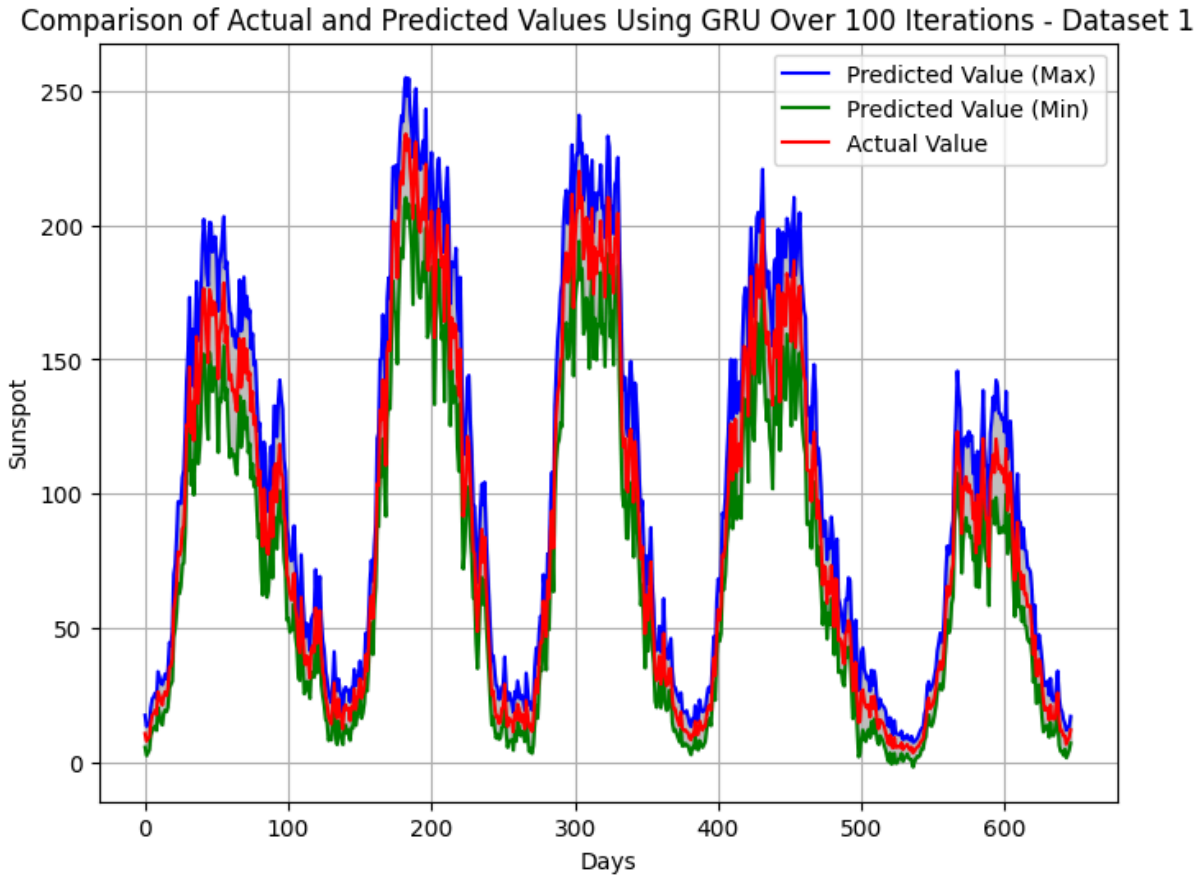


GRU

In [ ]:
```
temp = pd.read_excel('GRU_LOSS.xlsx')
shade_plot(temp, line = 'GRU_LOSS')
```



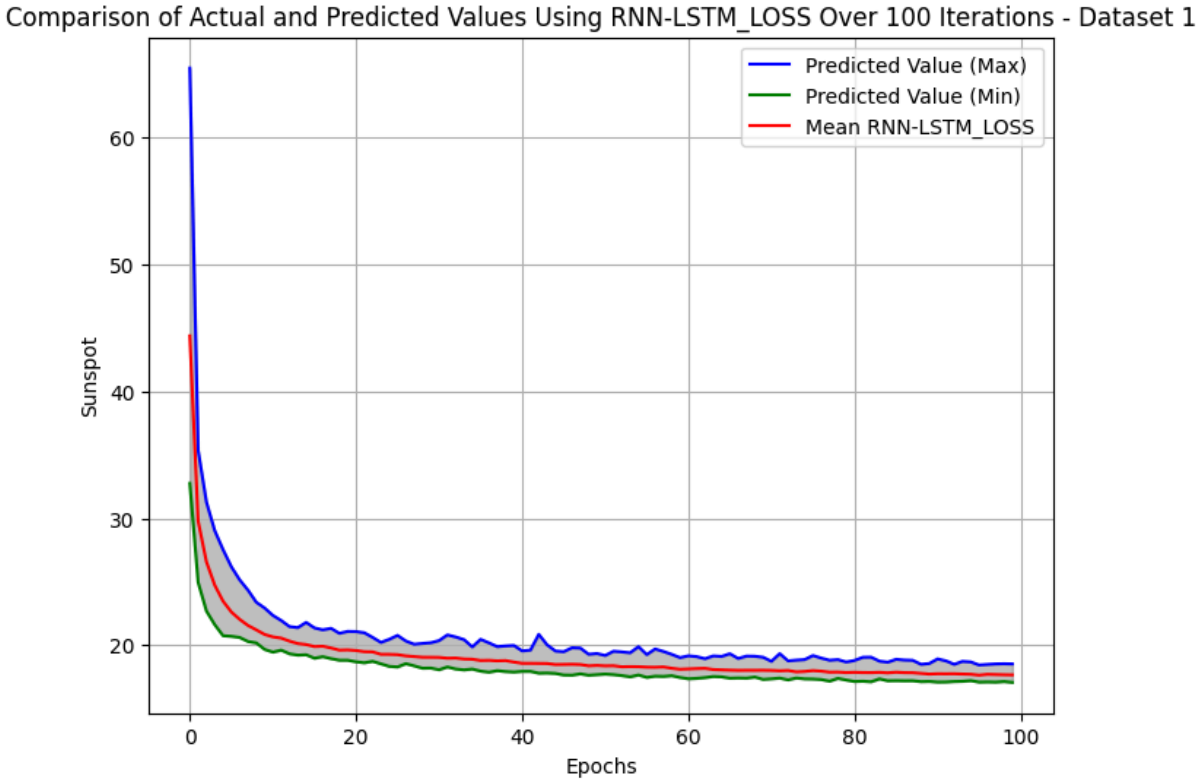
In [ ]:
```
temp = pd.read_excel('GRU_PREDICT.xlsx')
shade_plot(temp, line='GRU', predict='YES')
```



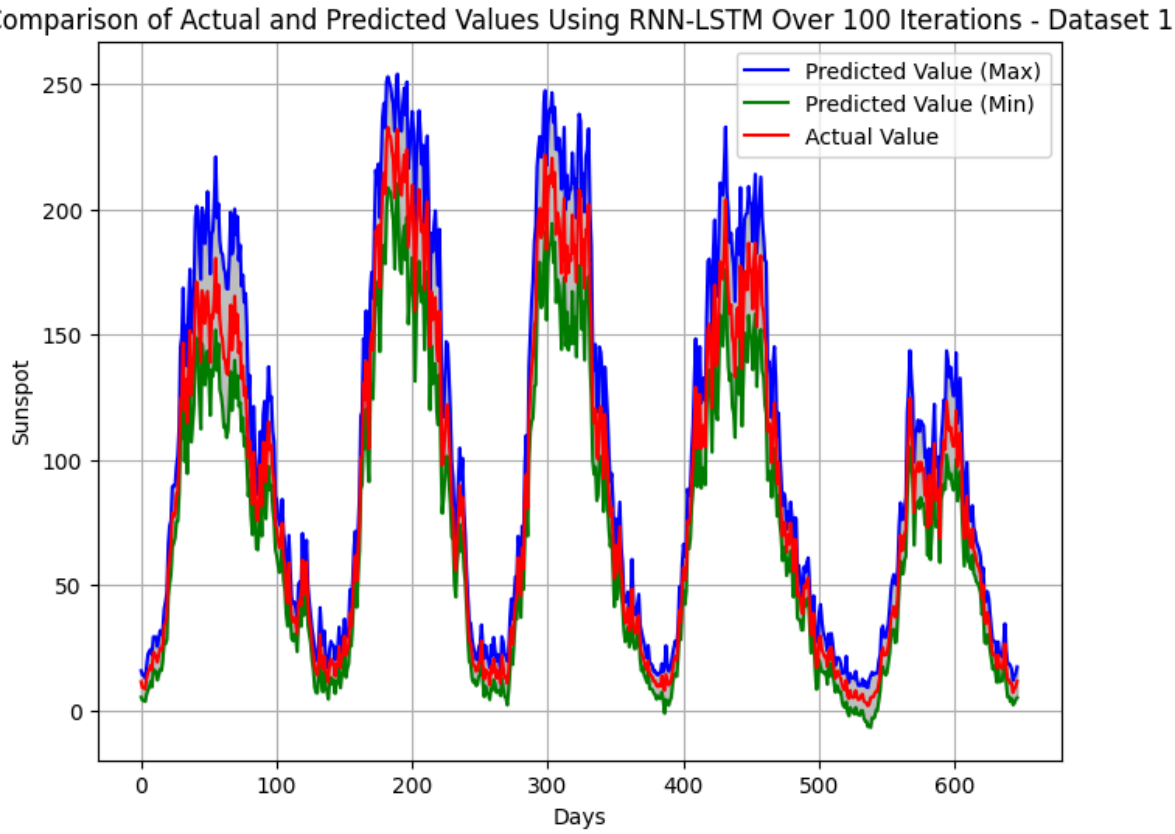
In [ ]:
```
!gdown 1Jcj7_vtMQ2foct0YPmWC6nO9q4T_x3bl
!gdown 1d-M7kCjRcggRBII-8o9WUhbWPaFxcPRg
```

RNN-LSTM

In [ ]:
```
temp = pd.read_excel('RNN-LSTM_LOSS.xlsx')
shade_plot(temp, line = 'RNN-LSTM_LOSS')
```

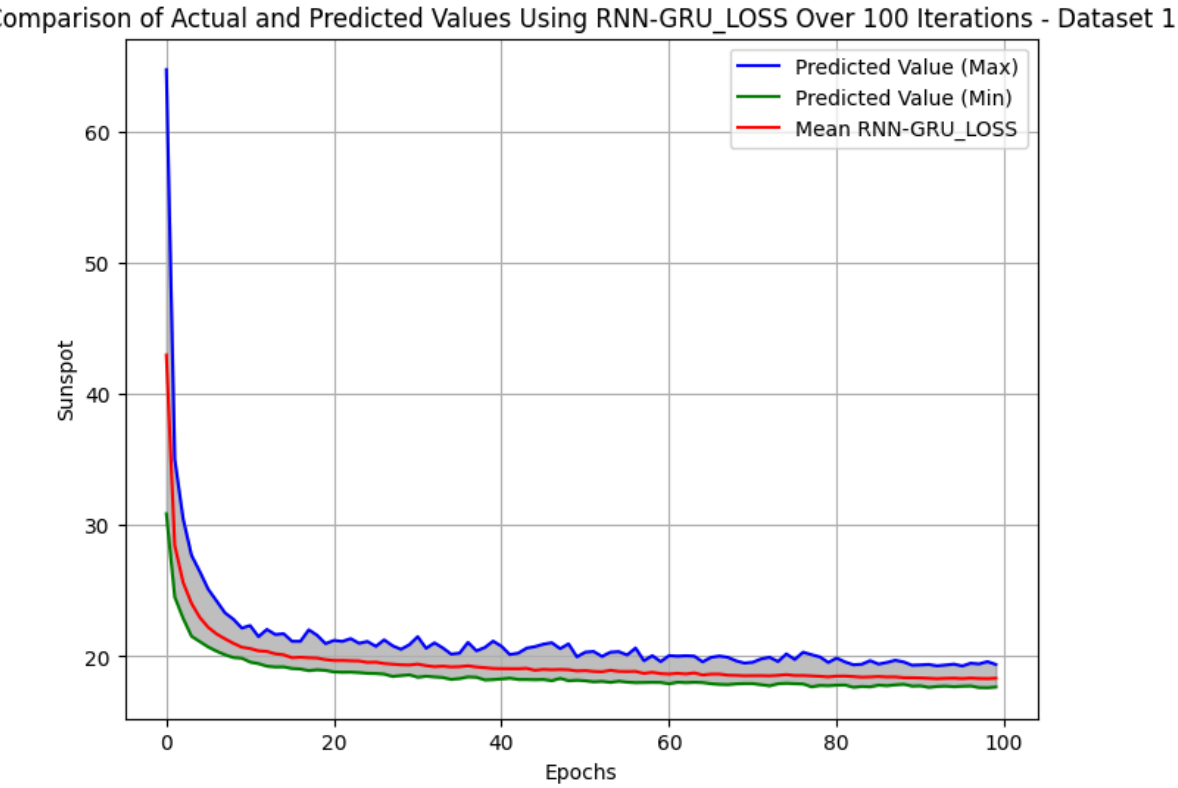


In [ ]:
```
temp = pd.read_excel('RNN-LSTM_PREDICT.xlsx')
shade_plot(temp, line='RNN-LSTM', predict='YES')
```

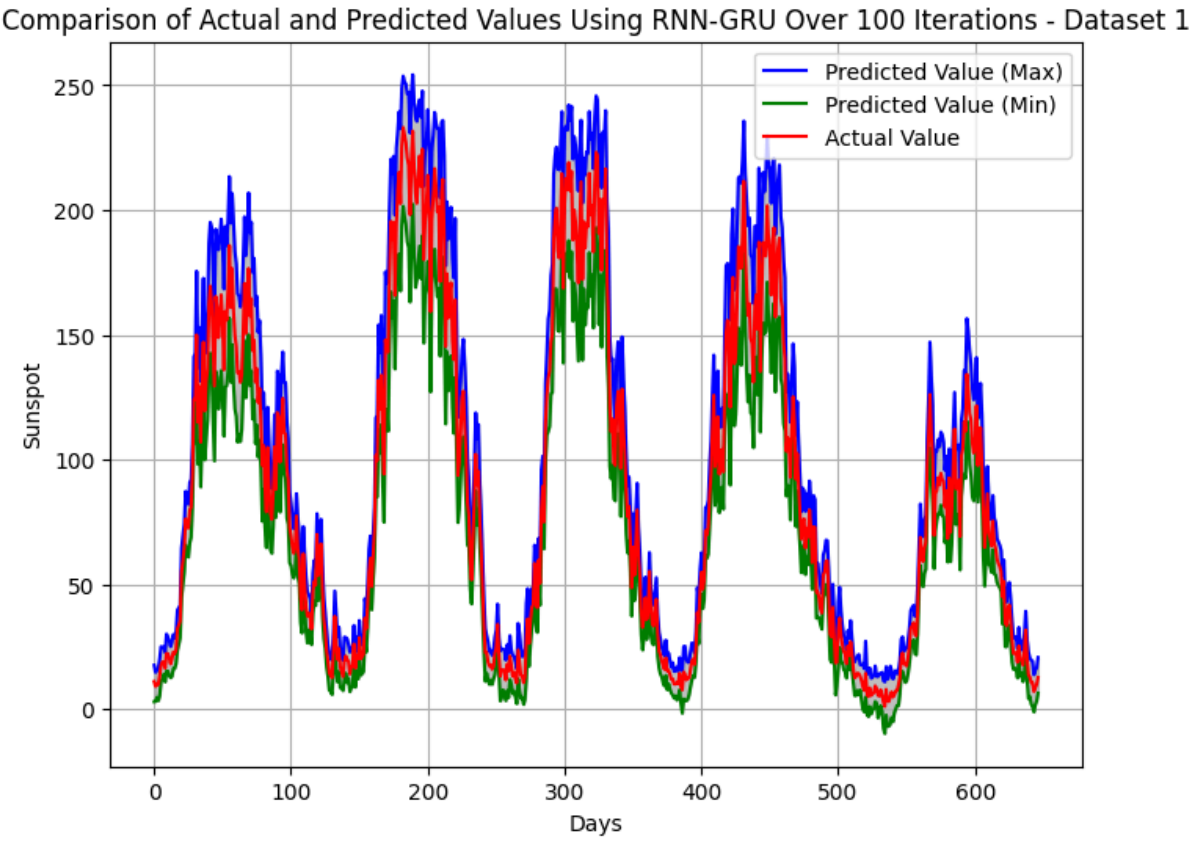


RNN-GRU

In [ ]:
```
temp = pd.read_excel('RNN-GRU_LOSS.xlsx')
shade_plot(temp, line = 'RNN-GRU_LOSS')
```

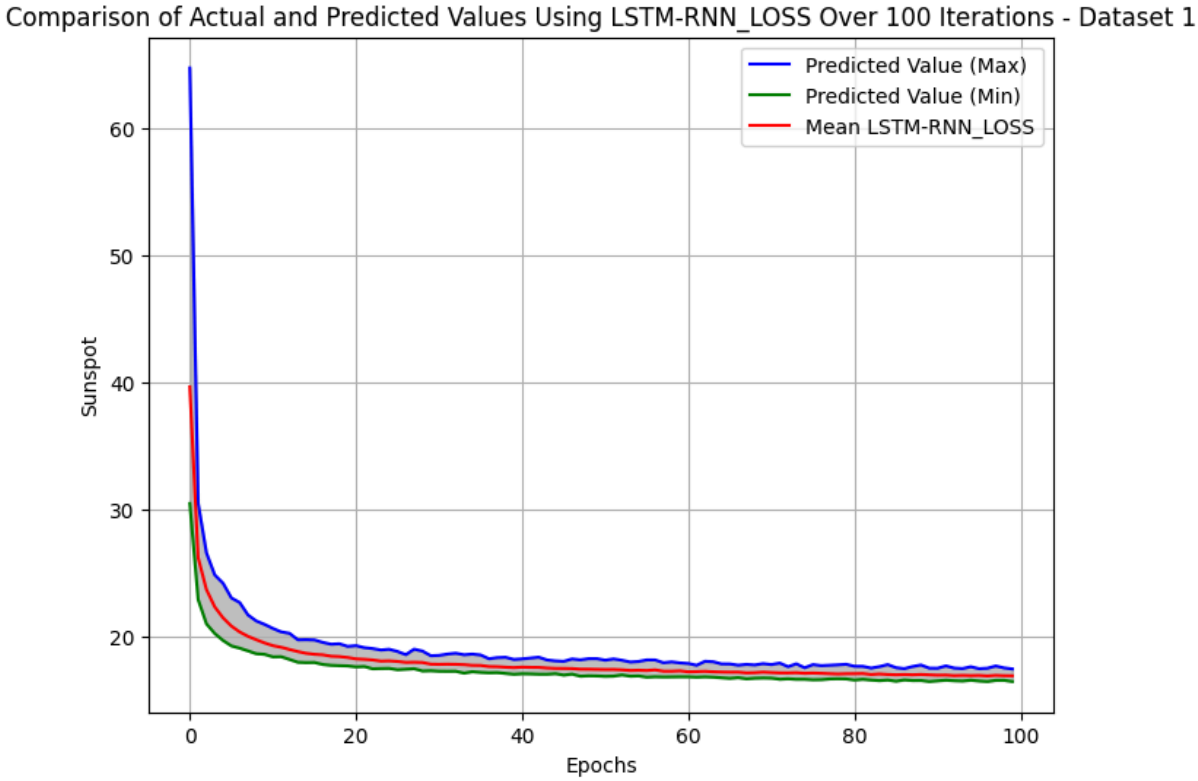


In [ ]:
```
temp = pd.read_excel('RNN-GRU_PREDICT.xlsx')
shade_plot(temp, line='RNN-GRU', predict='YES')
```

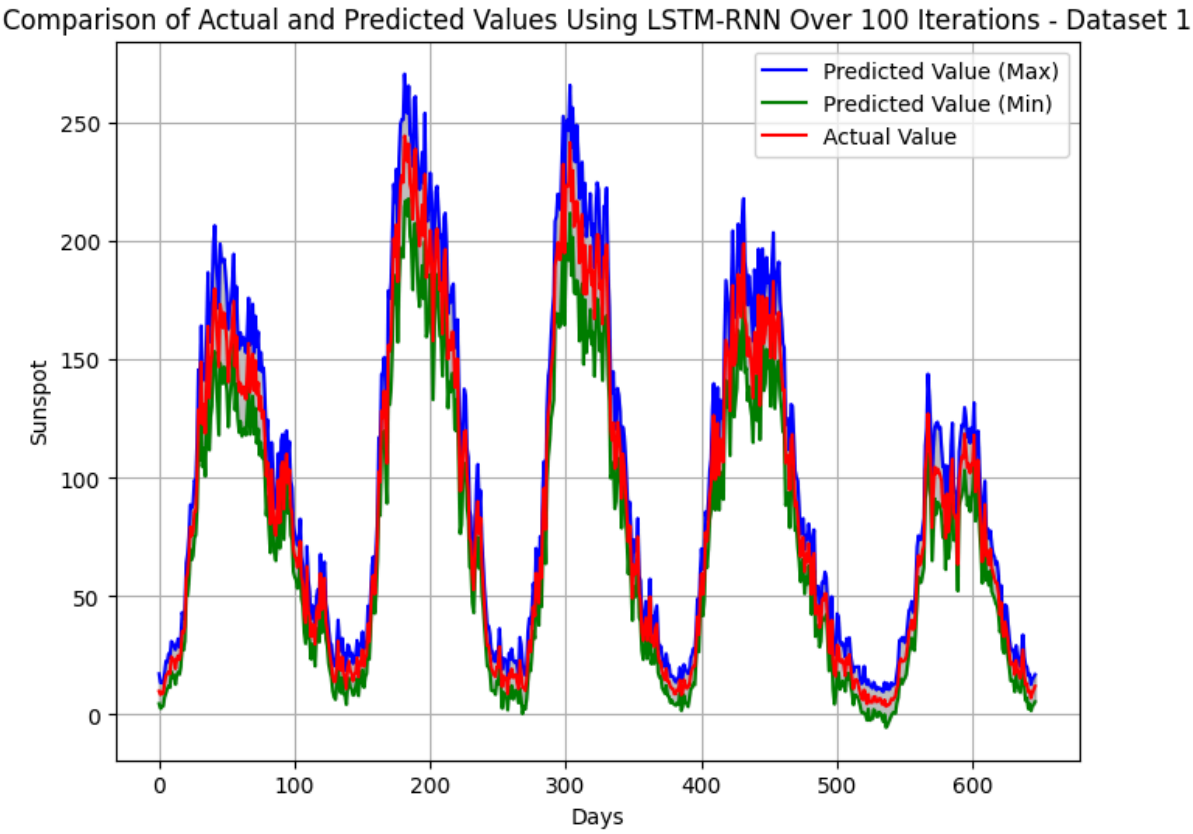


LSTM-RNN

In [ ]:
```
temp = pd.read_excel('LSTM-RNN_LOSS.xlsx')
shade_plot(temp, line = 'LSTM-RNN_LOSS')
```

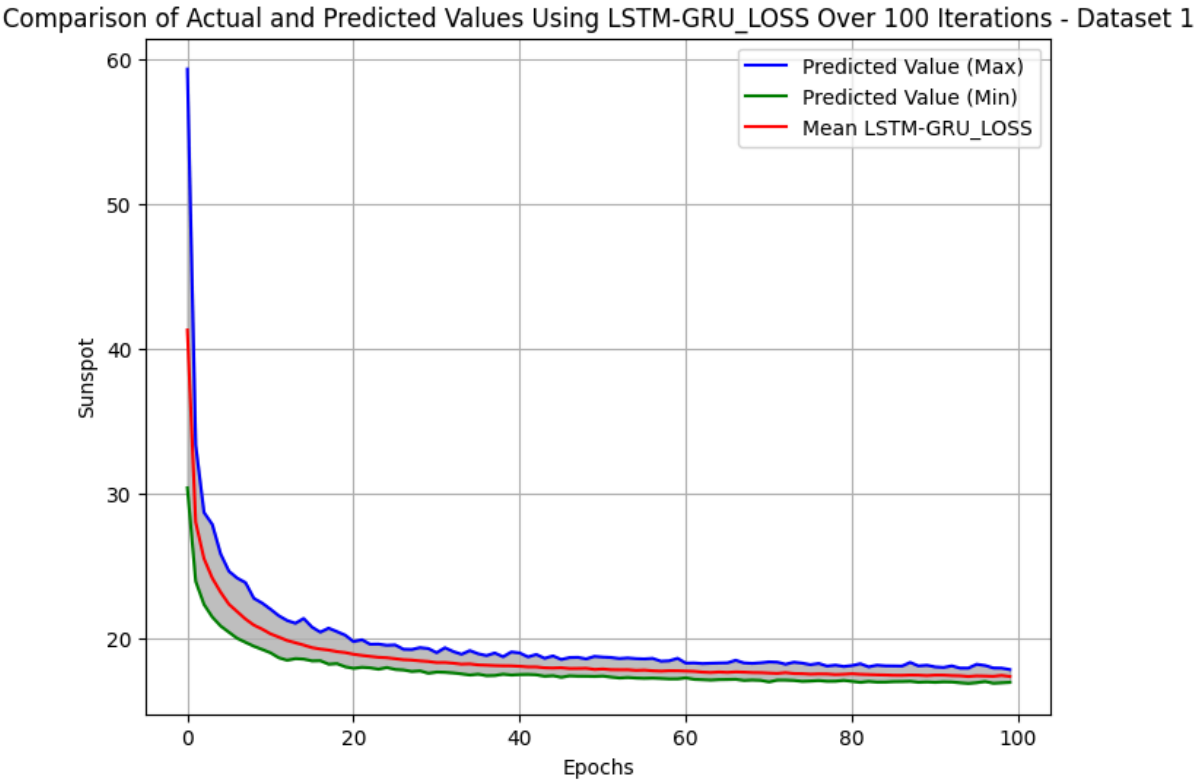


In [ ]:
```
temp = pd.read_excel('LSTM-RNN_PREDICT.xlsx')
shade_plot(temp, line='LSTM-RNN', predict='YES')
```

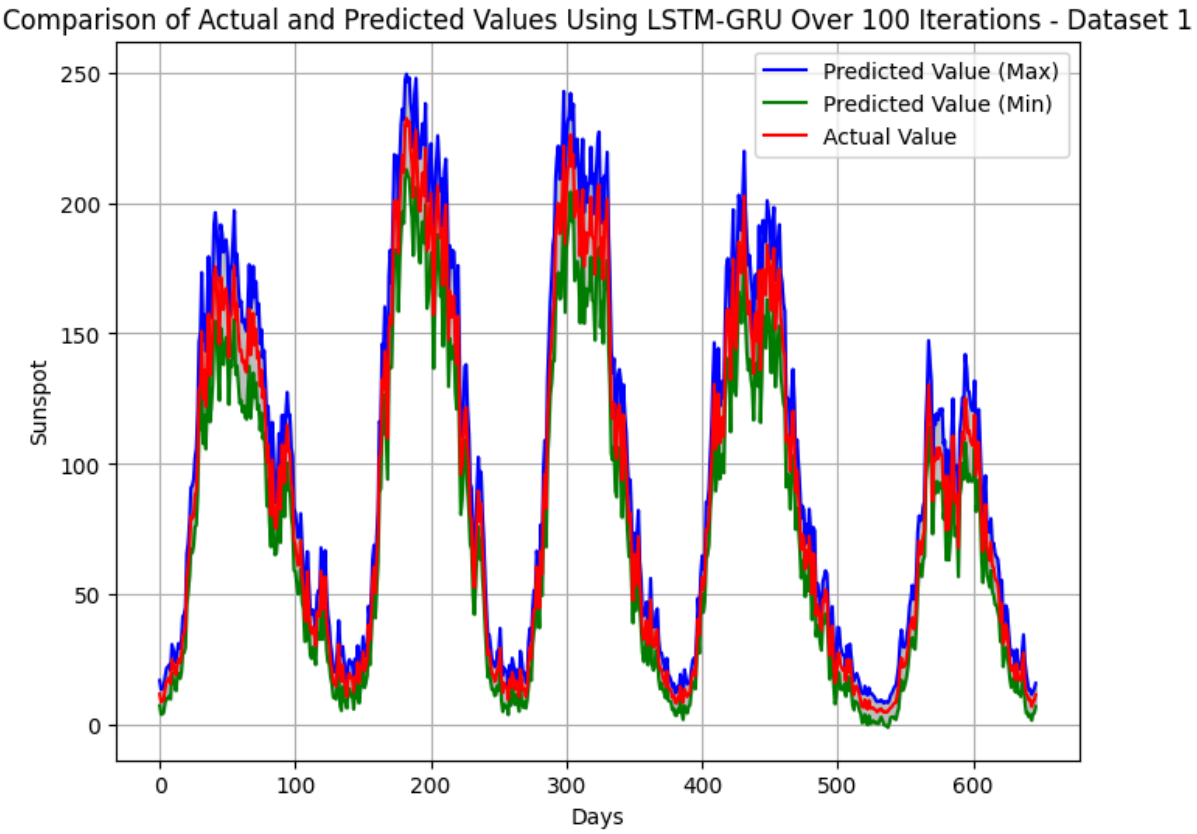


LSTM-GRU

In [ ]:
```
temp = pd.read_excel('LSTM-GRU_LOSS.xlsx')
shade_plot(temp, line = 'LSTM-GRU_LOSS')
```

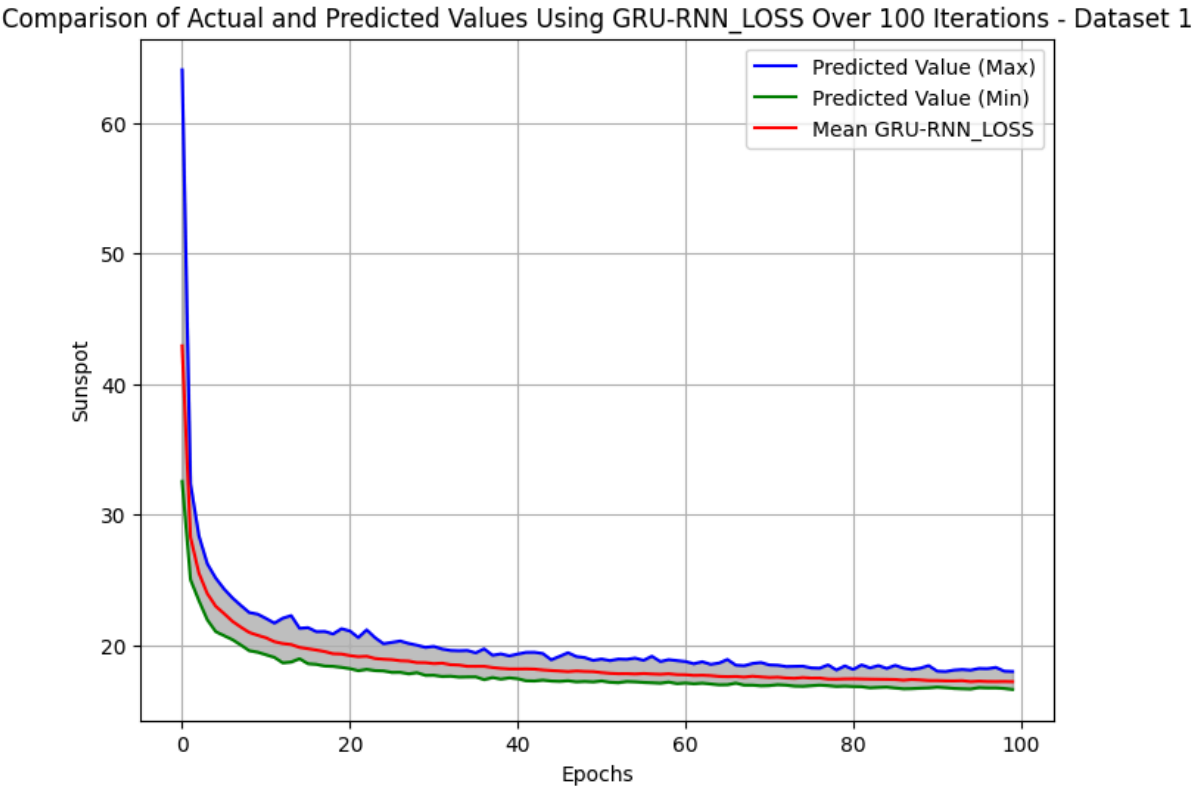


In [ ]:
```
temp = pd.read_excel('LSTM-GRU_PREDICT.xlsx')
shade_plot(temp, line='LSTM-GRU', predict='YES')
```

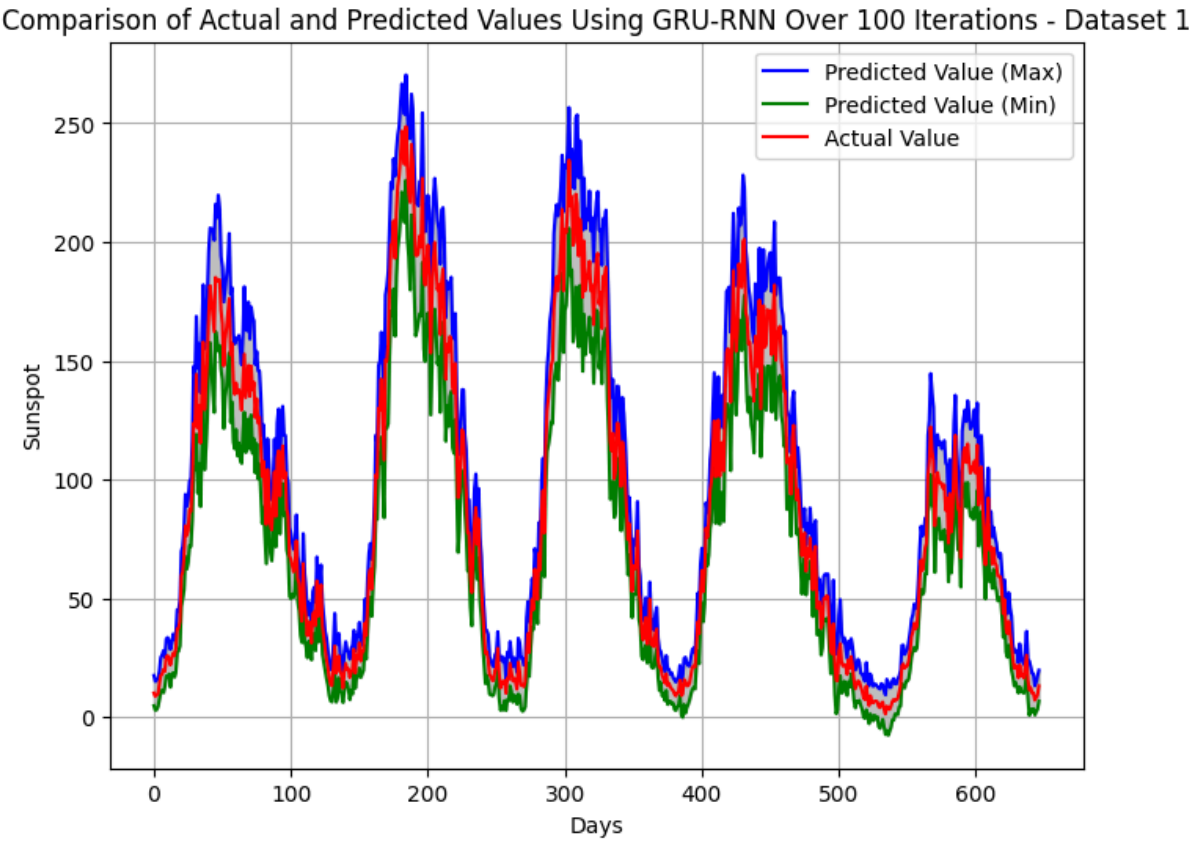


GRU-RNN

In [ ]:
```
temp = pd.read_excel('GRU-RNN_LOSS.xlsx')
shade_plot(temp, line = 'GRU-RNN_LOSS')
```

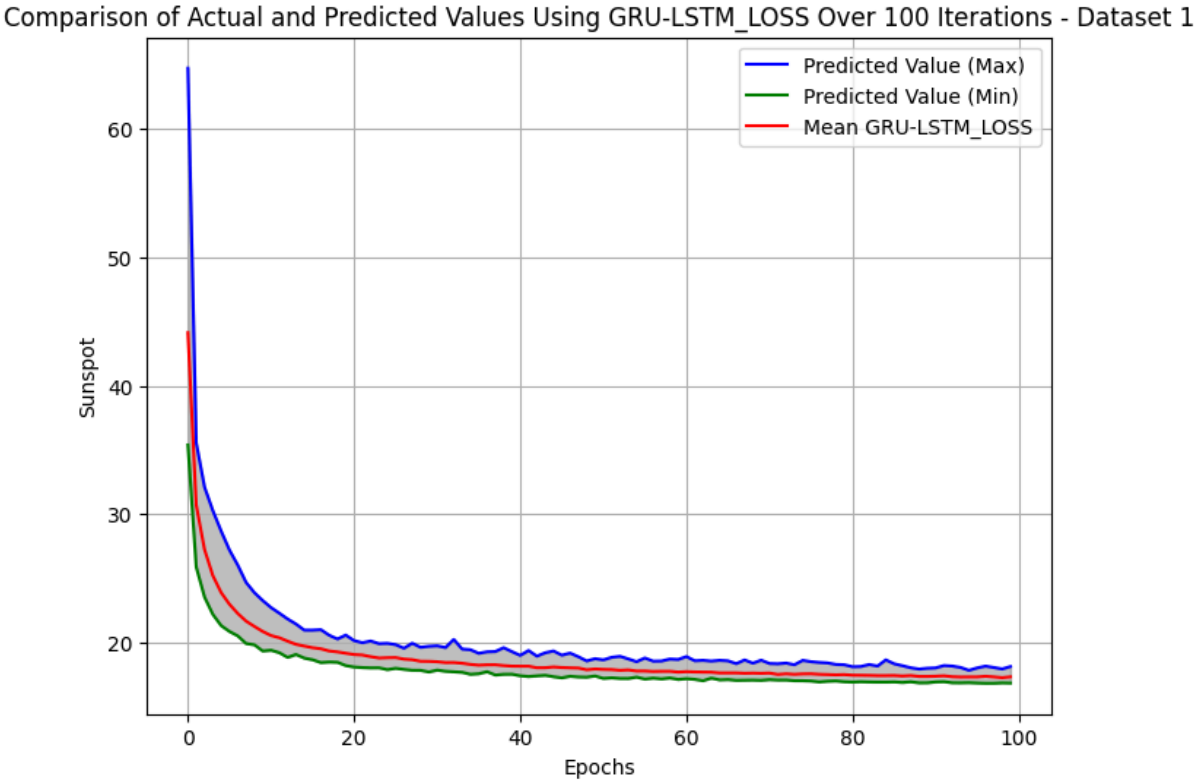


In [ ]:
```
temp = pd.read_excel('GRU-RNN_PREDICT.xlsx')
shade_plot(temp, line='GRU-RNN', predict='YES')
```

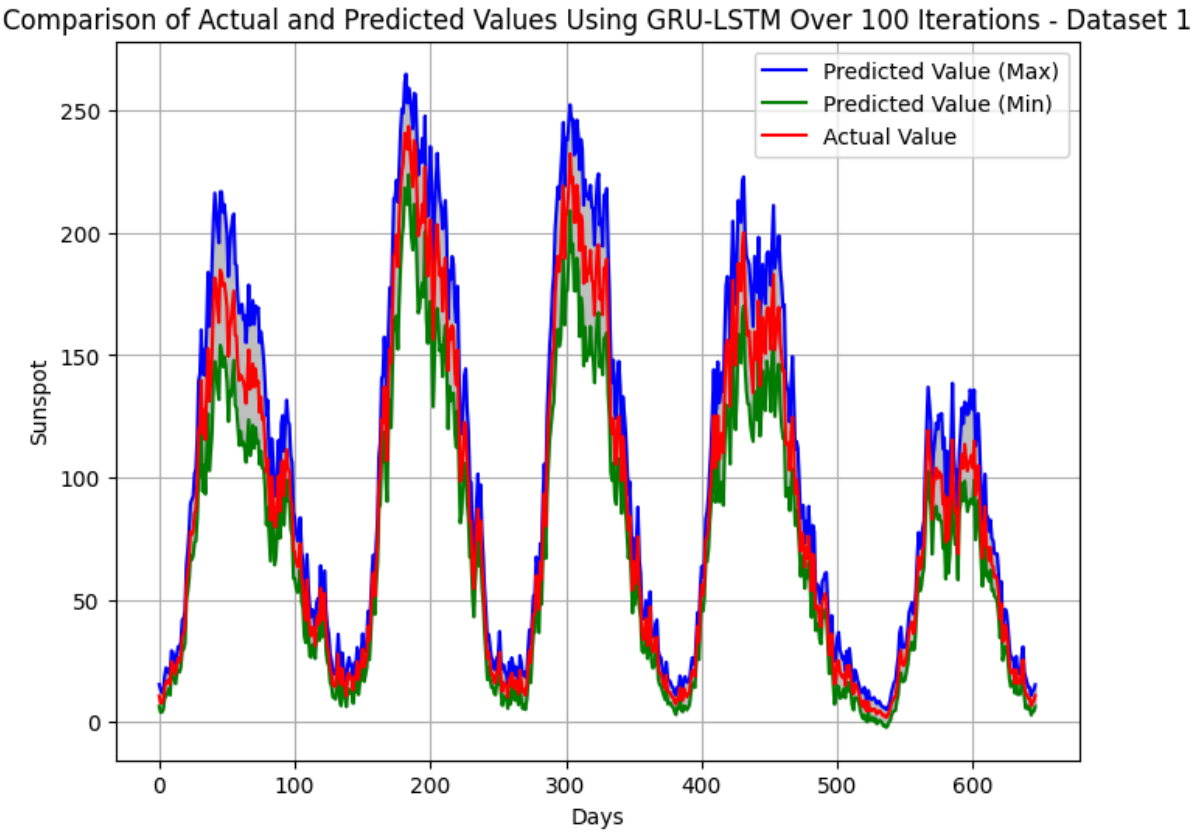


GRU-LSTM

In [ ]:
```
temp = pd.read_excel('GRU-LSTM_LOSS.xlsx')
shade_plot(temp, line = 'GRU-LSTM_LOSS')
```



In [ ]:
```
temp = pd.read_excel('GRU-LSTM_PREDICT.xlsx')
shade_plot(temp, line='GRU-LSTM', predict='YES')
```



In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import files
```

DONWLOAD DATA

In [ ]:
```
!gdown 1OHO_beLZo7QnekysxZa-JNAIjwAPfJj_
!gdown 1KM5U2pnghQeuIBGwa1SbmzvrMT18b91M
!gdown 1Lg6Gz_kXvypCKqjCxZOz1WalPGzapraV
!gdown 1Lg6Gz_kXvypCKqjCxZOz1WalPGzapraV
!gdown 1pBcD816adMQW1TxbokeRjJin8xGQGYer
!gdown 1rCcIouPG_QtA7OidY3HGQmRPooWpFJsp
!gdown 1m_5WyP0jU98vUgh124zyxCD67TEk_6zR
!gdown 1fsHt6tVw2ebOO_k0Xe4nkChJYKt8y1Ur
!gdown 1u0ywQUQAa69DcBPq_u9tcr7jePd98xiZ
!gdown 1H4KHj3giWcM8lPhU_AIQ32QyrOjY1JSh
!gdown 1RvkRM9vxddIDred7d56wJ2-aGe31u6uD
```

In [ ]:
```
!gdown 1y-VzDoWFEGmUOFWVpagg5CB0qkSzMiJB
!gdown 1Tls80_AHsKCkG4rR34y1hslf5rfOWif4
!gdown 1XOyVcuzMBwhitCjazcyemz33ZjoAOsvi
!gdown 11uxmJoYWW-9srCdDs-s9mSHk1i6AWBp0
!gdown 1mGXeE6uOVT3Pyvte7Lgqhn19hlYpybQH
!gdown 1bknN4MJf-puQAiEsvn1Hccdwmpqg1QkR
!gdown 1BakteERQlZYVln7buOHrMxcC2NL2aPCr
!gdown 1kbX7JzrmTkjlwX-NslbDxFSGZvT3sbpF
```

In [ ]:
```
!gdown 1HHSVdJzYWT93d_0qmeXE4VhOx5OYc0x0
!gdown 1En9dH6c-vmq72JmzAzcLlWHtI4X9xcT6
!gdown 172z5EOqrM8ACjr3g1JjSu-Oj5pwHdoxz
!gdown 1THv3DAS6EhuOsvKrqdKJ9PSZNoIweWfs
!gdown 1zgeDDSPKE4JF_rM3b29dE3ryy7M1W9gl
!gdown 1-f_PDTK891JK2vZOwTO5fuwl8GEC1Bh6
!gdown 1rRSz3vn4Y7ml5X9y57RbogUv5pipdVre
!gdown 1OYTOub9vv35Wiuxx3vgBAEIjOQzCksXK
!gdown 1lCYQvasS5ORVzdPbIm9Ipmj3qCXPcxjR
```

In [ ]:
```
!gdown 1GjhlV9cldg6dGRuPyyCctR5tTF-MHV1a
!gdown 1i_CkunKD18a04P0FF2EM-NxNyAKA5NZt
!gdown 1PAU06DAruF8R3xcG3DLytuGSL-gVmbB_
!gdown 198a4VKQm4fRFHE5EXWilcQdC5sBMv0qC
!gdown 13-Wt96xDZW3_jRi0bgctXclYNy0UOLUg
!gdown 1kScpmncOvzJnVa9x--jEP8bCG92bMPnI
!gdown 1G29FRXkxuHYEGn_PyshQ5oWxk59eH9Rk
!gdown 1G56MJL6SOOaYRjRrgml4vVPVdueLiE90
!gdown 1GqQs02UGzh1f_ZycvmVCPVdoc4JTOk4x
!gdown 1lCYQvasS5ORVzdPbIm9Ipmj3qCXPcxjR
```

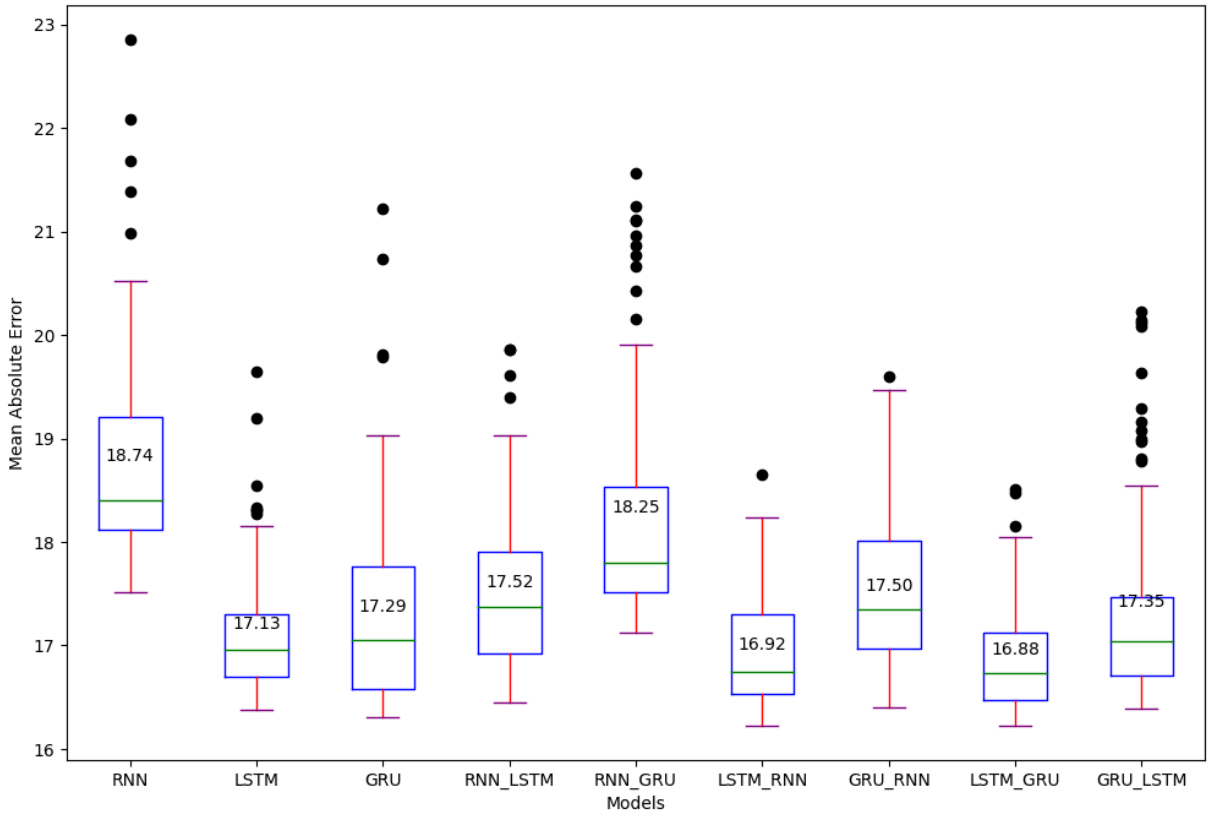
Metrik Evaluation

MAE

In [ ]:
```
df1 =  pd.read_excel('/content/RNN_FORECASTMae.xlsx')
df2 =  pd.read_excel('/content/LSTM_FORECASTMae.xlsx')
df3 =  pd.read_excel('/content/GRU_FORECASTMae.xlsx')
df4 =  pd.read_excel('/content/RNN-LSTM_FORECASTMae.xlsx')
df5 =  pd.read_excel('/content/RNN-GRU_FORECASTMae.xlsx')
df6 =  pd.read_excel('/content/LSTM-RNN_FORECASTMae.xlsx')
df7 =  pd.read_excel('/content/GRU-RNN_FORECASTMae.xlsx')
df8 =  pd.read_excel('/content/LSTM-GRU_FORECASTMae.xlsx')
df9 =  pd.read_excel('/content/GRU-LSTM_FORECASTMae.xlsx')
```

In [ ]:
```
df = pd.DataFrame()
df['RNN'] =  df1[0]
df['LSTM'] =  df2[0]
df['GRU'] =  df3[0]
df['RNN_LSTM'] =  df4[0]
df['RNN_GRU'] =  df5[0]
df['LSTM_RNN'] =  df6[0]
df['GRU_RNN'] =  df7[0]
df['LSTM_GRU'] =  df8[0]
df['GRU_LSTM'] =  df9[0]

# n = int(len(df) * (20 / 100))
n = int(len(df) * (2.5 / 100))
df = df.iloc[n:-n]
```

In [ ]:
```
df = pd.DataFrame()
df['RNN'] =  sorted(df1[0])
df['LSTM'] =  sorted(df2[0])
df['GRU'] =  sorted(df3[0])
df['RNN_LSTM'] =  sorted(df4[0])
df['RNN_GRU'] =  sorted(df5[0])
df['LSTM_RNN'] = sorted(df6[0])
df['GRU_RNN'] =  sorted(df7[0])
df['LSTM_GRU'] = sorted(df8[0])
df['GRU_LSTM'] = sorted(df9[0])


# n = int(len(df)*0.15)
n = int(len(df) * 0.025)
df = df[:].iloc[n:-n]
data = df[:]

box_color = 'blue'
whisker_color = 'red'
median_color = 'green'
flier_color = 'black'
cap_color = 'purple'

fig, ax = plt.subplots(figsize=(12, 8))

plt.boxplot(data.values, labels=data.columns,
            boxprops=dict(color=box_color),
            whiskerprops=dict(color=whisker_color),
            capprops=dict(color=cap_color),
            flierprops=dict(markerfacecolor=flier_color, marker='o', markersize=6),
            medianprops=dict(color=median_color))

for i, column in enumerate(data.columns):
    mean = np.mean(data[column])
    plt.text(i + 1, mean, f'{mean:.2f}', ha='center', va='bottom', color='black')

# plt.title('Predict with Boxplot')
plt.xlabel('Models')
plt.ylabel('Mean Absolute Error')
plt.savefig("Comparison_MAE_Dataset1.pdf", dpi=300)
files.download("Comparison_MAE_Dataset1.pdf")

plt.show()
```

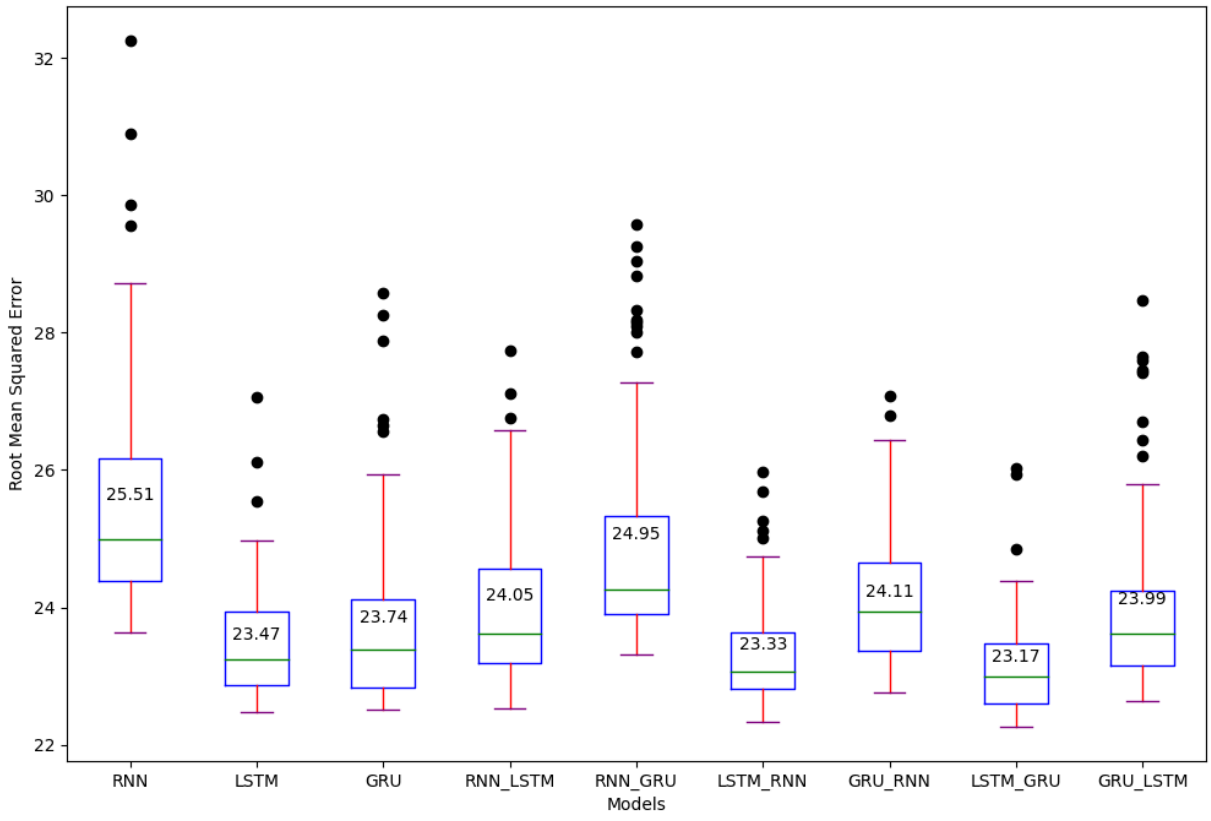


RMSE

In [ ]:
```
df1 =  pd.read_excel('/content/RNN_FORECASTRMse.xlsx')
df2 =  pd.read_excel('/content/LSTM_FORECASTRMse.xlsx')
df3 =  pd.read_excel('/content/GRU_FORECASTRMse.xlsx')
df4 =  pd.read_excel('/content/RNN-LSTM_FORECASTRMse.xlsx')
df5 =  pd.read_excel('/content/RNN-GRU_FORECASTRMse.xlsx')
df6 =  pd.read_excel('/content/LSTM-RNN_FORECASTRMse.xlsx')
df7 =  pd.read_excel('/content/GRU-RNN_FORECASTRMse.xlsx')
df8 =  pd.read_excel('/content/LSTM-GRU_FORECASTRMse.xlsx')
df9 =  pd.read_excel('/content/GRU-LSTM_FORECASTRMse.xlsx')
df = pd.DataFrame()
df['RNN'] =  df1[0]
df['LSTM'] =  df2[0]
df['GRU'] =  df3[0]
df['RNN_LSTM'] =  df4[0]
df['RNN_GRU'] =  df5[0]
df['LSTM_RNN'] =  df6[0]
df['GRU_RNN'] =  df7[0]
df['LSTM_GRU'] =  df8[0]
df['GRU_LSTM'] =  df9[0]

# n = int(len(df) * (20 / 100))
n = int(len(df) * (2.5 / 100))
df = df.iloc[n:-n]
```

In [ ]:
```
df = pd.DataFrame()
df['RNN'] =  sorted(df1[0])
df['LSTM'] =  sorted(df2[0])
df['GRU'] =  sorted(df3[0])
df['RNN_LSTM'] =  sorted(df4[0])
df['RNN_GRU'] =  sorted(df5[0])
df['LSTM_RNN'] = sorted(df6[0])
df['GRU_RNN'] =  sorted(df7[0])
df['LSTM_GRU'] = sorted(df8[0])
df['GRU_LSTM'] = sorted(df9[0])


# n = int(len(df)*0.15)
n = int(len(df) * 0.025)
```

```
df = df[:].iloc[n:-n]
data = df[:]

box_color = 'blue'
whisker_color = 'red'
median_color = 'green'
flier_color = 'black'
cap_color = 'purple'

fig, ax = plt.subplots(figsize=(12, 8))

plt.boxplot(data.values, labels=data.columns,
            boxprops=dict(color=box_color),
            whiskerprops=dict(color=whisker_color),
            capprops=dict(color=cap_color),
            flierprops=dict(markerfacecolor=flier_color, marker='o', markersize=6),
            medianprops=dict(color=median_color))

for i, column in enumerate(data.columns):
    mean = np.mean(data[column])
    plt.text(i + 1, mean, f'{mean:.2f}', ha='center', va='bottom', color='black')

# plt.title('Predict with Boxplot')
plt.xlabel('Models')
plt.ylabel('Root Mean Squared Error')
plt.savefig("Comparison_RMSE_Dataset1.pdf", dpi=300)
files.download("Comparison_RMSE_Dataset1.pdf")

plt.show()
```



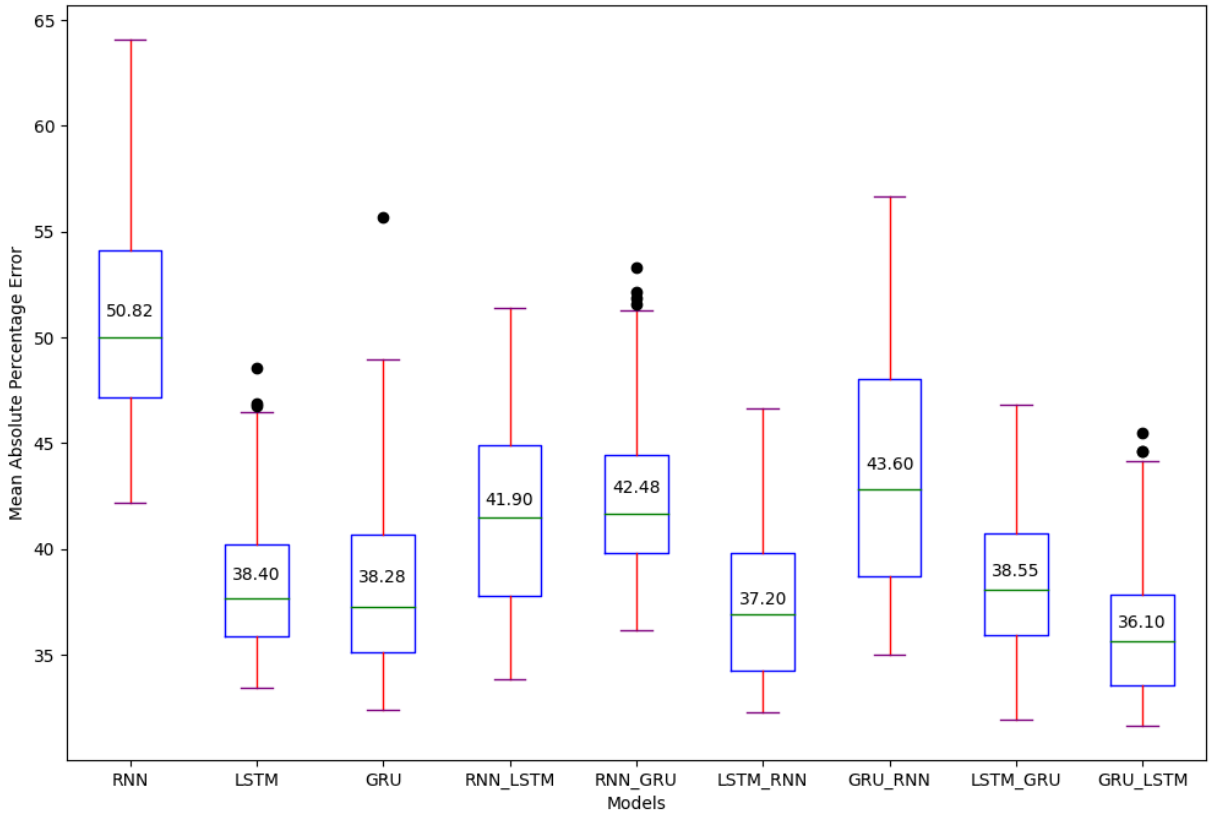
MAPE

In [ ]:
```
df1 =  pd.read_excel('/content/RNN_FORECASTMpe.xlsx')
df2 =  pd.read_excel('/content/LSTM_FORECASTMpe.xlsx')
df3 =  pd.read_excel('/content/GRU_FORECASTMpe.xlsx')
df4 =  pd.read_excel('/content/RNN-LSTM_FORECASTMpe.xlsx')
df5 =  pd.read_excel('/content/RNN-GRU_FORECASTMpe.xlsx')
df6 =  pd.read_excel('/content/LSTM-RNN_FORECASTMpe.xlsx')
df7 =  pd.read_excel('/content/GRU-RNN_FORECASTMpe.xlsx')
```

```
df8 =  pd.read_excel('/content/LSTM-GRU_FORECASTMpe.xlsx')
df9 =  pd.read_excel('/content/GRU-LSTM_FORECASTMpe.xlsx')
df = pd.DataFrame()
df['RNN'] =  df1[0]
df['LSTM'] =  df2[0]
df['GRU'] =  df3[0]
df['RNN_LSTM'] =  df4[0]
df['RNN_GRU'] =  df5[0]
df['LSTM_RNN'] =  df6[0]
df['GRU_RNN'] =  df7[0]
df['LSTM_GRU'] =  df8[0]
df['GRU_LSTM'] =  df9[0]

# n = int(len(df) * (20 / 100))
n = int(len(df) * (2.5 / 100))
df = df.iloc[n:-n]
```

In [ ]:

```
df = pd.DataFrame()
df['RNN'] =  sorted(df1[0])
df['LSTM'] =  sorted(df2[0])
df['GRU'] =  sorted(df3[0])
df['RNN_LSTM'] =  sorted(df4[0])
df['RNN_GRU'] =  sorted(df5[0])
df['LSTM_RNN'] = sorted(df6[0])
df['GRU_RNN'] =  sorted(df7[0])
df['LSTM_GRU'] = sorted(df8[0])
df['GRU_LSTM'] = sorted(df9[0])


# n = int(len(df)*0.15)
n = int(len(df) * 0.025)
df = df[:].iloc[n:-n]
data = df[:]

box_color = 'blue'
whisker_color = 'red'
median_color = 'green'
flier_color = 'black'
cap_color = 'purple'

fig, ax = plt.subplots(figsize=(12, 8))

plt.boxplot(data.values, labels=data.columns,
            boxprops=dict(color=box_color),
            whiskerprops=dict(color=whisker_color),
            capprops=dict(color=cap_color),
            flierprops=dict(markerfacecolor=flier_color, marker='o', markersize=6),
            medianprops=dict(color=median_color))

for i, column in enumerate(data.columns):
    mean = np.mean(data[column])
    plt.text(i + 1, mean, f'{mean:.2f}', ha='center', va='bottom', color='black')

# plt.title('Predict with Boxplot')
plt.xlabel('Models')
plt.ylabel('Mean Absolute Percentage Error')
plt.savefig("Comparison_MAPE_Dataset1.pdf", dpi=300)
files.download("Comparison_MAPE_Dataset1.pdf")

plt.show()
```

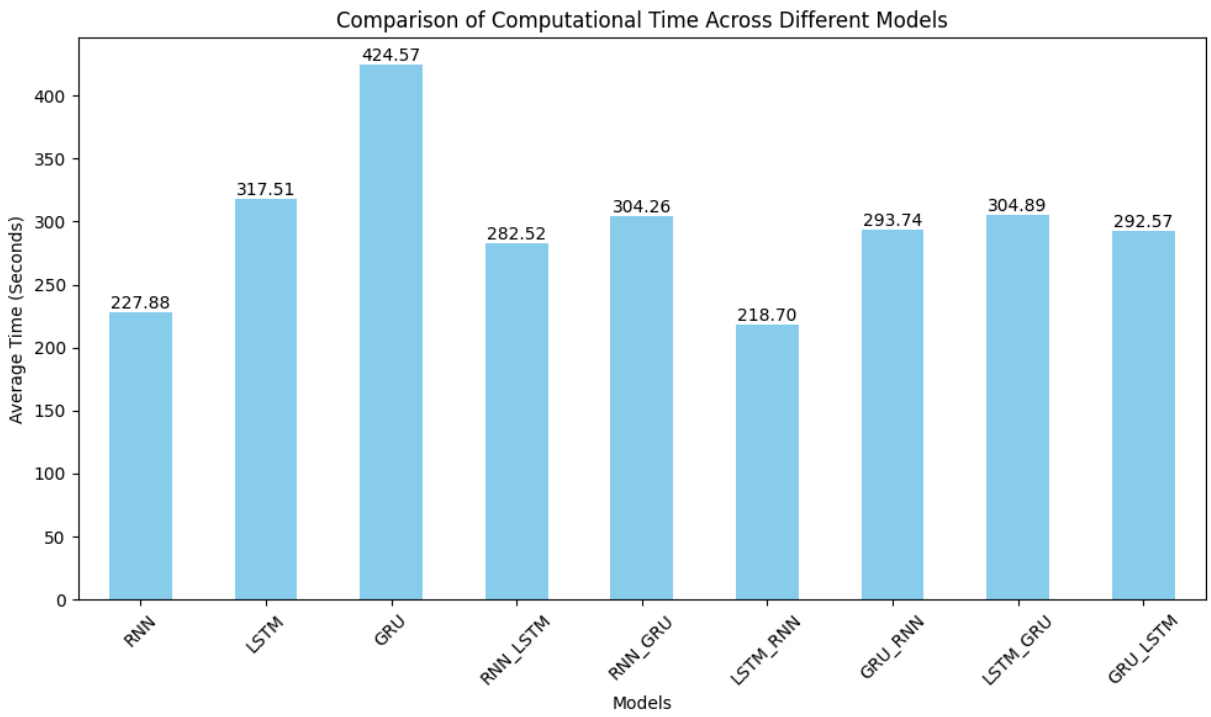


Time Comparison

In [ ]:
```
df1 =  pd.read_excel('/content/RNN_TIME.xlsx')
df2 =  pd.read_excel('/content/LSTM_TIME.xlsx')
df3 =  pd.read_excel('/content/GRU_TIME.xlsx')
df4 =  pd.read_excel('/content/RNN-LSTM_TIME.xlsx')
df5 =  pd.read_excel('/content/RNN-GRU_TIME.xlsx')
df6 =  pd.read_excel('/content/LSTM-RNN_TIME.xlsx')
df7 =  pd.read_excel('/content/GRU-RNN_TIME.xlsx')
df8 =  pd.read_excel('/content/LSTM-GRU_TIME.xlsx')
df9 =  pd.read_excel('/content/GRU-LSTM_TIME.xlsx')
df = pd.DataFrame()
df['RNN'] =  df1[0]
df['LSTM'] =  df2[0]
df['GRU'] =  df3[0]
df['RNN_LSTM'] =  df4[0]
df['RNN_GRU'] =  df5[0]
df['LSTM_RNN'] =  df6[0]
df['GRU_RNN'] =  df7[0]
df['LSTM_GRU'] =  df8[0]
df['GRU_LSTM'] =  df9[0]
```

In [ ]:
```
# Calculate the average time for each model
average_times = df.mean()

# Create a bar chart
plt.figure(figsize=(10, 6))
average_times.plot(kind='bar', color='skyblue')
plt.title('Comparison of Computational Time Across Different Models')
plt.xlabel('Models')
plt.ylabel('Average Time (Seconds)')
plt.xticks(rotation=45)
plt.tight_layout()

# Display the average times on top of the bars
for i, v in enumerate(average_times):
    plt.text(i, v, f"{v:.2f}", ha='center', va='bottom')

plt.savefig("Comparison_Time_Dataset1.pdf", dpi=300)
```

```
files.download("Comparison_Time_Dataset1.pdf")
plt.show()
```



1. Import Libraries

List of libraries that will be used.

In [6]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import csv
```

Using Utilites of Data Visualization

In [7]:
```
def plot_series(x, y, format="-", start=0, end=None,
                title=None, xlabel=None, ylabel=None, legend=None ):
    plt.figure(figsize=(10, 6))
    if type(y) is tuple:
      for y_curr in y:
        plt.plot(x[start:end], y_curr[start:end], format)
    else:
      plt.plot(x[start:end], y[start:end], format)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    if legend:
      plt.legend(legend)
    plt.title(title)
    plt.grid(True)
    plt.show()
```

2. Get DATA

Take Data from Upstream (Local, Gdrive, Github, etc).

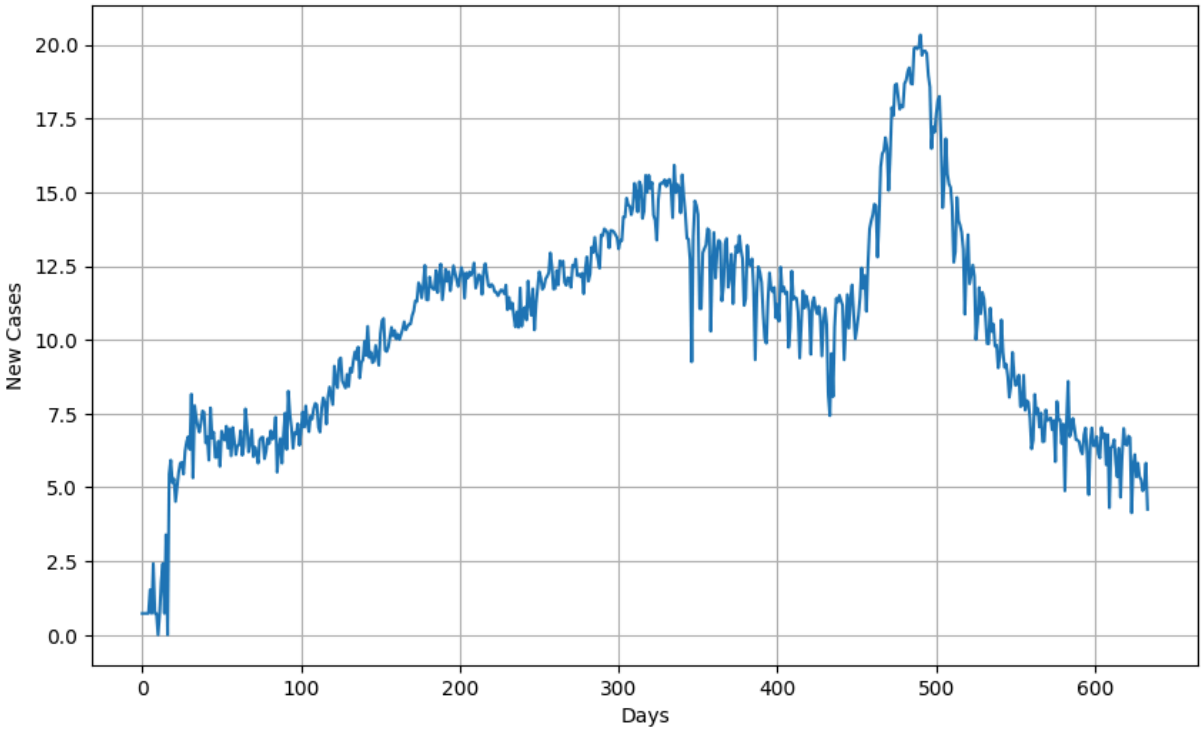
In [ ]:
```
!wget https://raw.githubusercontent.com/MhdIqbalPratama/ware-house/main/Data%20COVID-19%20Indon
```

Data Visualize

In [9]:
```
df = pd.read_csv(r"/content/Data COVID-19 Indonesia.csv", delimiter=',', parse_dates=['Date'])
df = df[df['Location'] == "DKI Jakarta"]
dx = df.groupby('Date').sum()
data =pd.DataFrame(dx).reset_index()
data.rename(columns={'New Cases' : 'New_Cases', 'New Deaths':'New_Deaths'}, inplace=True)
from scipy.stats import boxcox
data = data[data['New_Cases'] > 0]
data['New_Cases'], lambda_value = boxcox(data['New_Cases'])
```

In [10]:
```
def take(file):
  time_step = []
  data = []

  for i in range(len(file)):
    time_step.append(i)
    data.append(float(file.iloc[i][3]))
  time = np.array(time_step)
  series = np.array(data)
  return time, series
time, series = take(data)
plot_series(time, series, xlabel='Days', ylabel='New Cases')
```

```
<ipython-input-10-54f33a54f1e4>:7: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  data.append(float(file.iloc[i][3]))
```



3. Data Prerocessing

Analysis and Cleaning Data -> Extract, Transform, Load.

Split Data

In [11]:
```
split_time = int(len(time)*0.7)

time_train = time[:split_time]
x_train = series[:split_time]

time_valid = time[split_time:]
x_valid = series[split_time:]
```

Prepare Features and Labels

In [12]:
```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
    dataset = dataset.map(lambda window: (window[:-1], window[-1]))
    dataset = dataset.shuffle(shuffle_buffer)
    dataset = dataset.batch(batch_size).prefetch(1)
    return dataset

window_size = 7
batch_size = 32
shuffle_buffer_size = 1000

train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)
```

4. Model Architechture

Using library from Scikit-learn, Tensorflow(?).

RNN Model

In [13]:
```
model_RNN = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                        activation="relu",
                        padding='causal',
                        input_shape=[window_size, 1]),
  tf.keras.layers.SimpleRNN(64, return_sequences=True),
  tf.keras.layers.SimpleRNN(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_RNN.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: 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__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 7, 64)	256
simple_rnn (SimpleRNN)	(None, 7, 64)	8,256
simple_rnn_1 (SimpleRNN)	(None, 64)	8,256
dense (Dense)	(None, 1)	65
lambda (Lambda)	(None, 1)	0

Total params: 16,833 (65.75 KB)

Trainable params: 16,833 (65.75 KB)

Non-trainable params: 0 (0.00 B)

LSTM Model

In [14]:
```
model_LSTM = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                        activation="relu",
                        padding='causal',
                        input_shape=[window_size, 1]),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.LSTM(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_LSTM.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 7, 64)	256
lstm (LSTM)	(None, 7, 64)	33,024
lstm_1 (LSTM)	(None, 64)	33,024
dense_1 (Dense)	(None, 1)	65
lambda_1 (Lambda)	(None, 1)	0

Total params: 66,369 (259.25 KB)

Trainable params: 66,369 (259.25 KB)

GRU Model

In [15]:
```
model_GRU = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                        activation="relu",
                        padding='causal',
                        input_shape=[window_size, 1]),
  tf.keras.layers.GRU(64, return_sequences=True),
  tf.keras.layers.GRU(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_GRU.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv1d_2 (Conv1D)	(None, 7, 64)	256
gru (GRU)	(None, 7, 64)	24,960
gru_1 (GRU)	(None, 64)	24,960
dense_2 (Dense)	(None, 1)	65
lambda_2 (Lambda)	(None, 1)	0

Total params: 50,241 (196.25 KB)

Trainable params: 50,241 (196.25 KB)

Non-trainable params: 0 (0.00 B)

RNN-LSTM Model

In [16]:
```
model_RNN_LSTM = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                         activation="relu",
                         padding='causal',
                         input_shape=[window_size, 1]),
  tf.keras.layers.SimpleRNN(64, return_sequences=True),
  tf.keras.layers.LSTM(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_RNN_LSTM.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv1d_3 (Conv1D)	(None, 7, 64)	256
simple_rnn_2 (SimpleRNN)	(None, 7, 64)	8,256
lstm_2 (LSTM)	(None, 64)	33,024
dense_3 (Dense)	(None, 1)	65
lambda_3 (Lambda)	(None, 1)	0

Total params: 41,601 (162.50 KB)

Trainable params: 41,601 (162.50 KB)

RNN-GRU Model

In [17]:
```
model_RNN_GRU = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                         activation="relu",
                         padding='causal',
                         input_shape=[window_size, 1]),
  tf.keras.layers.SimpleRNN(64, return_sequences=True),
  tf.keras.layers.GRU(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_RNN_GRU.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv1d_4 (Conv1D)	(None, 7, 64)	256
simple_rnn_3 (SimpleRNN)	(None, 7, 64)	8,256
gru_2 (GRU)	(None, 64)	24,960
dense_4 (Dense)	(None, 1)	65
lambda_4 (Lambda)	(None, 1)	0

Total params: 33,537 (131.00 KB)

Trainable params: 33,537 (131.00 KB)

Non-trainable params: 0 (0.00 B)

LSTM-RNN

In [18]:
```
model_LSTM_RNN = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                         activation="relu",
                         padding='causal',
                         input_shape=[window_size, 1]),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.SimpleRNN(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
```

```
])
model_LSTM_RNN.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
conv1d_5 (Conv1D)	(None, 7, 64)	256
lstm_3 (LSTM)	(None, 7, 64)	33,024
simple_rnn_4 (SimpleRNN)	(None, 64)	8,256
dense_5 (Dense)	(None, 1)	65
lambda_5 (Lambda)	(None, 1)	0

Total params: 41,601 (162.50 KB)

Trainable params: 41,601 (162.50 KB)

Non-trainable params: 0 (0.00 B)

LSTM-GRU Model

In [19]:
```
model_LSTM_GRU = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                         activation="relu",
                         padding='causal',
                         input_shape=[window_size, 1]),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.GRU(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_LSTM_GRU.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
conv1d_6 (Conv1D)	(None, 7, 64)	256
lstm_4 (LSTM)	(None, 7, 64)	33,024
gru_3 (GRU)	(None, 64)	24,960
dense_6 (Dense)	(None, 1)	65
lambda_6 (Lambda)	(None, 1)	0

Total params: 58,305 (227.75 KB)

Trainable params: 58,305 (227.75 KB)

Non-trainable params: 0 (0.00 B)

GRU-RNN Model

In [20]:
```
model_GRU_RNN = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                         activation="relu",
                         padding='causal',
```

```
                      input_shape=[window_size, 1]),
  tf.keras.layers.GRU(64, return_sequences=True),
  tf.keras.layers.SimpleRNN(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_GRU_RNN.summary()
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
conv1d_7 (Conv1D)	(None, 7, 64)	256
gru_4 (GRU)	(None, 7, 64)	24,960
simple_rnn_5 (SimpleRNN)	(None, 64)	8,256
dense_7 (Dense)	(None, 1)	65
lambda_7 (Lambda)	(None, 1)	0

Total params: 33,537 (131.00 KB)

Trainable params: 33,537 (131.00 KB)

Non-trainable params: 0 (0.00 B)

GRU-LSTM Model

In [21]:
```
model_GRU_LSTM = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                        activation="relu",
                        padding='causal',
                        input_shape=[window_size, 1]),
  tf.keras.layers.GRU(64, return_sequences=True),
  tf.keras.layers.LSTM(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_GRU_LSTM  .summary()
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
conv1d_8 (Conv1D)	(None, 7, 64)	256
gru_5 (GRU)	(None, 7, 64)	24,960
lstm_5 (LSTM)	(None, 64)	33,024
dense_8 (Dense)	(None, 1)	65
lambda_8 (Lambda)	(None, 1)	0

Total params: 58,305 (227.75 KB)

Trainable params: 58,305 (227.75 KB)

Non-trainable params: 0 (0.00 B)

5. Model Training

In [22]:
```
tf.keras.backend.clear_session()
learning_rate = 8e-7
```

In [ ]:
```
optimizer_RNN = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_RNN.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_RNN,
              metrics=["mae", "mse", "mape"])
history_RNN = model_RNN.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_LSTM = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_LSTM.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_LSTM,
              metrics=["mae", "mse", "mape"])
history_LSTM = model_LSTM.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_GRU= tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_GRU.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_GRU,
              metrics=["mae", "mse", "mape"])
history_GRU = model_GRU.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_RNN_LSTM = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_RNN_LSTM.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_RNN_LSTM,
              metrics=["mae", "mse", "mape"])
history_RNN_LSTM = model_RNN_LSTM.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_RNN_GRU = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_RNN_GRU.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_RNN_GRU,
              metrics=["mae", "mse", "mape"])
history_RNN_GRU = model_RNN_GRU.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_LSTM_RNN = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_LSTM_RNN.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_LSTM_RNN,
              metrics=["mae", "mse", "mape"])
history_LSTM_RNN = model_LSTM_RNN.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_LSTM_GRU = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_LSTM_GRU.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_LSTM_GRU,
              metrics=["mae", "mse", "mape"])
history_LSTM_GRU = model_LSTM_GRU.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_GRU_RNN= tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_GRU_RNN.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_GRU_RNN,
              metrics=["mae", "mse", "mape"])
history_GRU_RNN = model_GRU_RNN.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_GRU_LSTM= tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_GRU_LSTM.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_GRU_LSTM,
              metrics=["mae", "mse", "mape"])
history_GRU_LSTM = model_GRU_LSTM.fit(train_set,epochs=100)
```

6. Model Evaluation

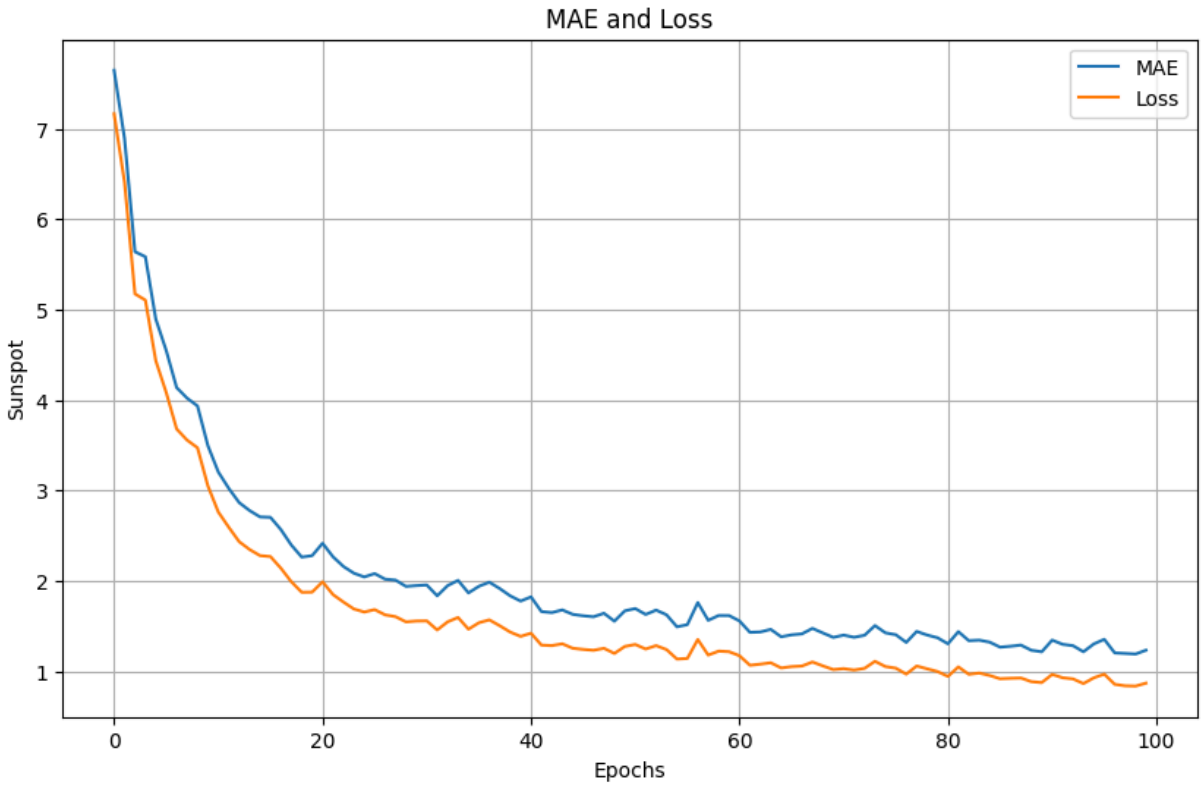
Loss evaluation using Scatter Plot

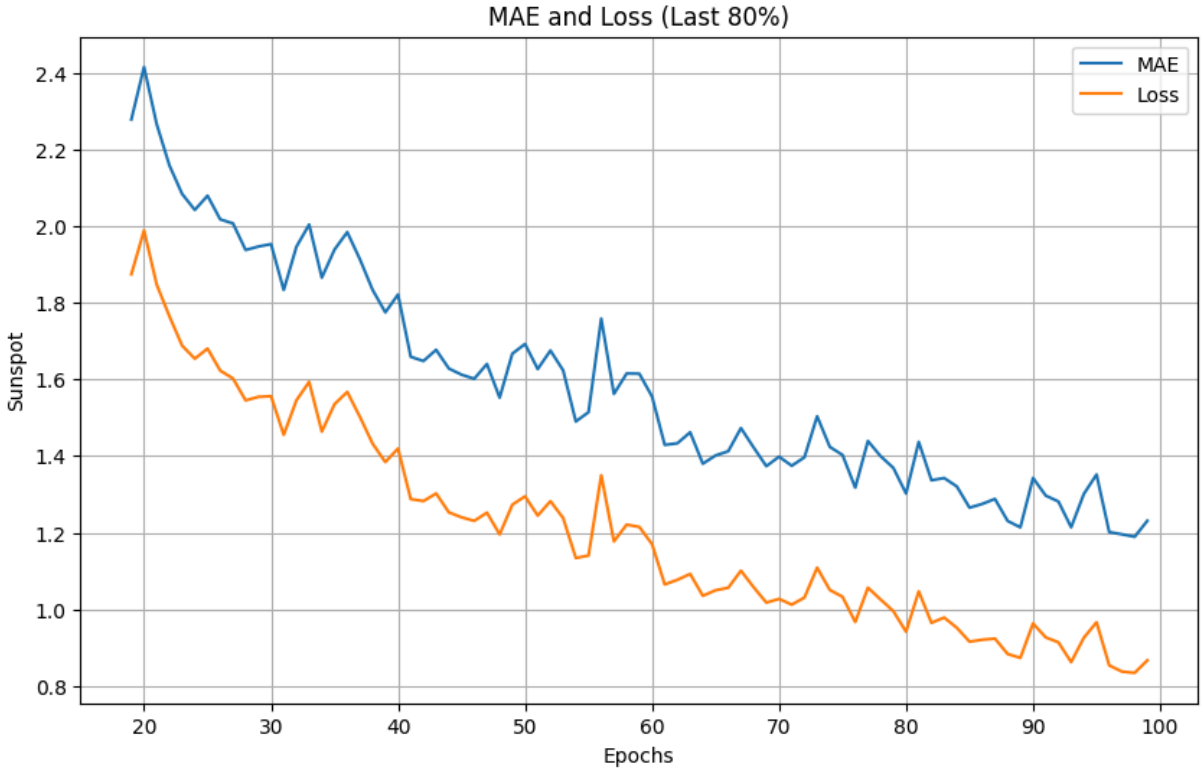
In [32]:
```
# Plot Function
def visualize_evaluation(history):
  mae=history.history['mae']
  loss=history.history['loss']
```

```
epochs=range(len(loss))

plot_series(
    x=epochs,
    y=(mae, loss),
    title='MAE and Loss',
    xlabel='Epochs',
    ylabel='Sunspot',
    legend=['MAE', 'Loss'])

zoom_split = int(epochs[-1] * 0.2)
epochs_zoom = epochs[zoom_split:]
mae_zoom = mae[zoom_split:]
loss_zoom = loss[zoom_split:]

plot_series(
    x=epochs_zoom,
    y=(mae_zoom, loss_zoom),
    title='MAE and Loss (Last 80%)',
    xlabel='Epochs',
    ylabel='Sunspot',
    legend=['MAE', 'Loss'])
```

RNN Evaluation

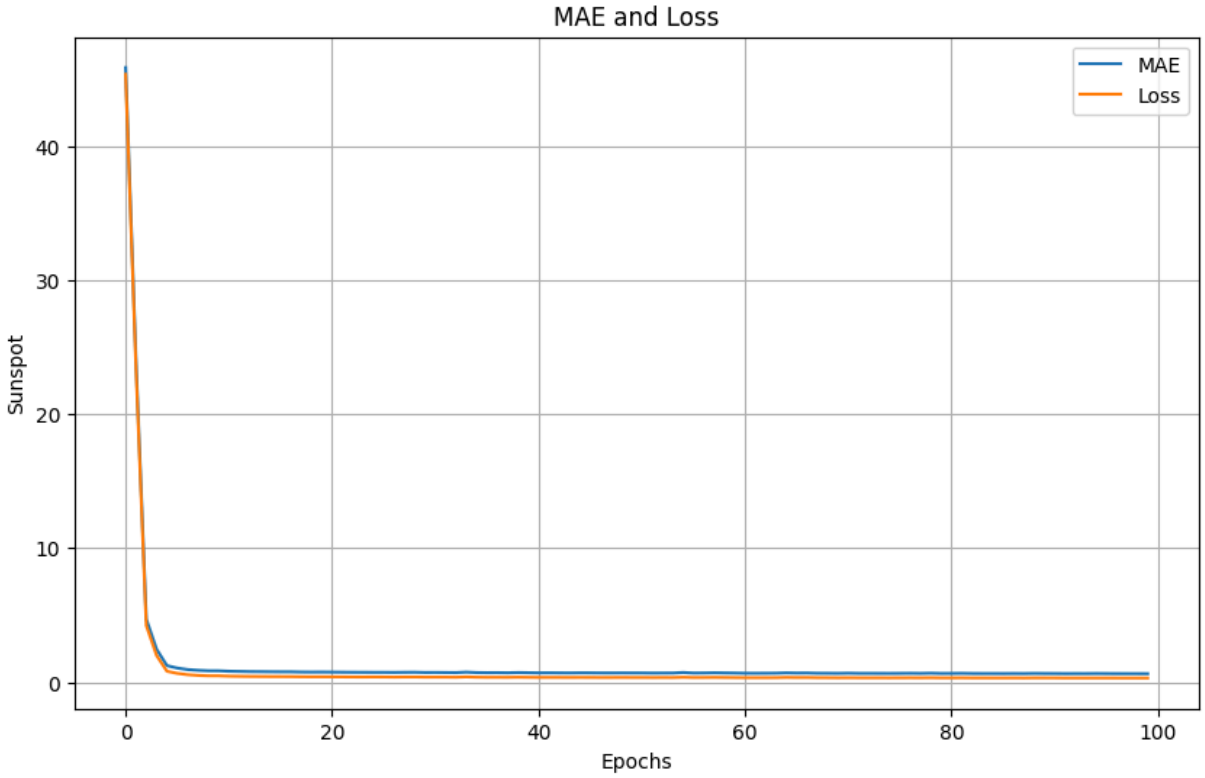
In [33]:
```
visualize_evaluation(history_RNN)
```

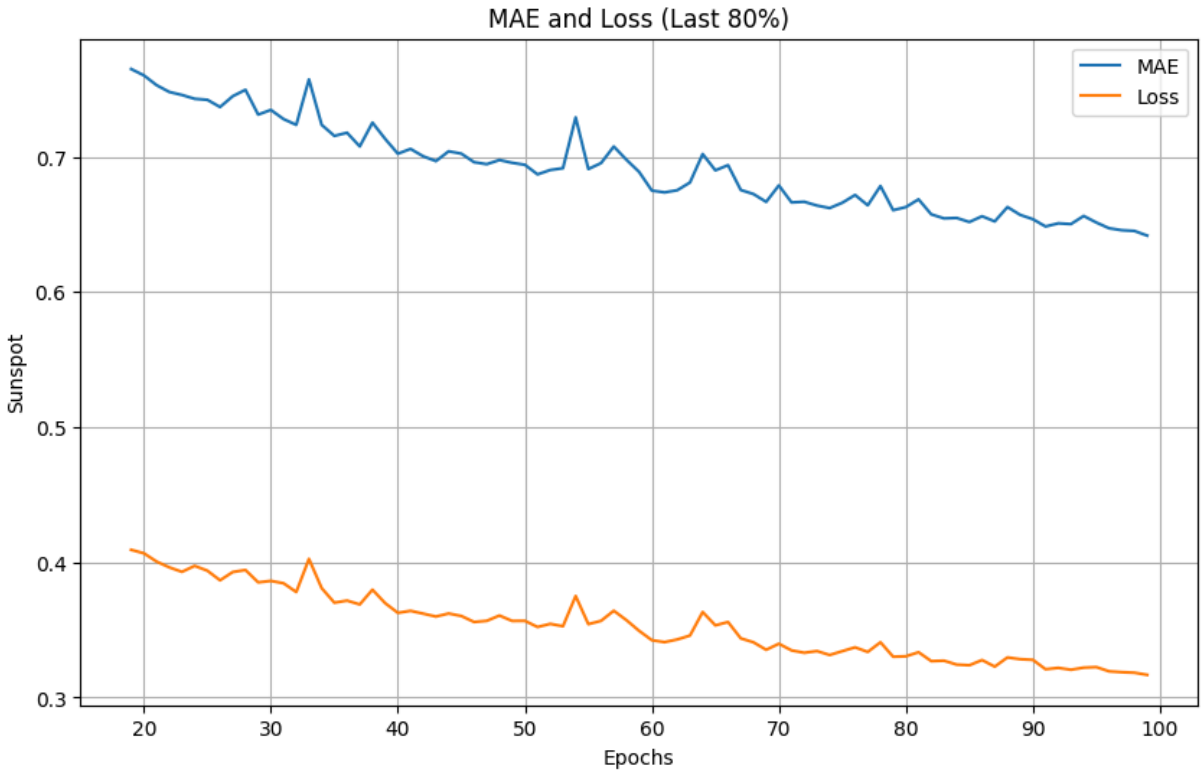




LSTM Evaluation

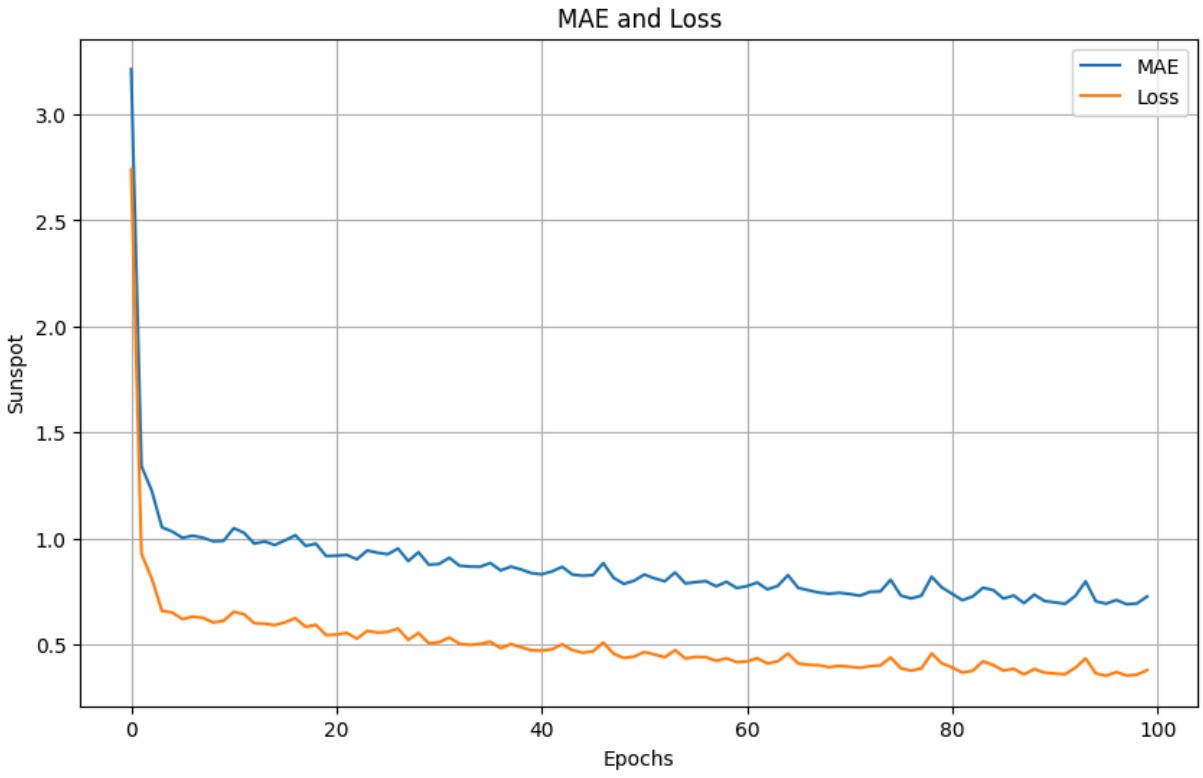
In [34]:
```
visualize_evaluation(history_LSTM)
```

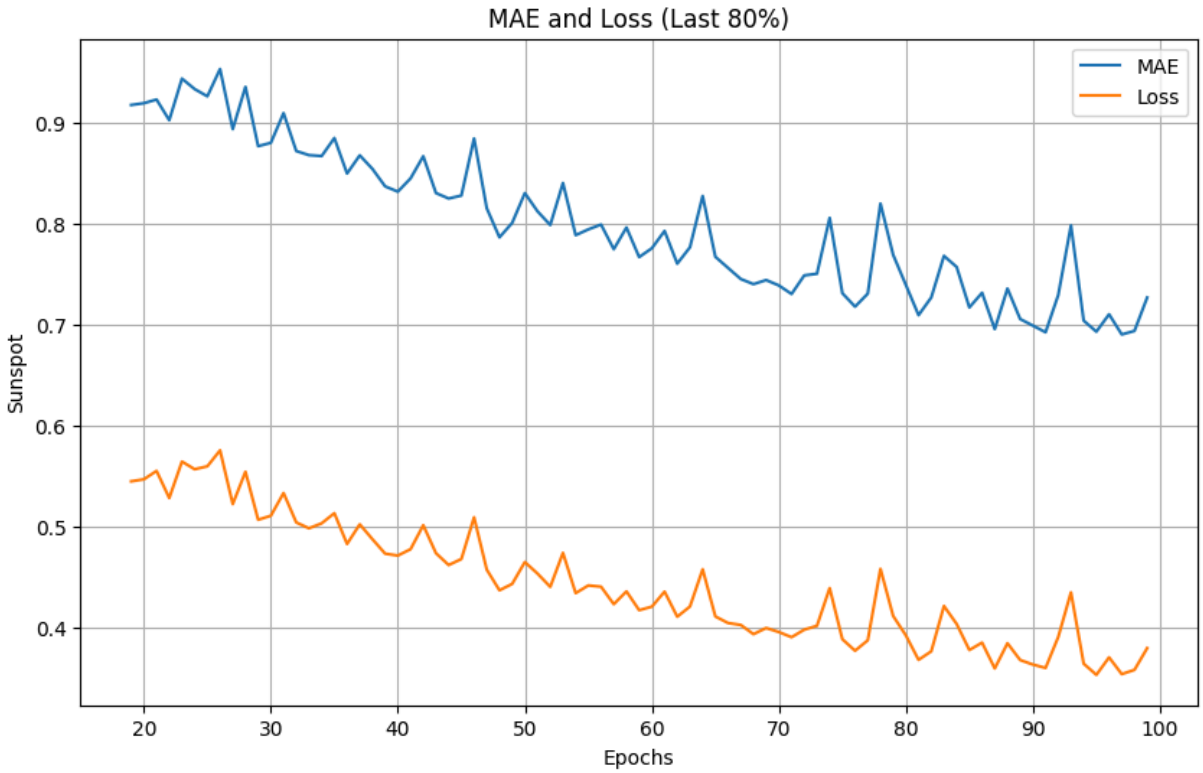




GRU Evaluation

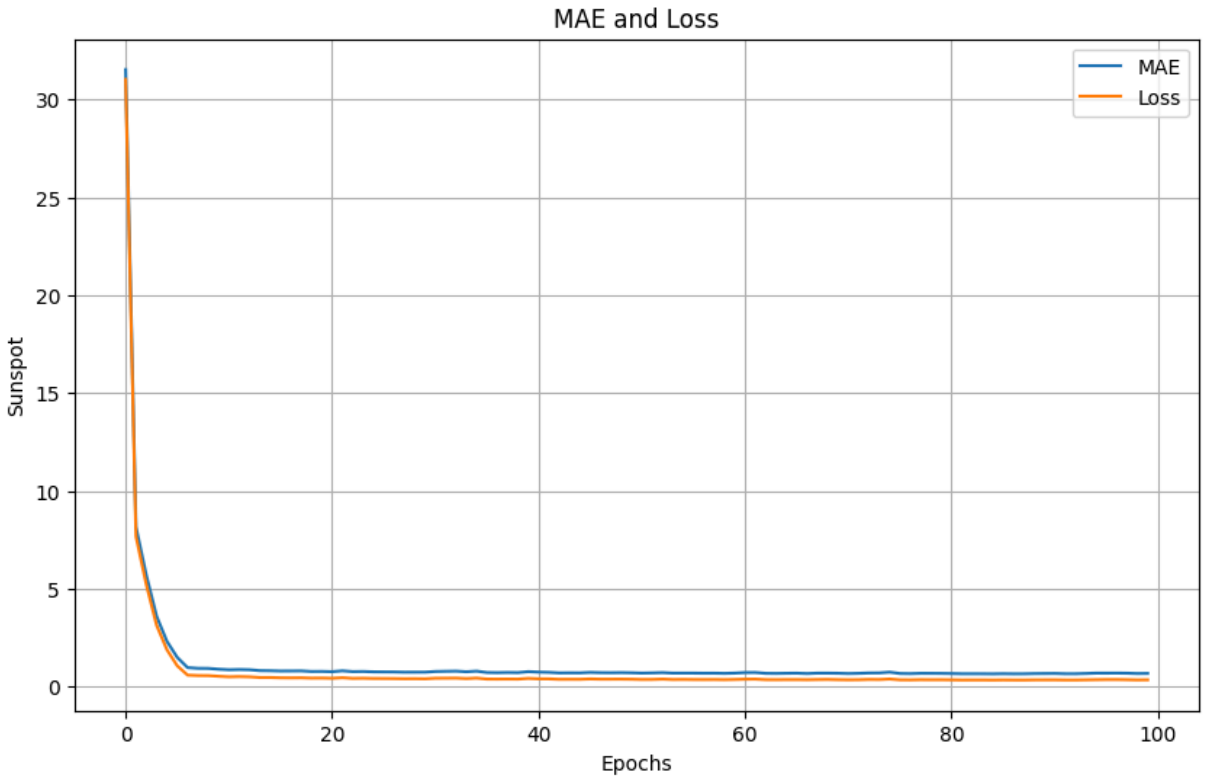
In [35]:
```
visualize_evaluation(history_GRU)
```

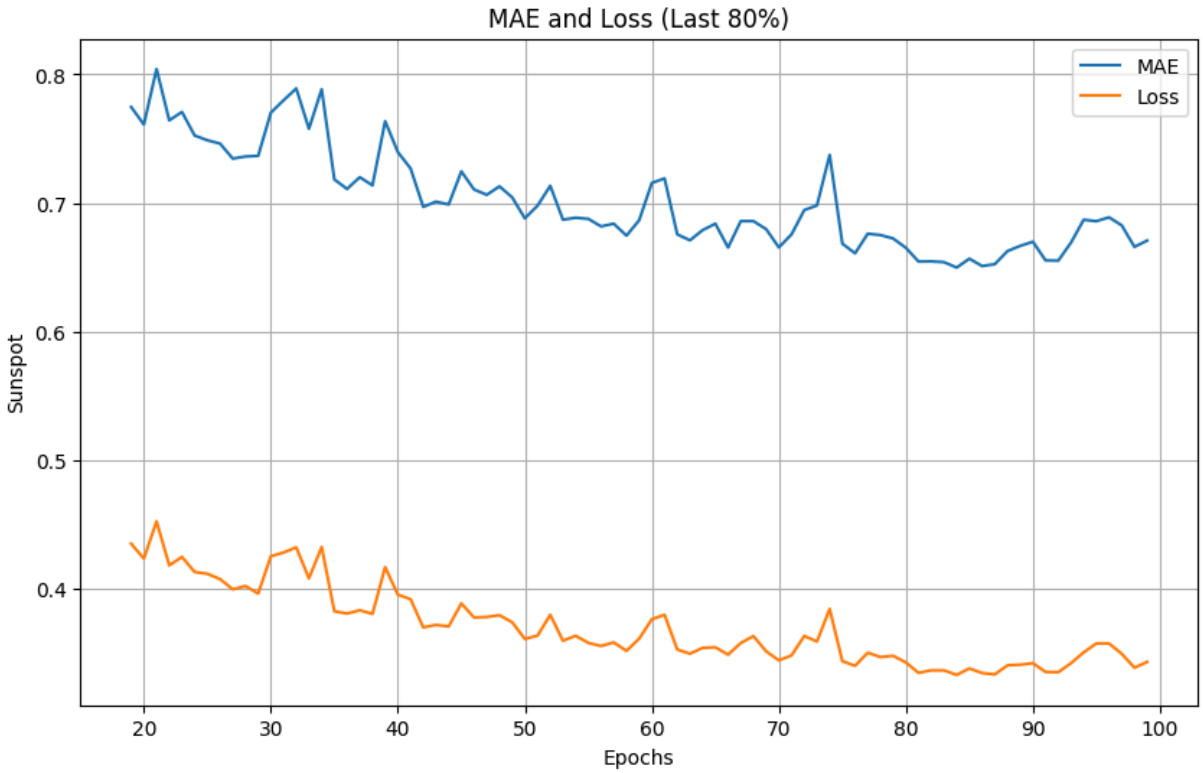




RNN-LSTM Evaluation

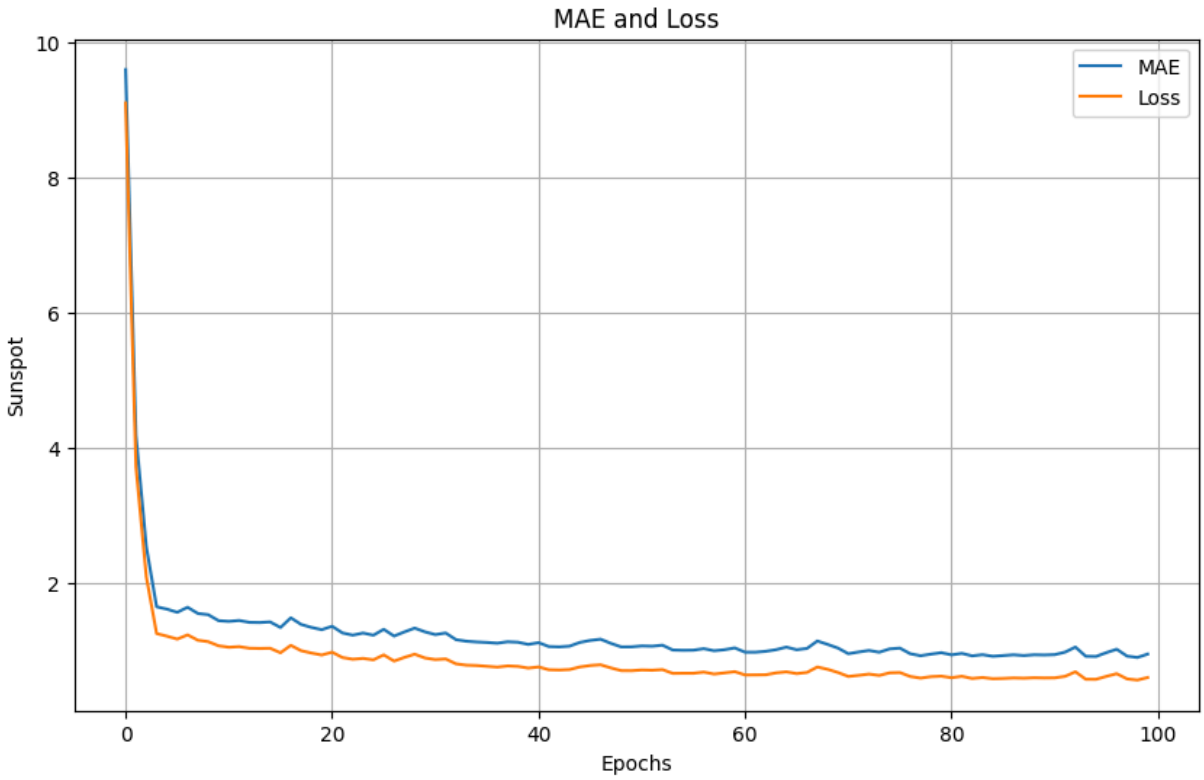
In [36]:
```
visualize_evaluation(history_RNN_LSTM)
```

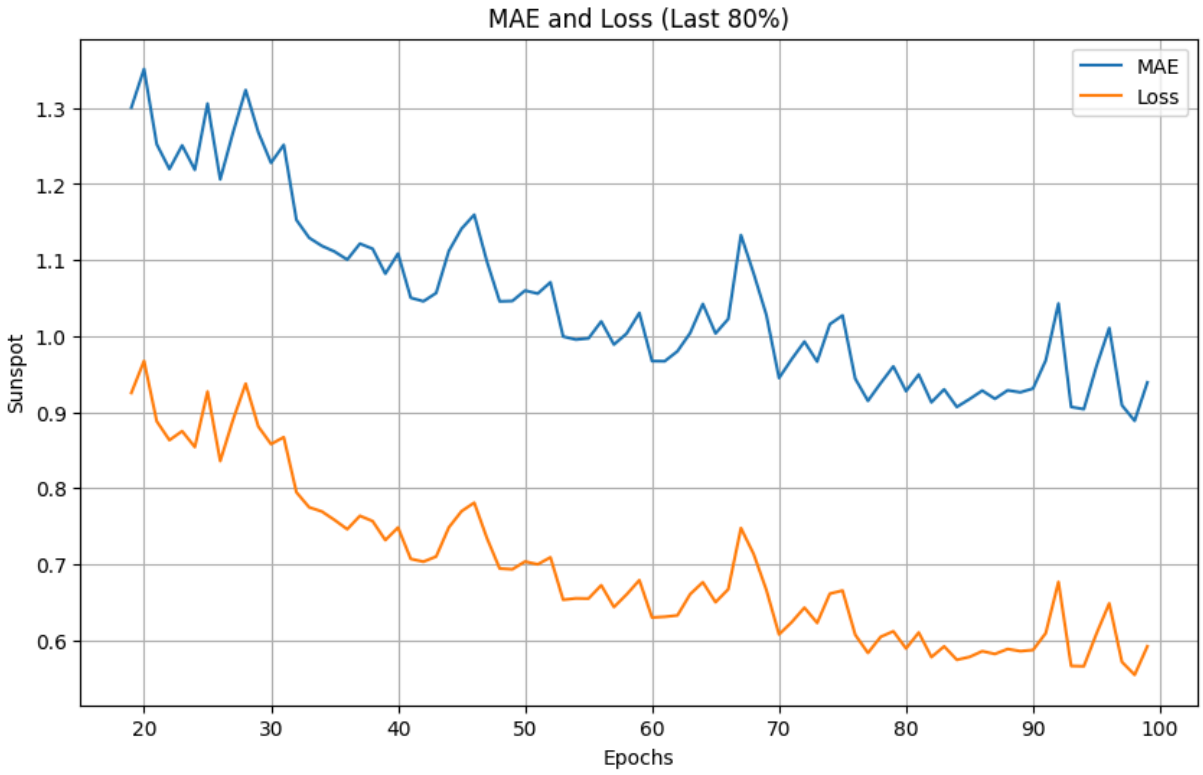




RNN-GRU Evaluation

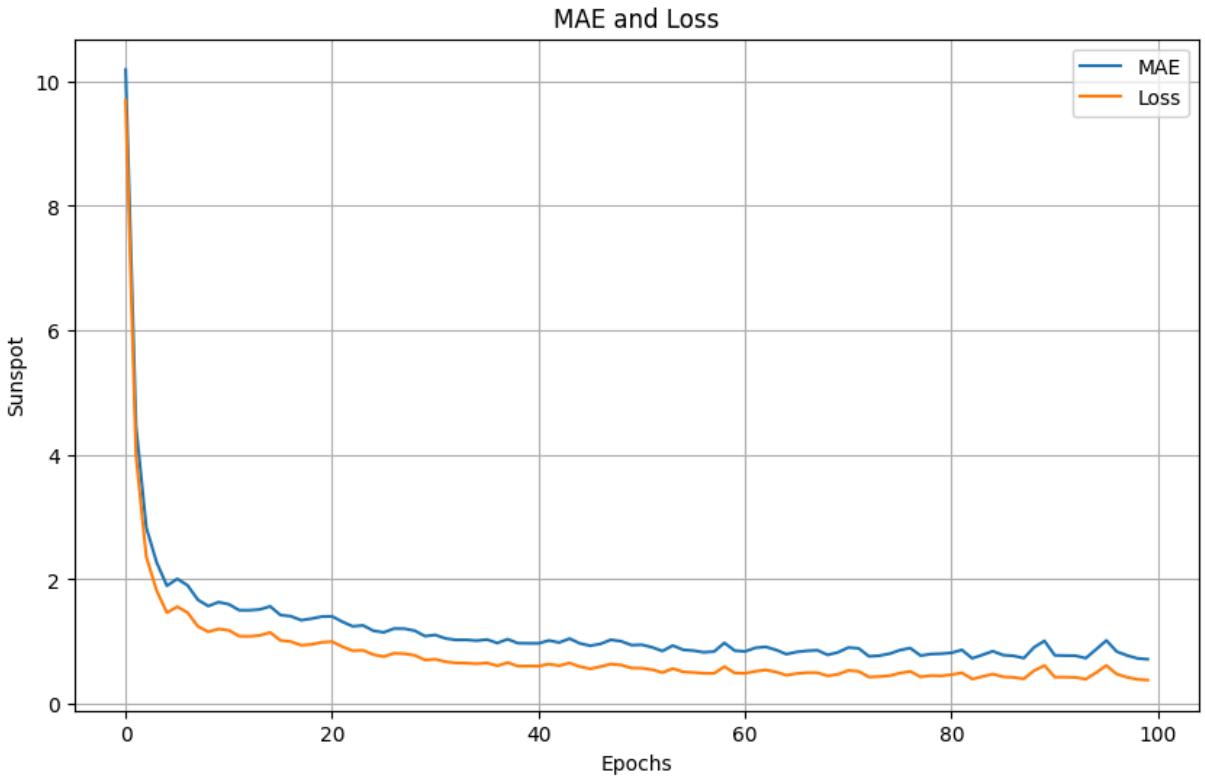
In [37]:
```
visualize_evaluation(history_RNN_GRU)
```

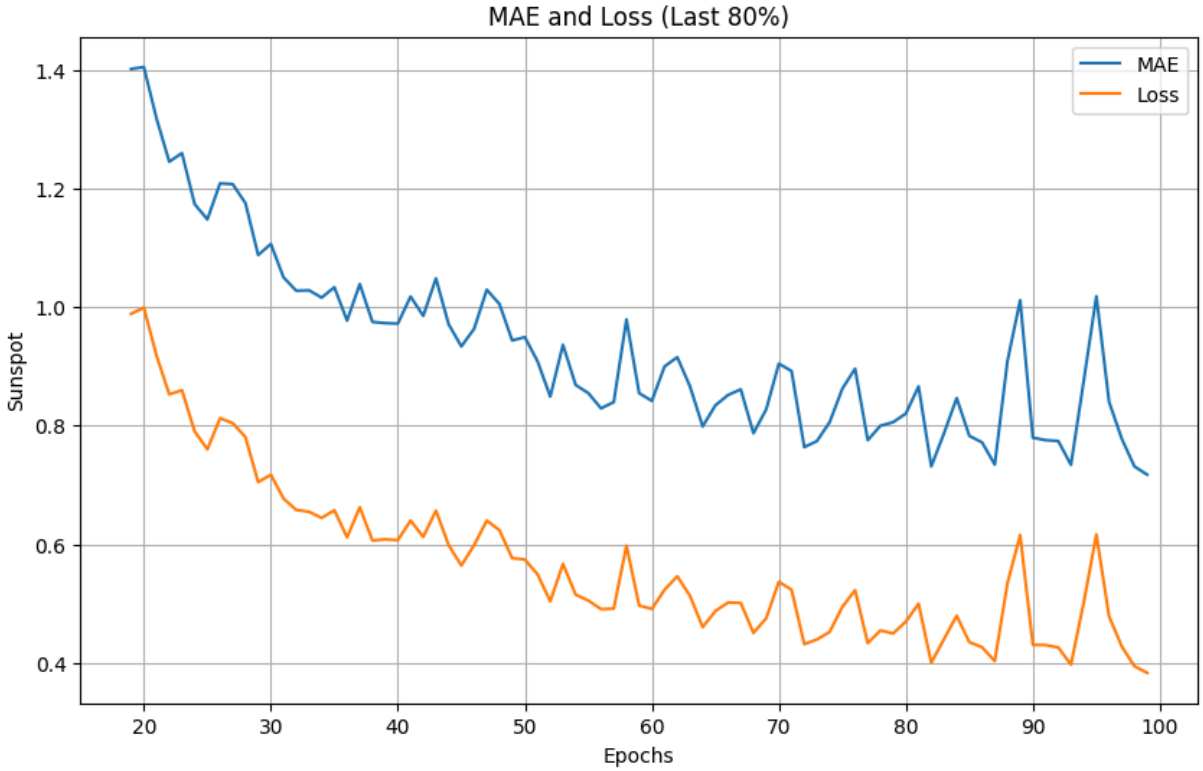




LSTM-RNN Evaluation

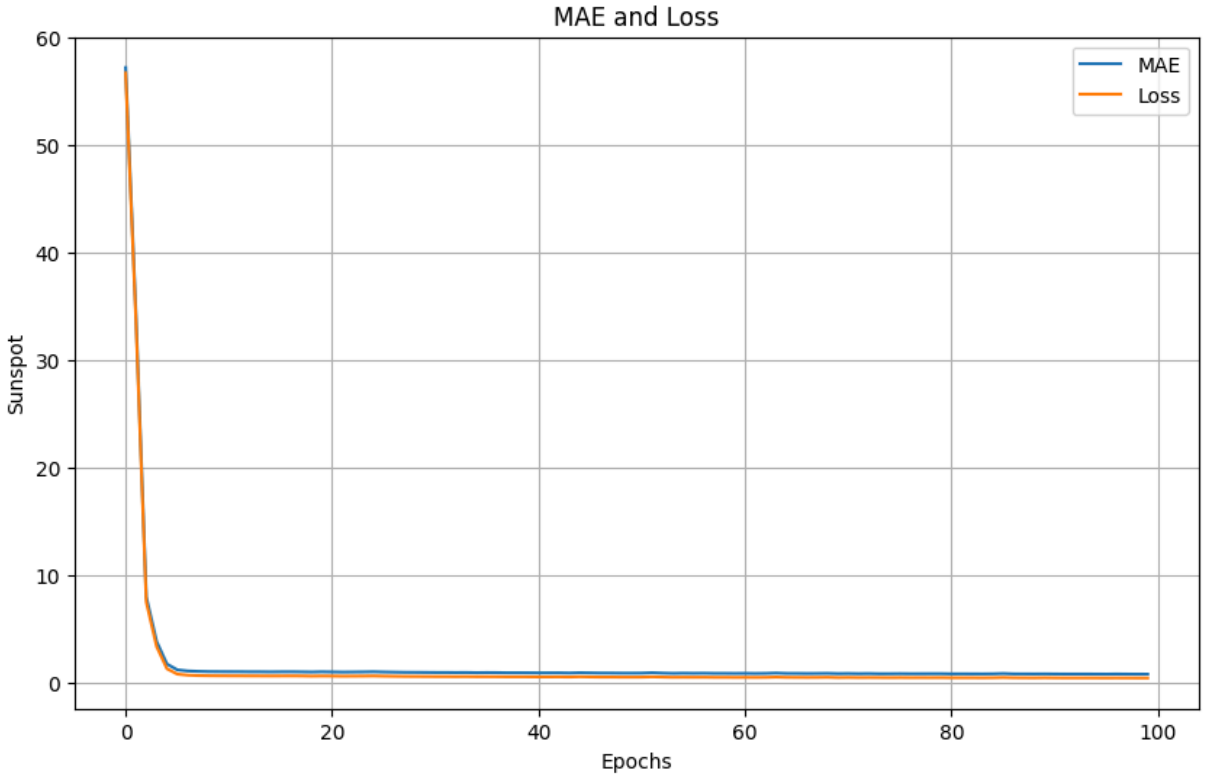
In [38]:
```
visualize_evaluation(history_LSTM_RNN)
```

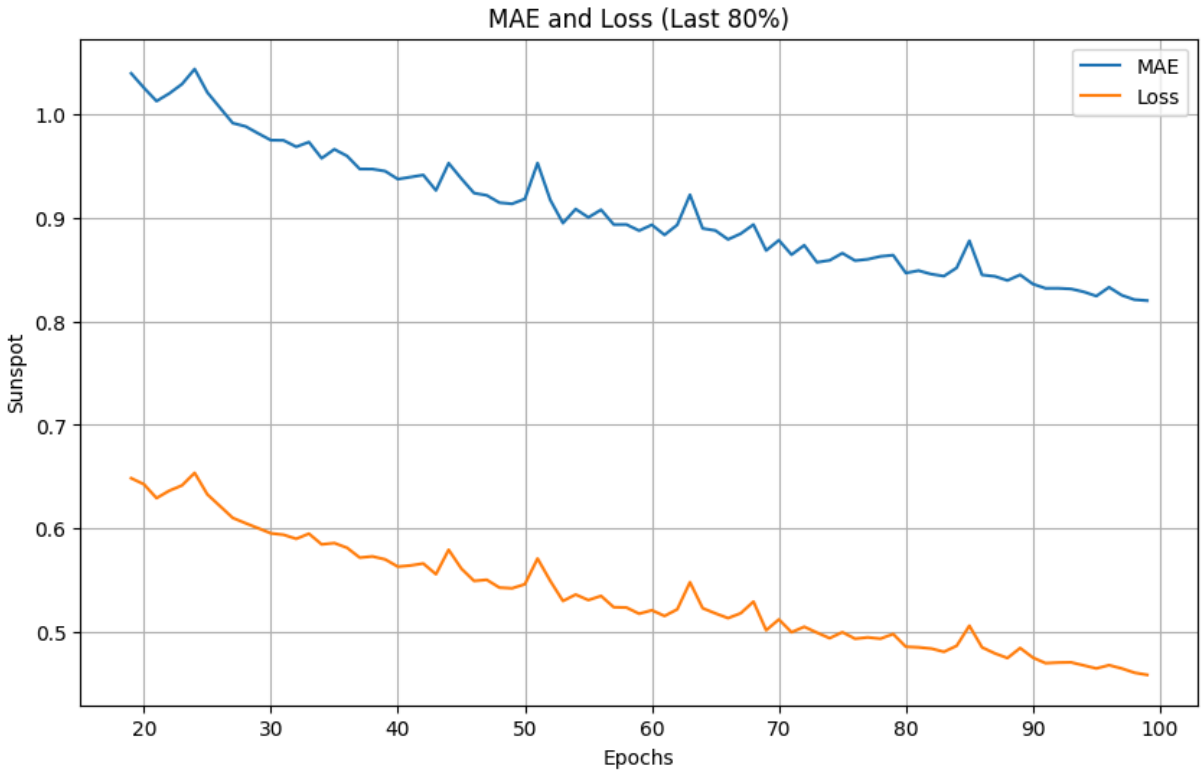




LSTM-GRU Evaluation

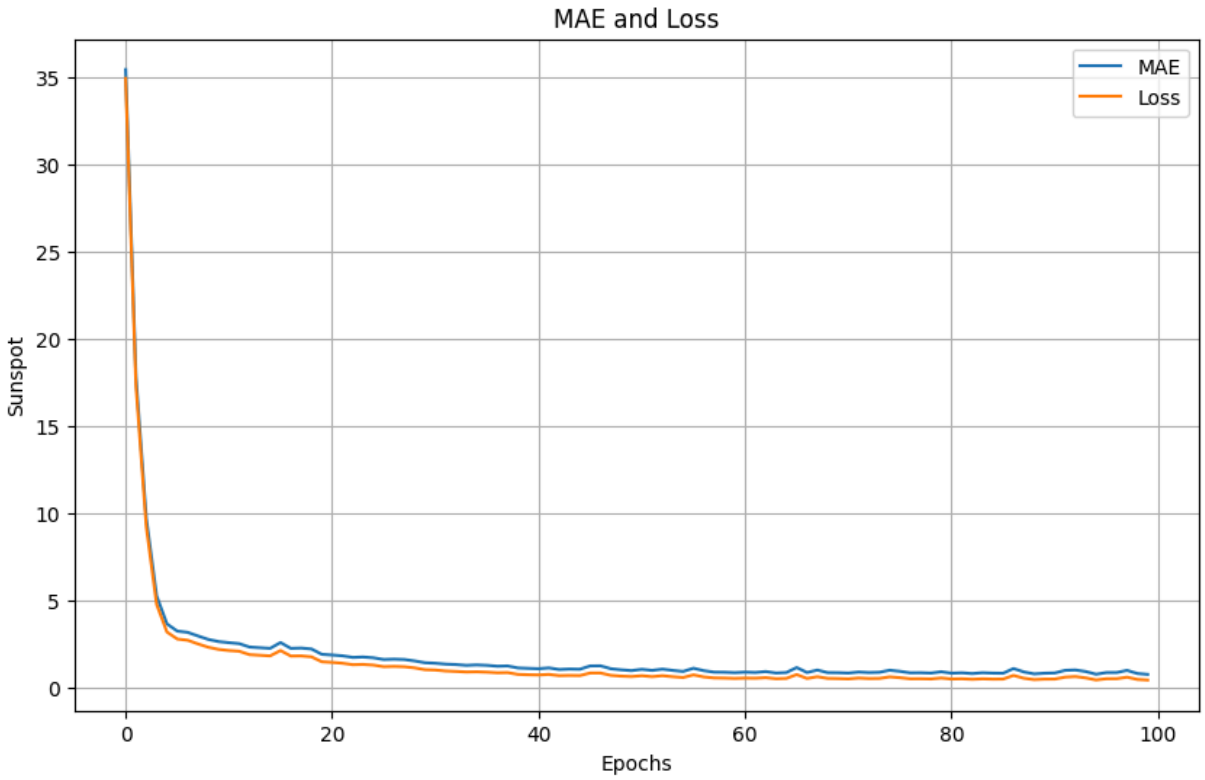
In [39]:
```
visualize_evaluation(history_LSTM_GRU)
```

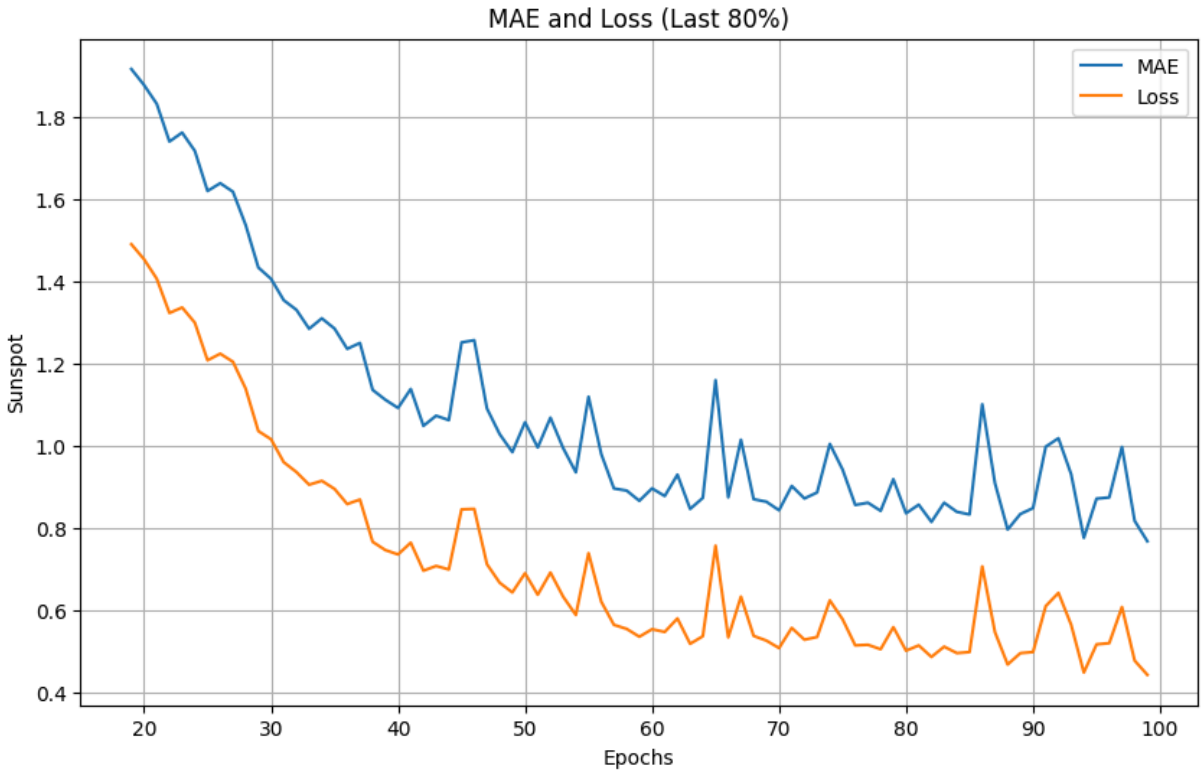




GRU-RNN Evaluation

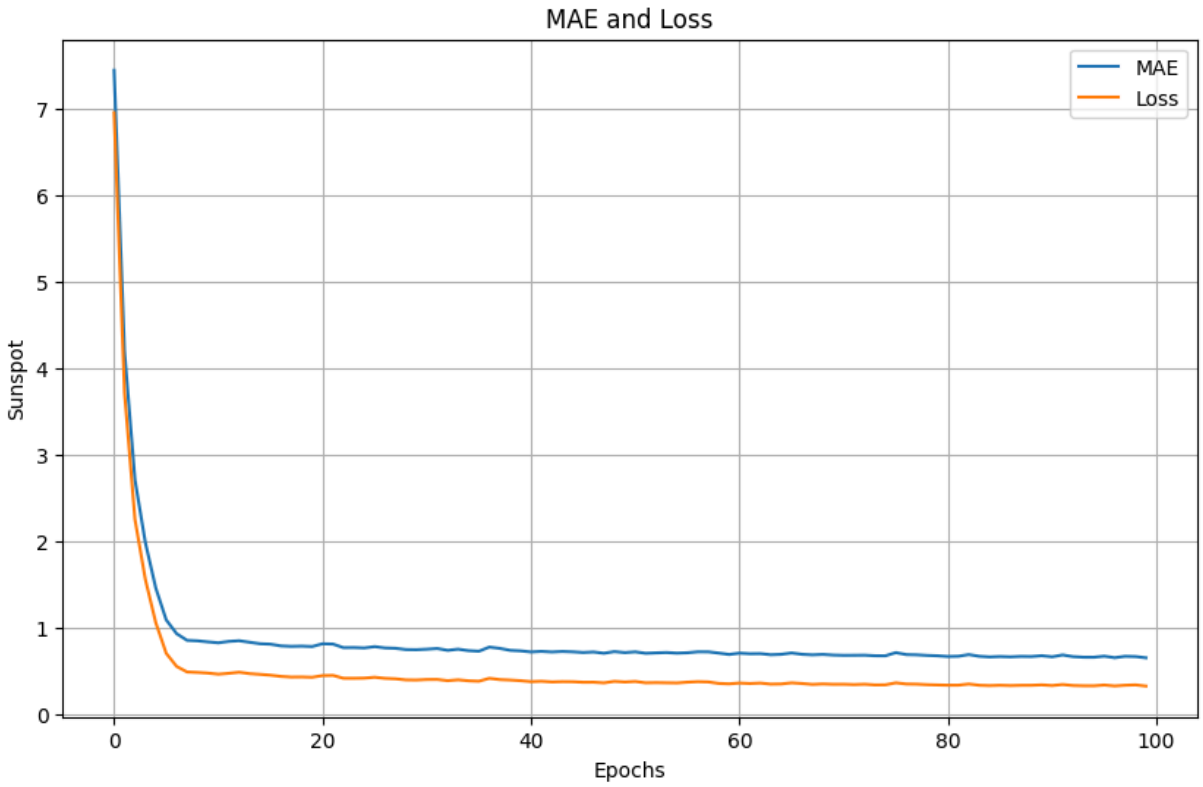
In [40]:
```
visualize_evaluation(history_GRU_RNN)
```

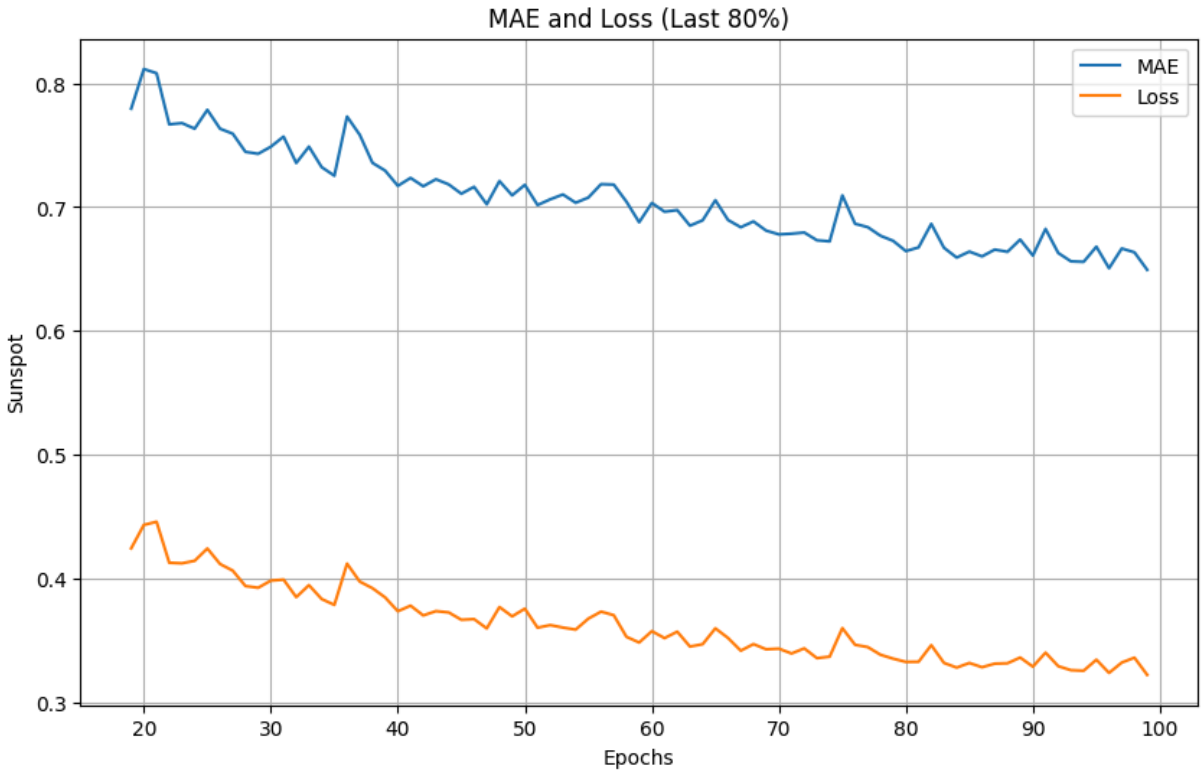




GRU-LSTM Evaluation

In [41]:
```
visualize_evaluation(history_GRU_LSTM)
```





7. Model Prediction

Analysis outcome of clustering for whale transaction characteristic

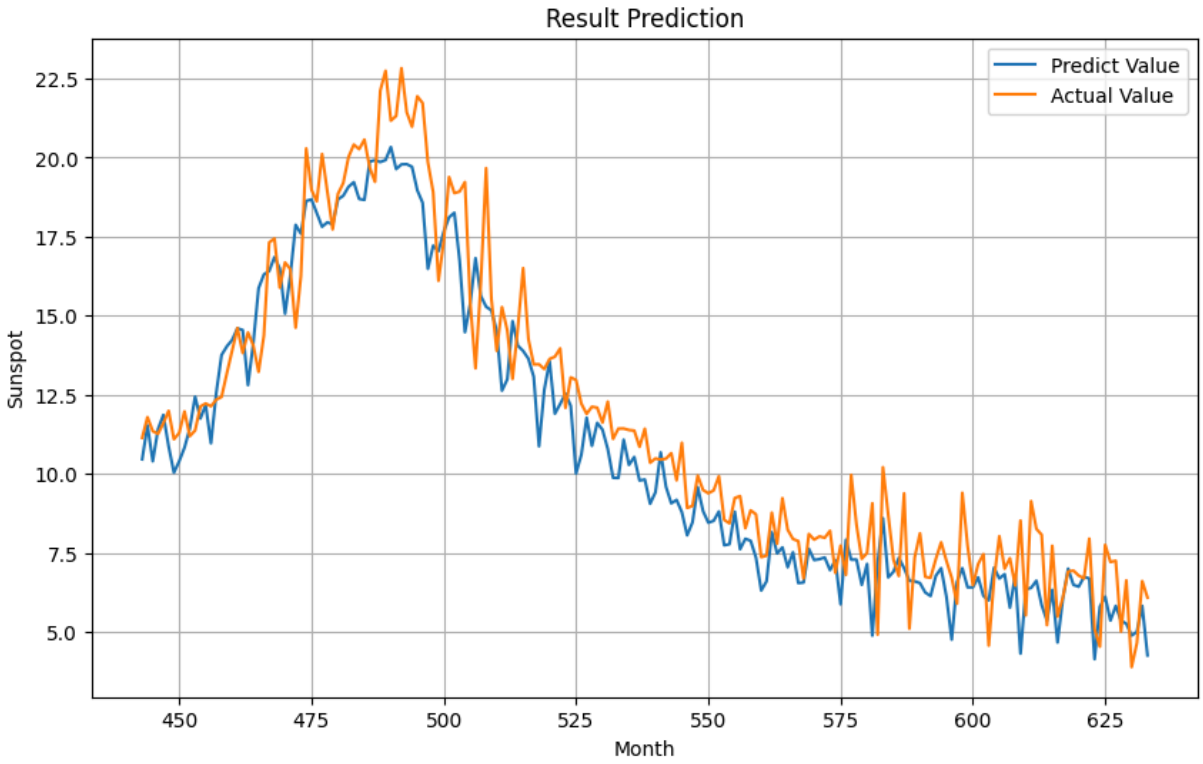
In [42]:
```
def model_forecast(model, series, window_size, batch_size):
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda w: w.batch(window_size))
    dataset = dataset.batch(batch_size).prefetch(1)
    forecast = model.predict(dataset)
    return forecast
```

In [43]:
```
def predict(model):
  forecast_series = series[split_time-window_size:-1]
  forecast = model_forecast(model, forecast_series, window_size, batch_size)
  results = forecast.squeeze()
  plot_series(time_valid, (x_valid, results, ), title='Result Prediction',
  xlabel='Month',
  ylabel='Sunspot',
  legend=['Predict Value', 'Actual Value'])
```

RNN Prediction

In [44]:
```
predict(model_RNN)
```

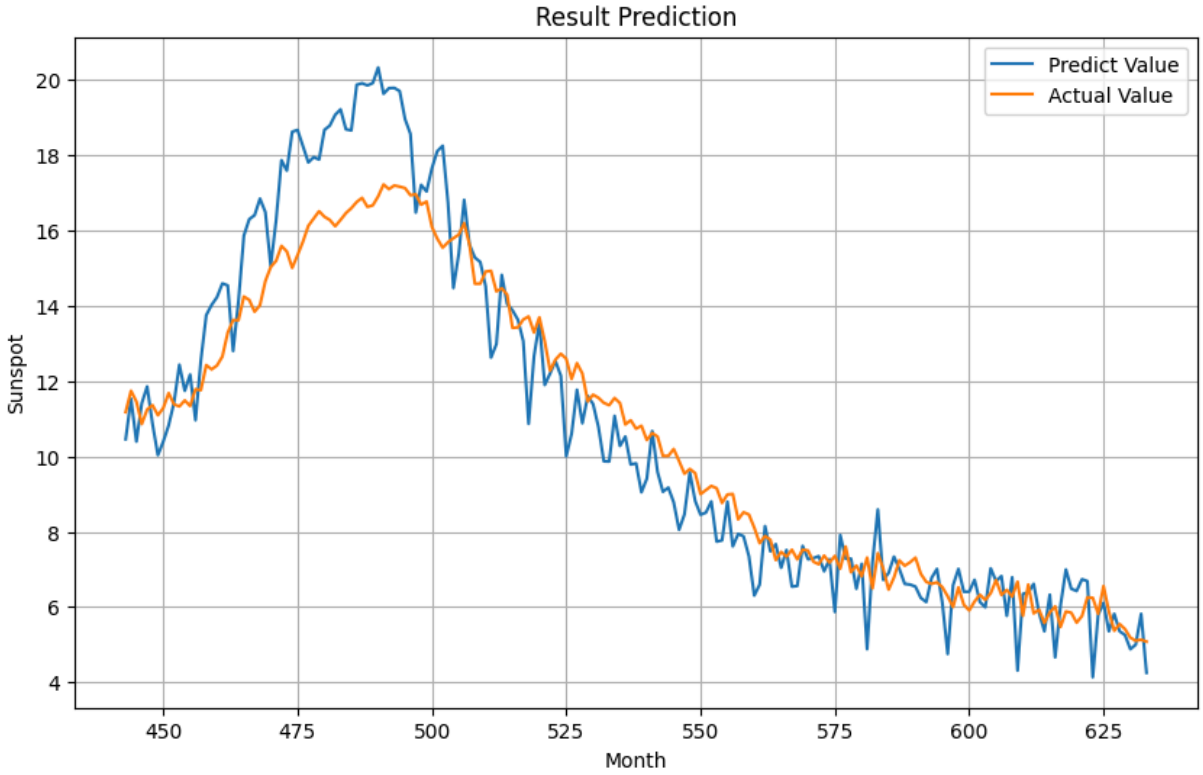
```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step
```



LSTM Prediction

In [45]:
```
predict(model_LSTM)
```

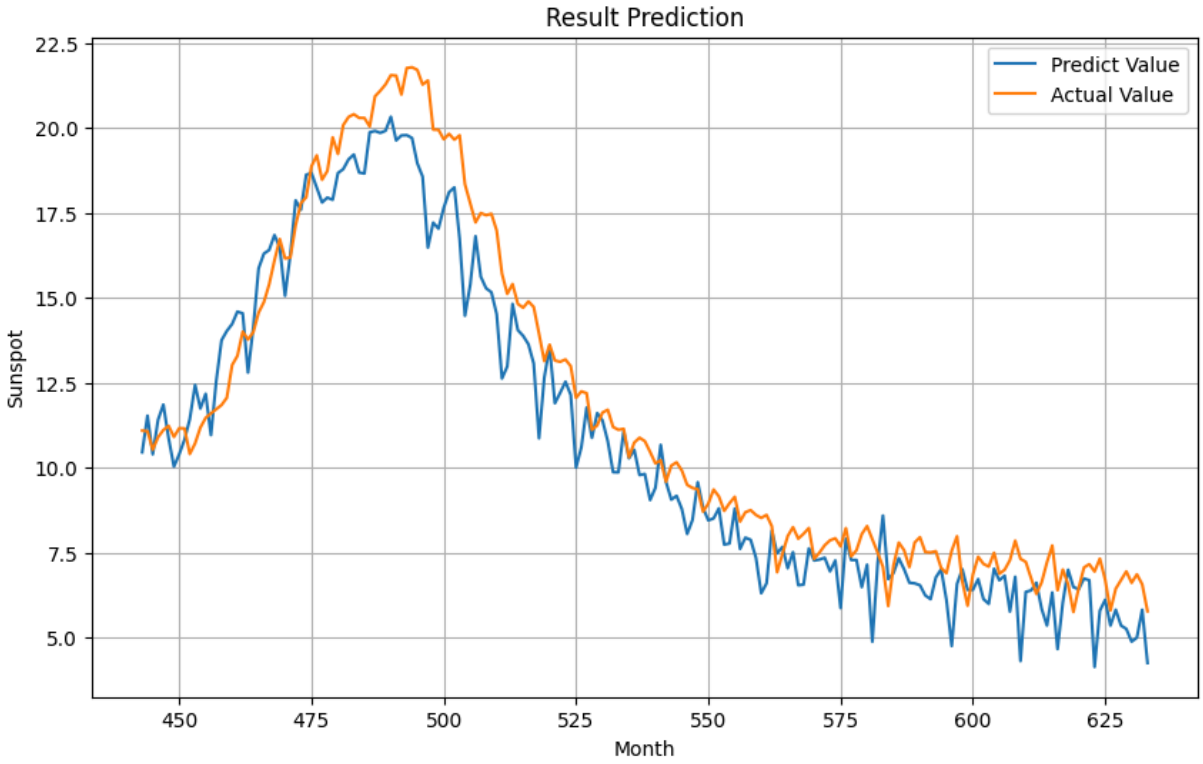
```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step
```



GRU Prediction

In [46]:
```
predict(model_GRU)
```

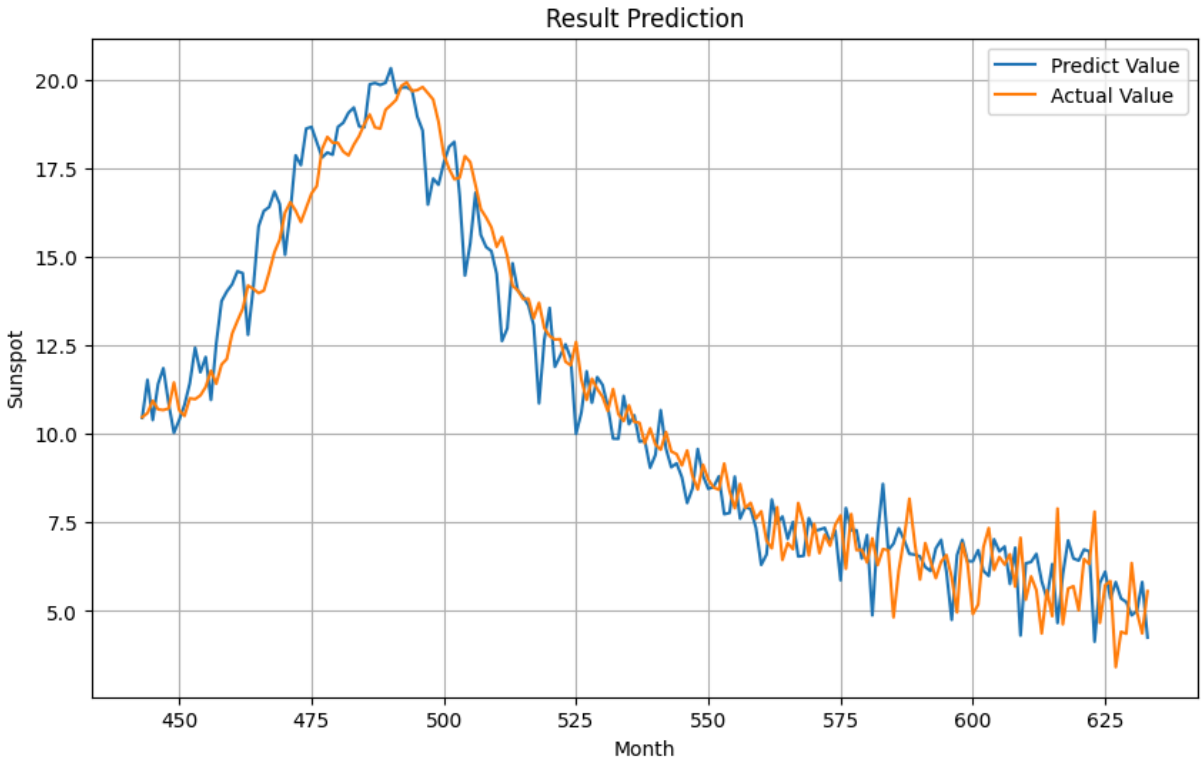
```
WARNING:tensorflow:5 out of the last 13 calls to <function TensorFlowTrainer.make_predict_funct
ion.<locals>.one_step_on_data_distributed at 0x7bdd492005e0> triggered tf.function retracing. T
racing is expensive and the excessive number of tracings could be due to (1) creating @tf.funct
ion repeatedly 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.tensor
flow.org/api_docs/python/tf/function for  more details.
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 73ms/step
```



RNN-LSTM Prediction

In [47]:
```
predict(model_RNN_LSTM)
```

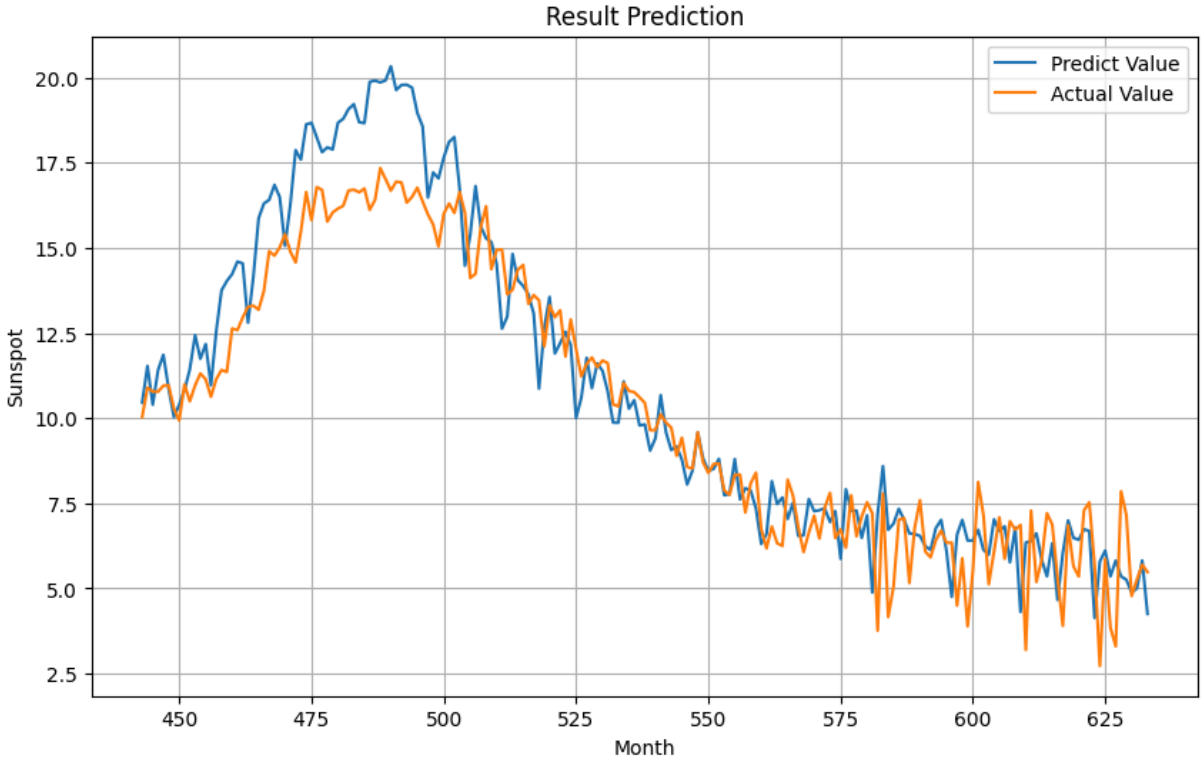
```
WARNING:tensorflow:5 out of the last 13 calls to <function TensorFlowTrainer.make_predict_funct
ion.<locals>.one_step_on_data_distributed at 0x7bdd49171c60> triggered tf.function retracing. T
racing is expensive and the excessive number of tracings could be due to (1) creating @tf.funct
ion repeatedly 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.tensor
flow.org/api_docs/python/tf/function for  more details.
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 57ms/step
```



RNN-GRU Prediction

In [48]:
```
predict(model_RNN_GRU)
```

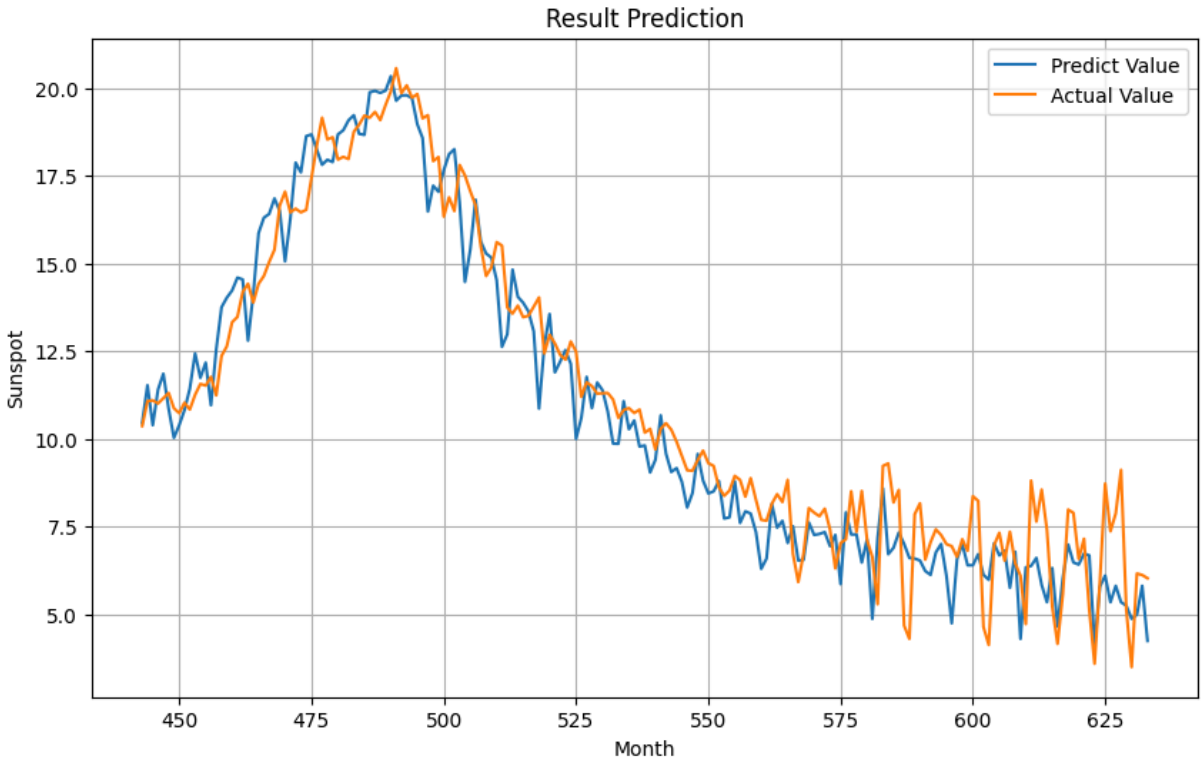
```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step
```



LSTM-RNN Prediction

In [49]:
```
predict(model_LSTM_RNN)
```

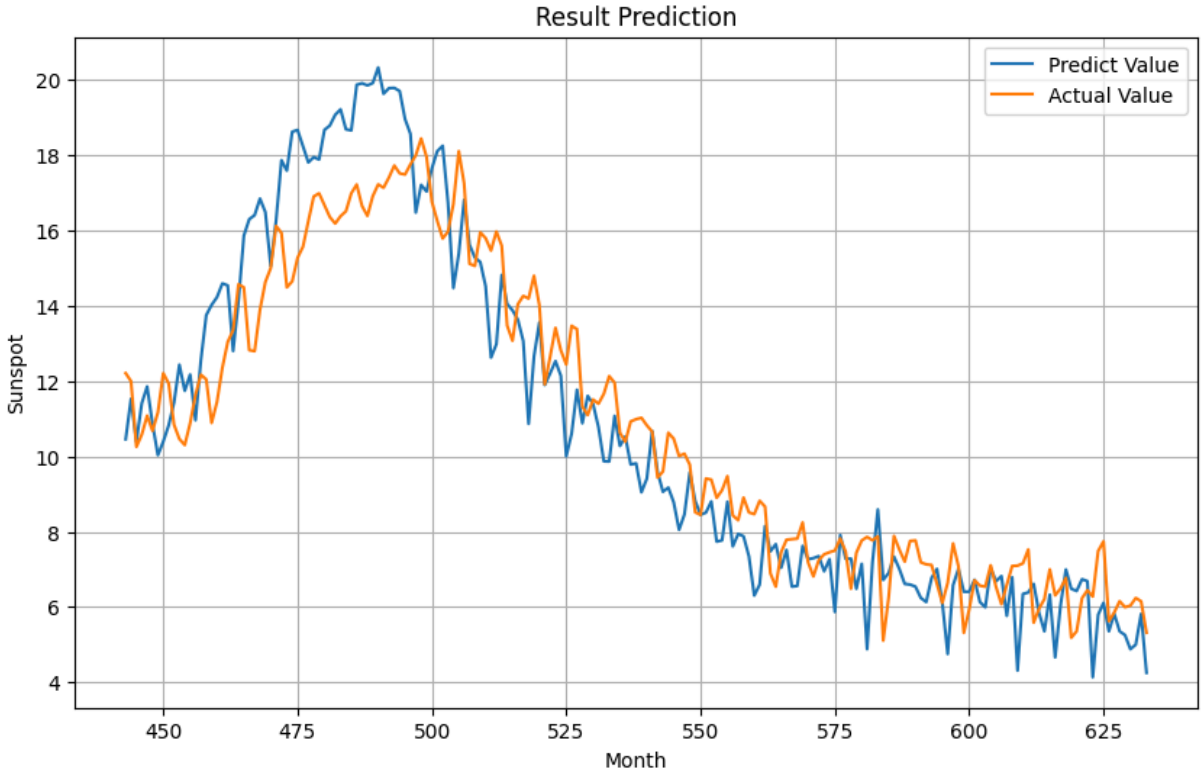
```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 91ms/step
```



LSTM-GRU Prediction

In [50]:
```
predict(model_LSTM_GRU)
```

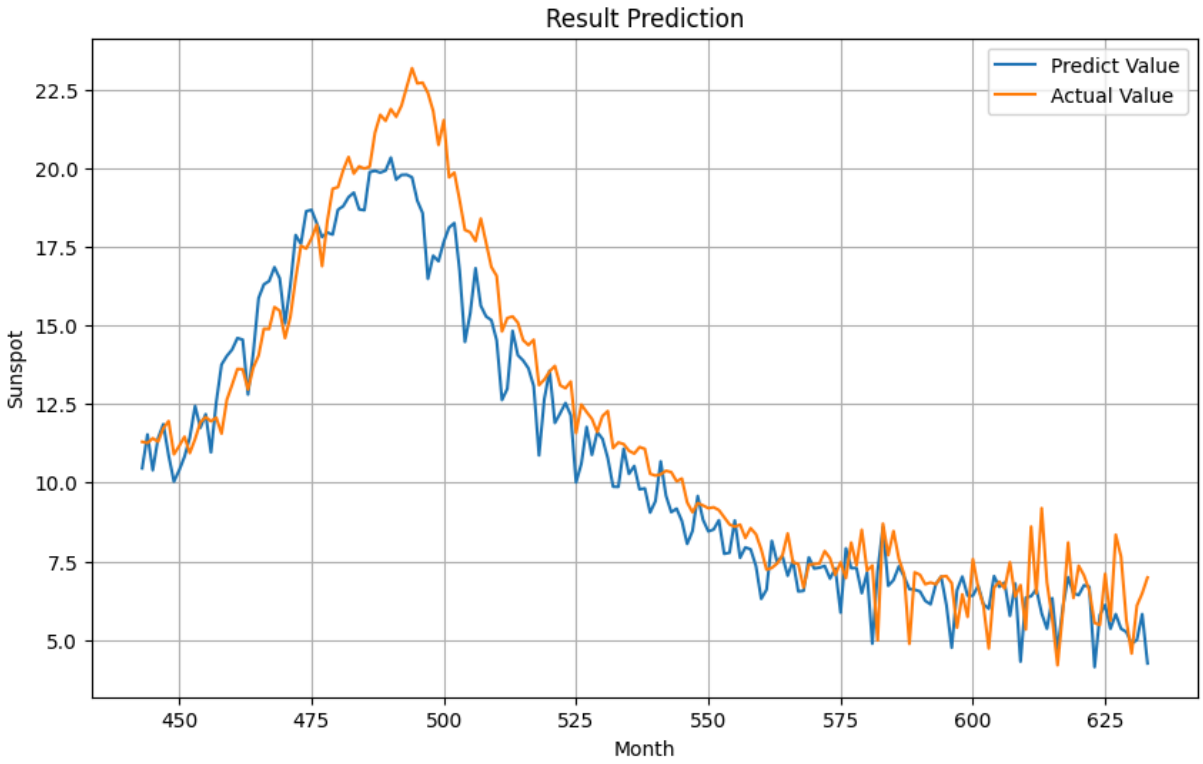
```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 106ms/step
```



GRU-RNN Prediction

In [51]:
```
predict(model_GRU_RNN)
```

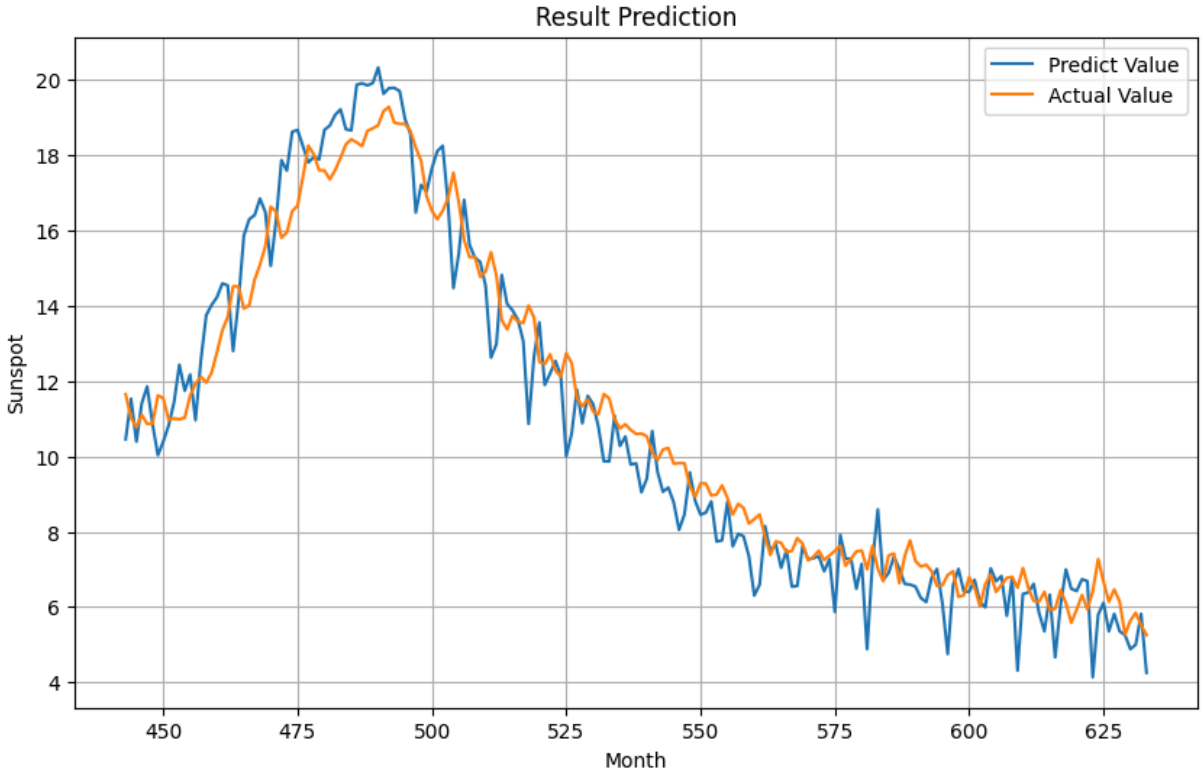
```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step
```



GRU-LSTM Prediction

In [52]:
```
predict(model_GRU_LSTM)
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 74ms/step
```



Confident Interval

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import files
import time
```

Donwload Data

In [ ]:
```
!wget https://github.com/kkms51/data/raw/main/GRU_LSTM_TIME.xlsx
!wget https://github.com/kkms51/data/raw/main/GRU_RNN_TIME.xlsx
!wget https://github.com/kkms51/data/raw/main/GRU_TIME.xlsx
!wget https://github.com/kkms51/data/raw/main/LSTM_GRU_TIME.xlsx
!wget https://github.com/kkms51/data/raw/main/LSTM_RNN_TIME.xlsx
!wget https://github.com/kkms51/data/raw/main/RNN_LSTM_TIME.xlsx
!wget https://github.com/kkms51/data/raw/main/RNN_GRU_TIME.xlsx
```

In [ ]:
```
!gdown 1OusiKKuYGBZxP41JqALwjS5fiZ5bVET8
!gdown 17Z12L9mQh91d7wz5tu2V4ywzrVJ91hRj
!gdown 1sgrbiaGJrwF-AEXEr8R5x2Ulk95dq56K
!gdown 11dDMmQ8RWVDHcZuNyEeWlC-k0nHx5pv4
!gdown 1RCRBAGpTxkxruxc-XBDxBUq5A-eIS7RB
!gdown 1jzDhLE81rWPZffIwBg-MvuMqVPpg-bY-
!gdown 1ar8H4y9QWquY6zM_cxpk3vp_aDy6gcjU
!gdown 1Lmaa9hUtdTD_OFdC4aRwfYxtMjjLWU7i
!gdown 1ar8H4y9QWquY6zM_cxpk3vp_aDy6gcjU
!gdown 1Lmaa9hUtdTD_OFdC4aRwfYxtMjjLWU7i
!gdown 1Fg9-g0HfM6OPAIBetFllDdiHOQDU7_Pe
!gdown 1uHwTBW5IhPG9cnnBN3P5vGAW0HlJEUfG
!gdown 1xOn5Jvk_5hzaLVbmCWMSulcEqPzJNm29
!gdown 1RkZQ50WhKxoCRswyMTEDp-Jpfbwm5six
!gdown 1-uCIfgqEUmM2EOiFb9v7iBVRL1kMMXj1
!gdown 18hpQAoeUicSeYaJb0SKuHj-Op6LAHXrG
!gdown 1gXJIGhIKKkPERaecvST5rX16lrnqGZ1j
!gdown 1MGNVSDpfEerjj9fOlZI3LYkLxN6YHGiS
```

Viz

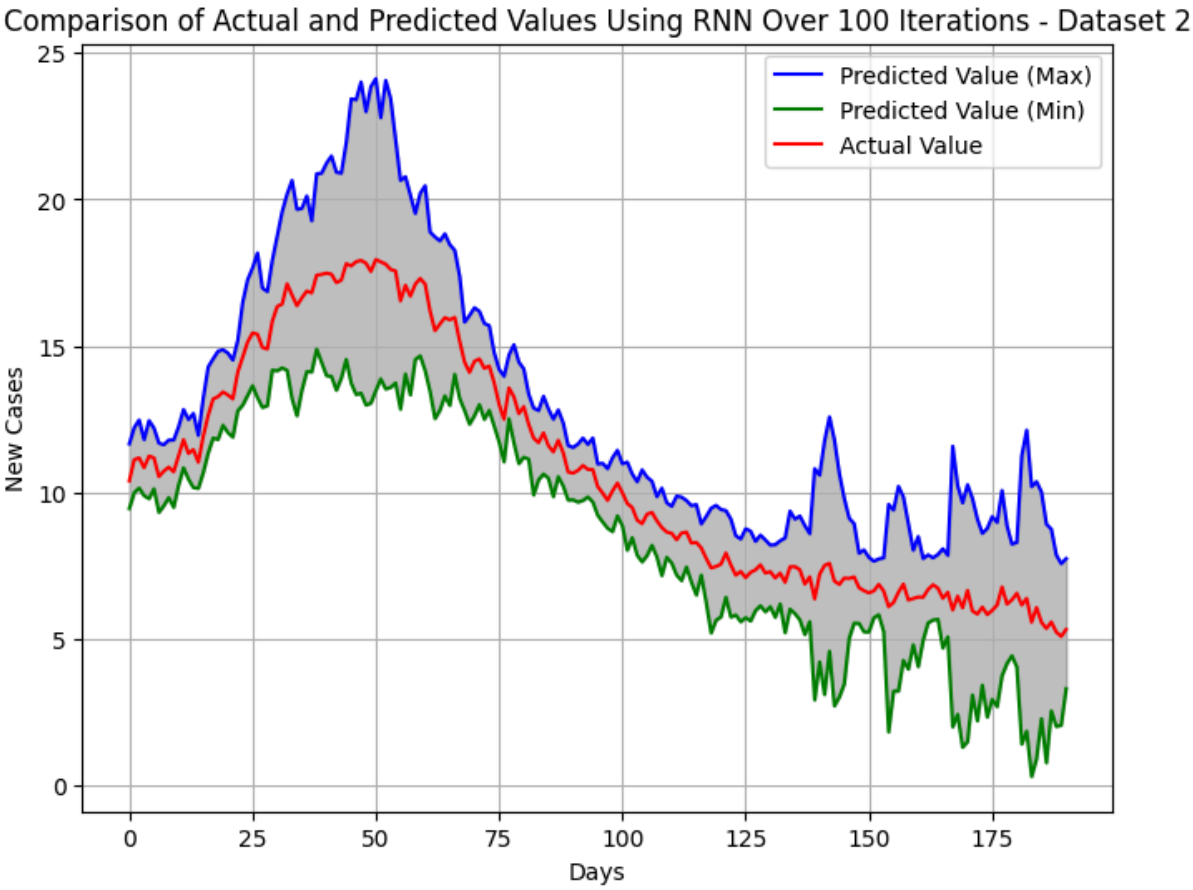
In [4]:
```
from re import X
def shade_plot(temp, x_valid='0', line='MAE', predict=''):
  import matplotlib.pyplot as plt

  if predict == '':
    redline = f'Mean {line}'
    axis = 'Epochs'
  else:
    redline = 'Actual Value'
    axis = 'Days'

  #SORT
  temp = temp.apply(lambda x: x.sort_values().reset_index(drop=True))

  #CUT 2.5%
  n = int(len(temp) * 0.025)
  temp = temp.iloc[n:-n]

  batas_atas = temp.max().tolist()
  batas_bawah = temp.min().tolist()
  rata = temp.mean().tolist()

  # Replace these lists with your actual data
  # Dataset 1
  x1 = [i for i in range(0,len(batas_atas))]
```

```
  y1 = batas_atas

  # Dataset 2
  x2 = [i for i in range(0,len(batas_bawah))]
  y2 = batas_bawah

  if(x_valid=='0'):
    # Dataset 3
    x3 = [i for i in range(0,len(rata))]
    y3 = rata
  else:
    x3 = [i for i in range(0,len(x_valid))]
    y3 = x_valid

  # Create the plot
  plt.figure(figsize=(8, 6))

  # Plot the first linear dataset
  plt.plot(x1, y1, label=f'Predicted Value (Max)', color='blue')

  # Plot the second linear dataset
  plt.plot(x2, y2, label=f'Predicted Value (Min)', color='green')

  plt.plot(x3, y3, label=f'{redline}', color='red')

  # Create a mask for the shaded area where y1 >= y2
  mask = [y1_val >= y2_val for y1_val, y2_val in zip(y1, y2)]

  # Shade the area between the two datasets
  plt.fill_between(x1, y1, y2, where=mask, interpolate=True, color='gray', alpha=0.5)

  # Add labels and legend
  plt.xlabel(f'{axis}')
  plt.ylabel('New Cases')
  plt.title(f'Comparison of Actual and Predicted Values Using {line} Over 100 Iterations - Data
  plt.legend()
  plt.savefig(f"Predicted_Actual_Value_Dataset2_{line}.pdf", dpi=300)
  files.download(f"Predicted_Actual_Value_Dataset2_{line}.pdf")
  # Show the plot
  plt.grid(True)
  plt.show()
```

In [ ]:
```
!wget https://raw.githubusercontent.com/MhdIqbalPratama/ware-house/main/Data%20COVID-19%20Indon
```

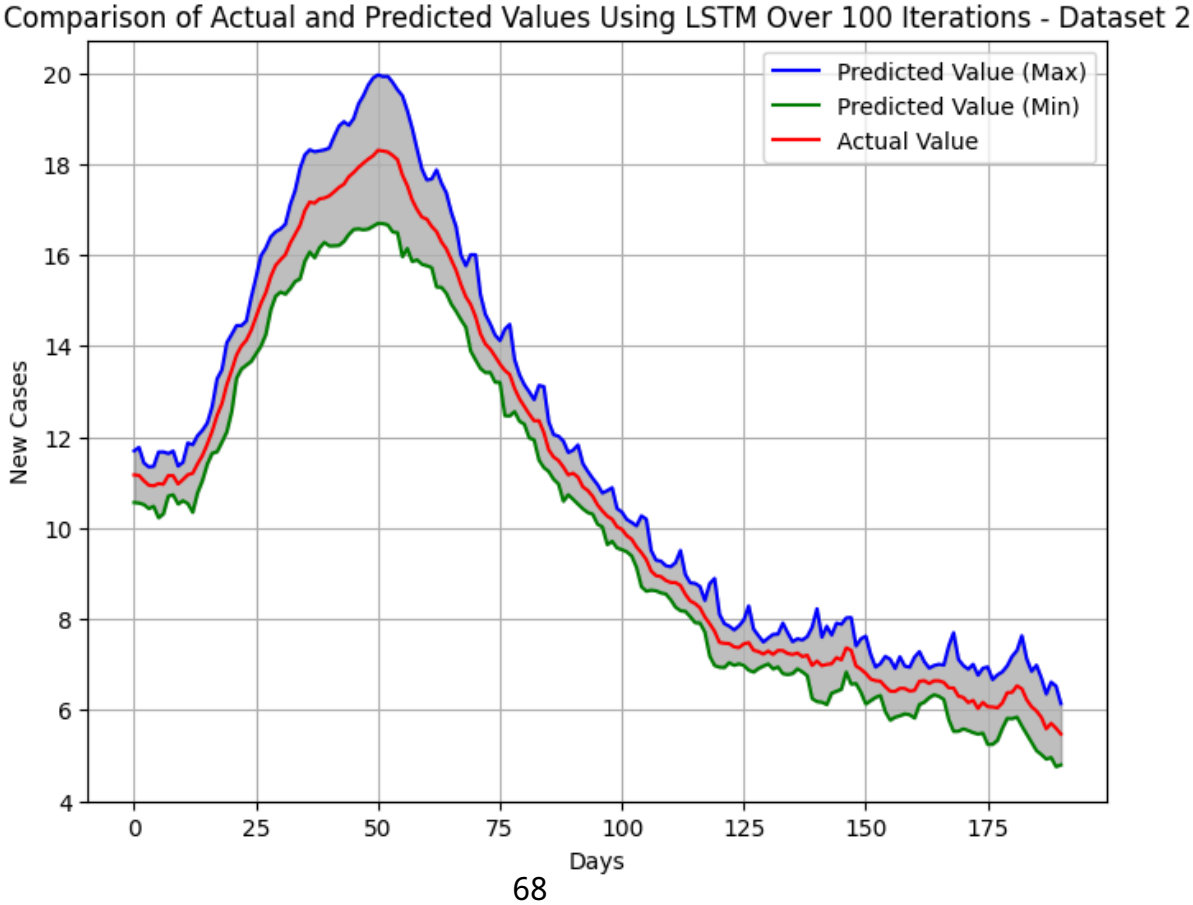
RNN

In [13]:
```
temp = pd.read_excel('RNN_PREDICT.xlsx')
shade_plot(temp, line='RNN', predict='YES')
```



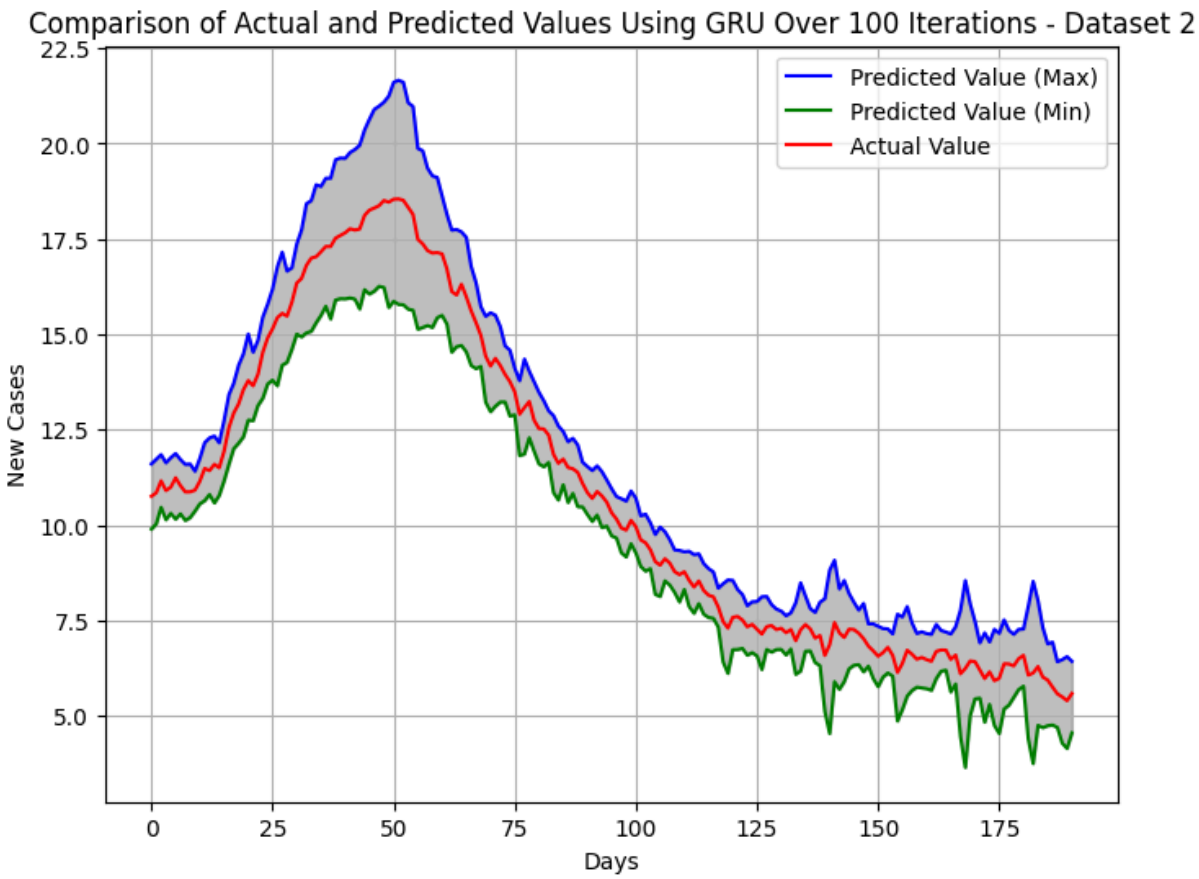
LSTM

In [18]:
```
temp = pd.read_excel('LSTM_PREDICT.xlsx')
shade_plot(temp, line='LSTM', predict='YES')
```



GRU

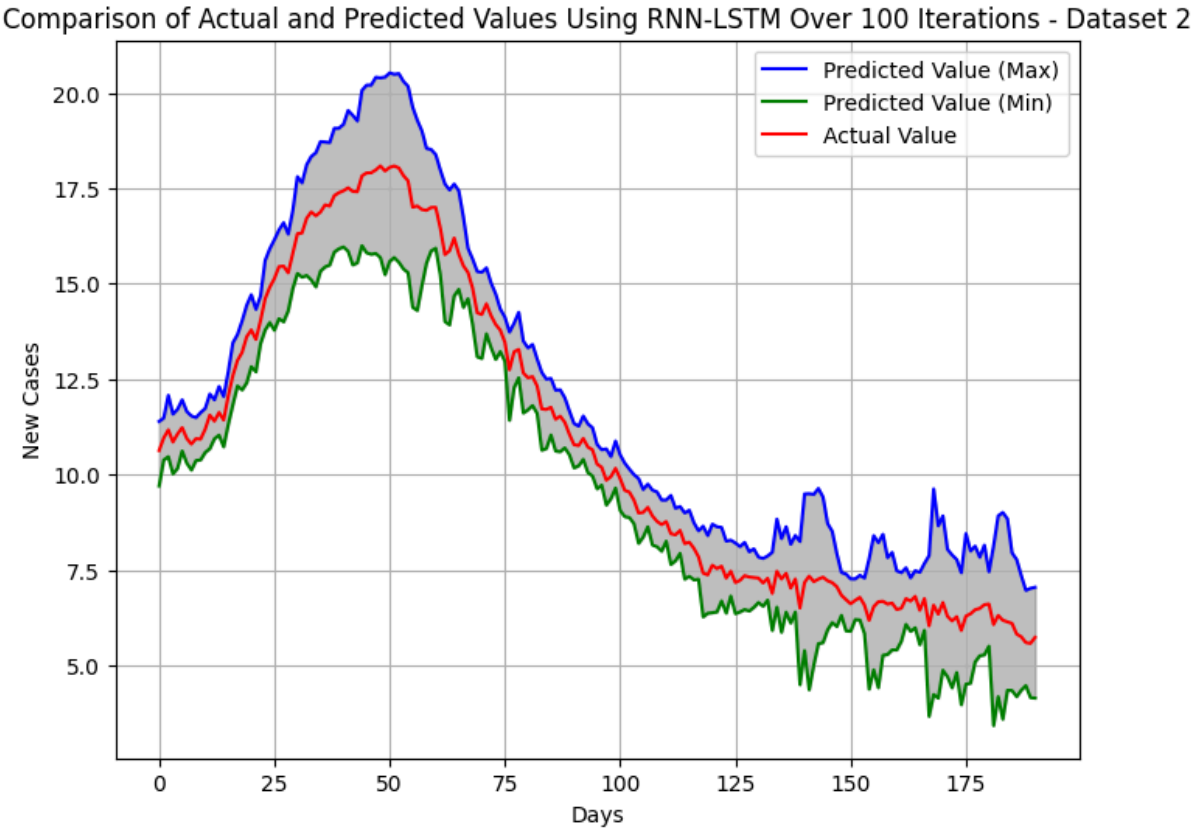
In [23]:
```
temp = pd.read_excel('GRU_PREDICT.xlsx')
shade_plot(temp, line='GRU', predict='YES')
```



In [ ]:
```
!gdown 1Jcj7_vtMQ2foct0YPmWC6nO9q4T_x3bl
!gdown 1d-M7kCjRcggRBII-8o9WUhbWPaFxcPRg
```

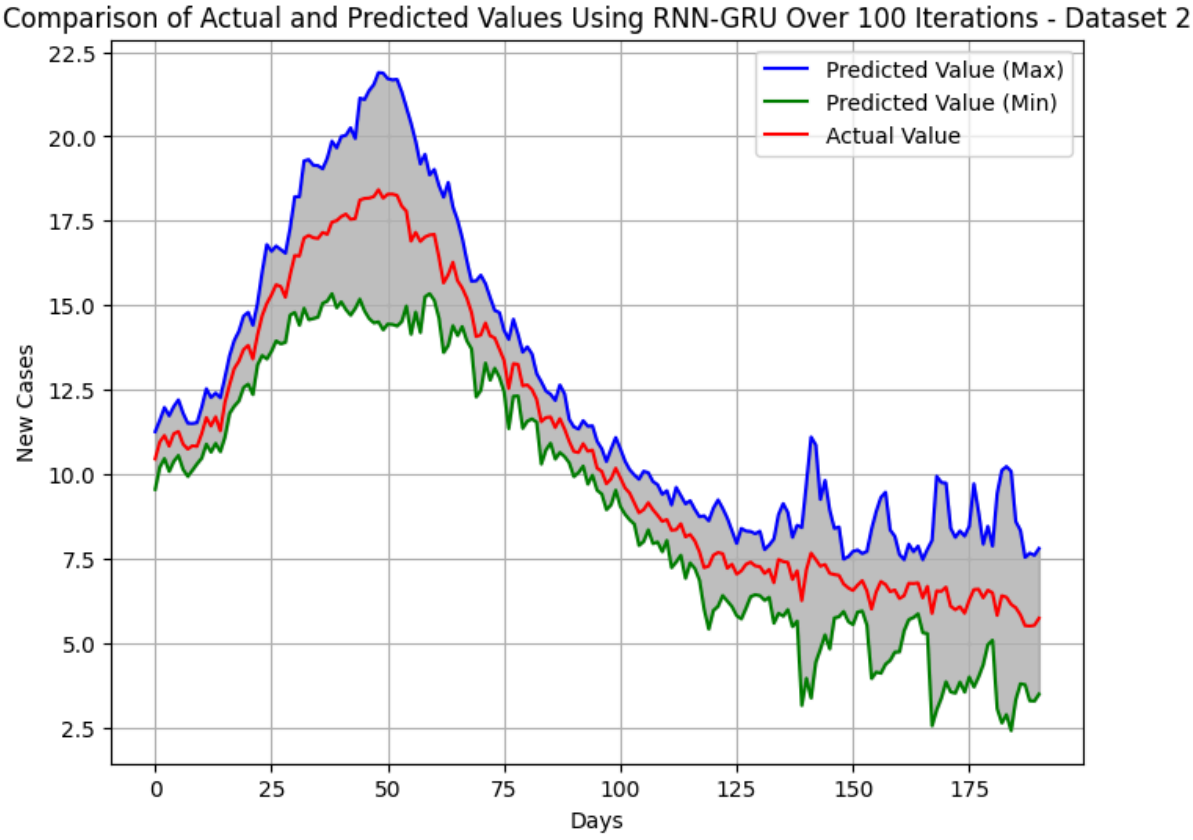
RNN-LSTM

In [29]:
```
temp = pd.read_excel('RNN_LSTM_PREDICT.xlsx')
shade_plot(temp, line='RNN-LSTM', predict='YES')
```



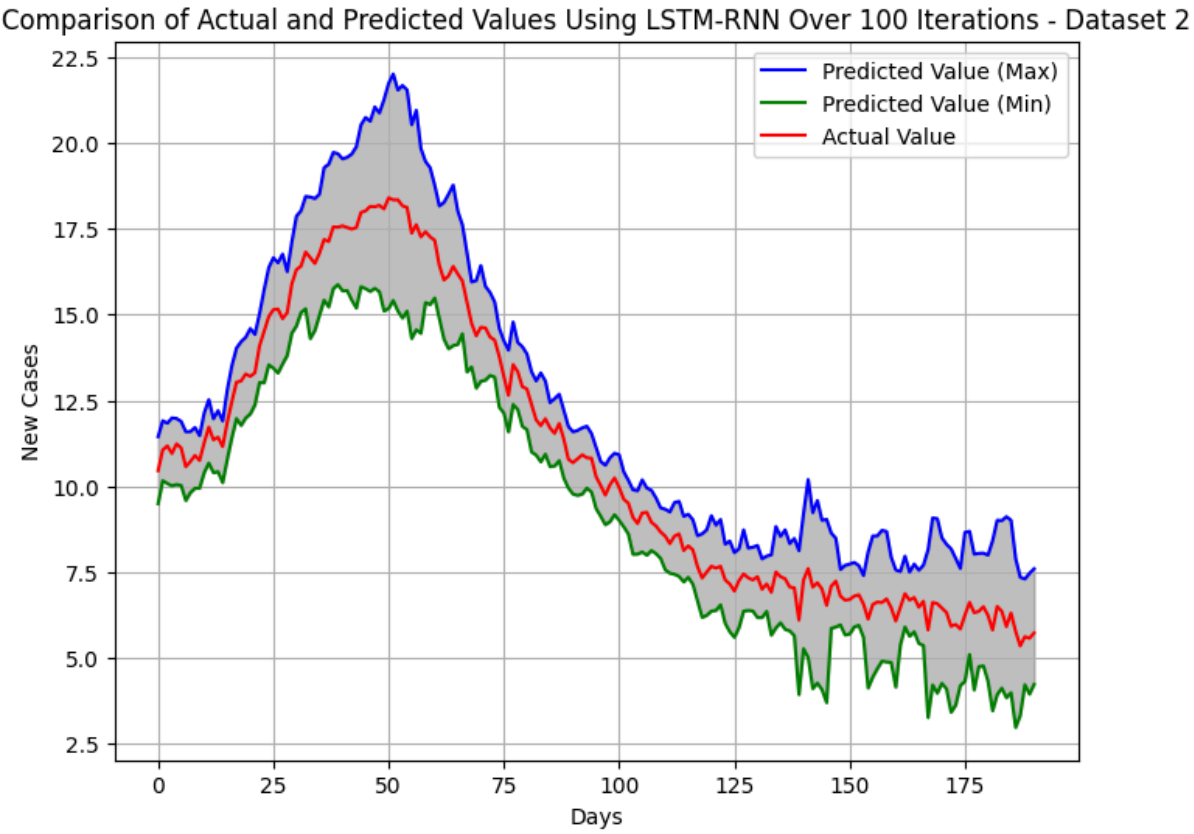
RNN-GRU

In [34]:
```
temp = pd.read_excel('RNN_GRU_PREDICT.xlsx')
shade_plot(temp, line='RNN-GRU', predict='YES')
```



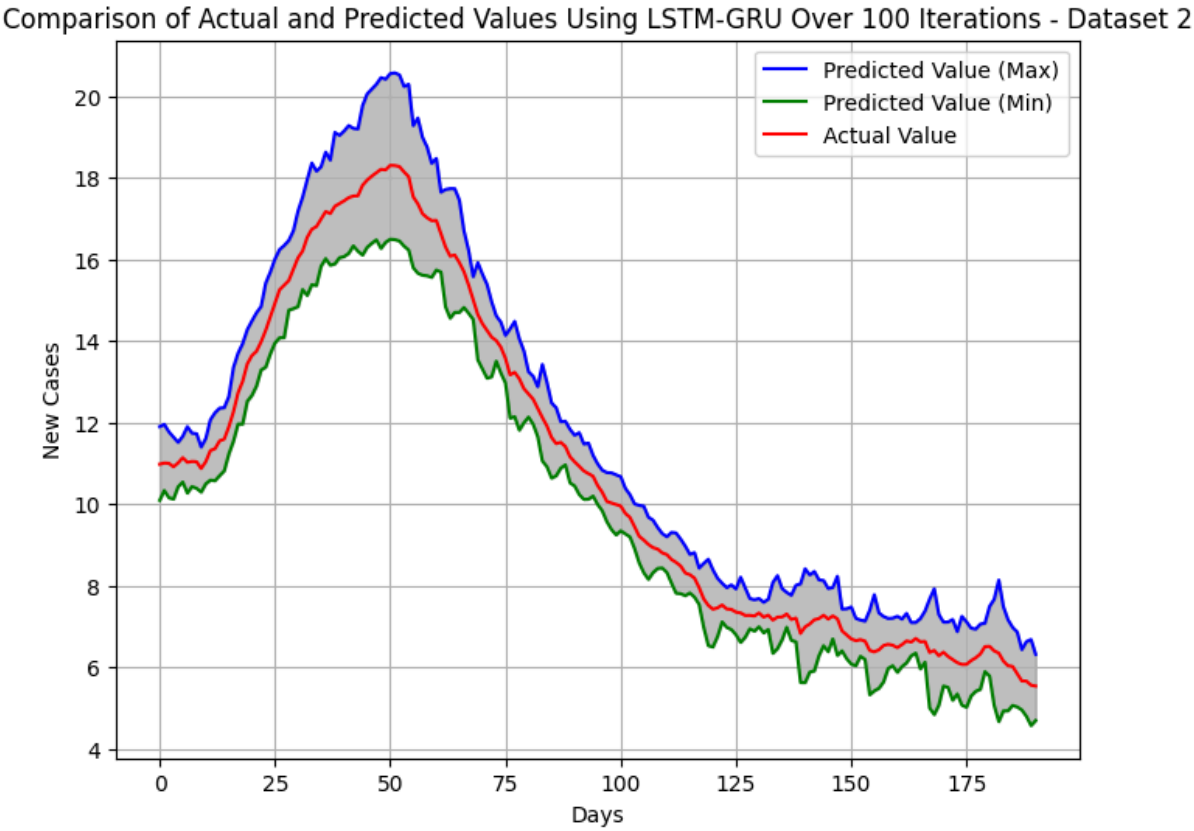
LSTM-RNN

In [39]:
```
temp = pd.read_excel('LSTM_RNN_PREDICT.xlsx')
shade_plot(temp, line='LSTM-RNN', predict='YES')
```



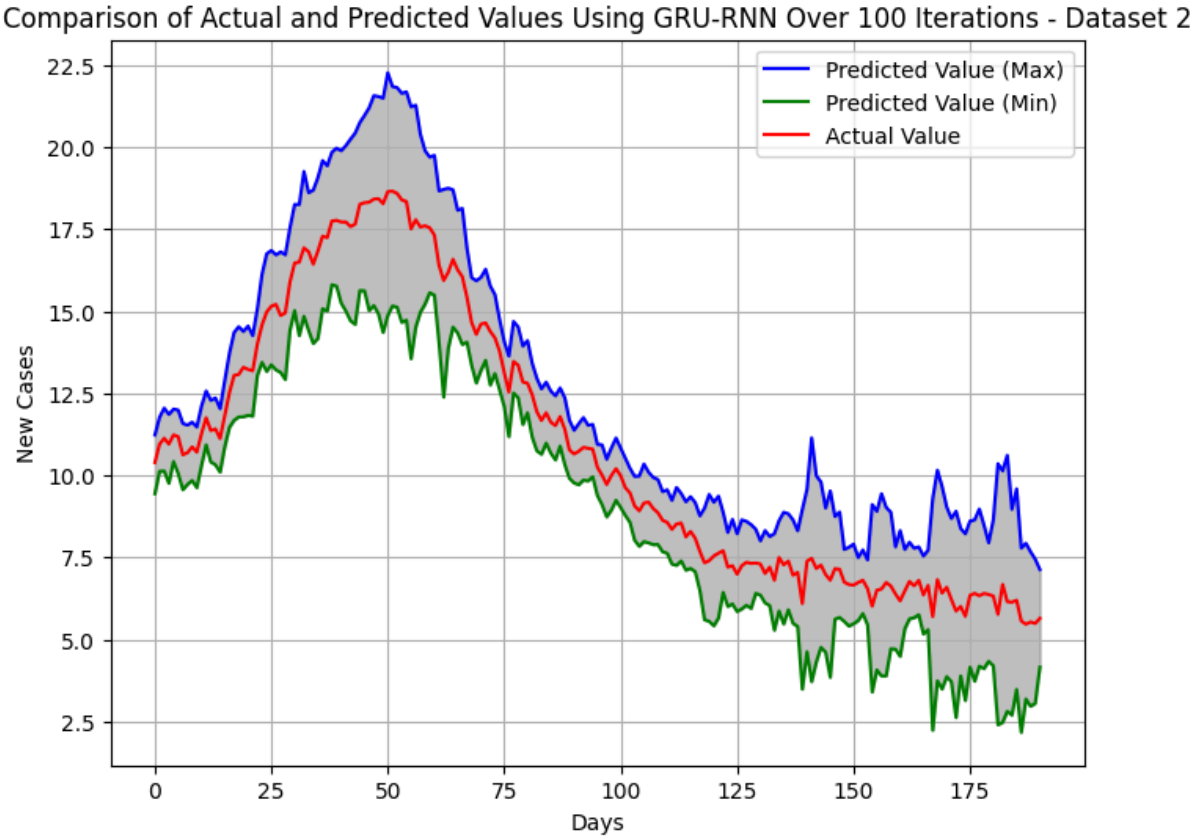
LSTM-GRU

In [44]:
```
temp = pd.read_excel('LSTM_GRU_PREDICT.xlsx')
shade_plot(temp, line='LSTM-GRU', predict='YES')
```



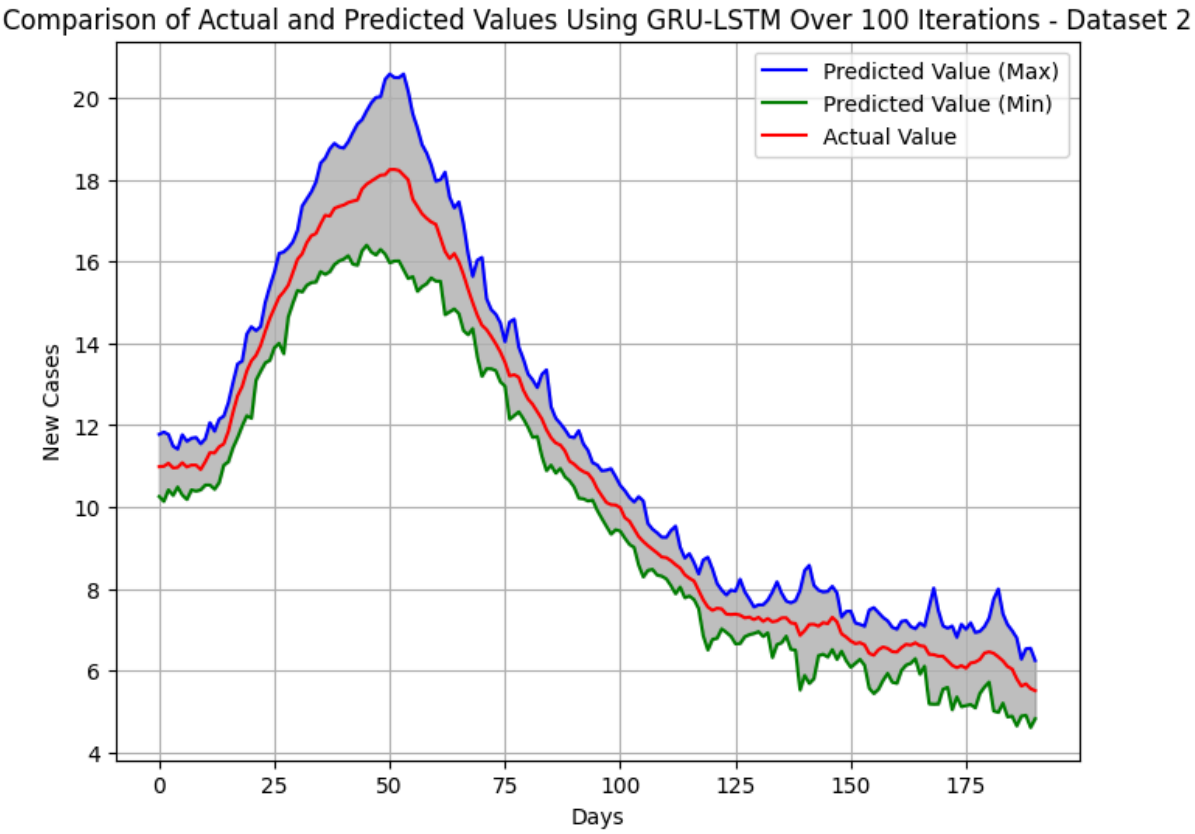
GRU-RNN

In [49]:
```
temp = pd.read_excel('GRU_RNN_PREDICT.xlsx')
shade_plot(temp, line='GRU-RNN', predict='YES')
```



GRU-LSTM

In [54]:
```
temp = pd.read_excel('GRU_LSTM_PREDICT.xlsx')
shade_plot(temp, line='GRU-LSTM', predict='YES')
```



DONWLOAD DATA

In [ ]:
```
!gdown 1_Xk_aBt72cg0IhtrDHByatBuGuLi-svr
!gdown 1ZrtG6EfZxXeV19WMzla_qniBb4E0vHGp
!gdown 1YCs0hRgt-wpSVI7EbtLit5HvjM7brmtB
!gdown 194--ZfpP3lr5eg8jfjbDLSWDSvWVVx18
!gdown 1mrYem8IETC1Bvv4XLfkuFc_Yxs0L-1M5
!gdown 11ElHC2LM013wujQwqW12SD6gDrv60mOl
!gdown 1f62wFr9ML2AAO39aeU2dbKsmh3idLyOi
!gdown 1EImqlbcinVzjSxp5wM27YfCo9Nf_YBn0
```

In [ ]:
```
!gdown 1NyWC6xVK2tlvtjcx0xv8I4O_bI0KebdS
!gdown 1UFgEoC6-313M_dpjq7mW9LICaUvbkoNa
!gdown 1vuERyAz3ckNgg_vAAcjznPa7jpNJN9Rt
!gdown 1ZumVvaJpYdkWhSKZNlqbrHzi--A3iwtx
!gdown 1z0qcyj5D7XoZ4bTMDZ3yaOuHaFaqmXSZ
!gdown 16dF5YhHT4R_g5HbdAsO_cEXejJQ87Pg2
!gdown 1DLTaT7OQxd_R71nSjjj7r7NSJfkG0X8X
!gdown 1A1ARznJ4X2xQM5AL05yfhZo6KIQ21rxZ
!gdown 1fvzFbPqCzqdoLujlHxqIqNlnuCRPddTV
!gdown 1avI3Q59EZV92abBlLrSjNYG05ndLcOJ9
!gdown 16ZR-PaPNwqSyzFj-0SS8AqB7xIJ4OrYe
!gdown 1SgjgWPVq-QTgva1OFAZNKrVzQr0aJwRr
!gdown 1b1zT-srRQd1PEmXi2MyhojUFTGhPTHBM
!gdown 1zsmjBebdjFhh-tm8V4zPF9rQDKs76Vyv
!gdown 1raNErRqrAcr5qFCXH2T1G4bqhbu6l-m5
!gdown 1MHL7n1sIjWsouMWbYfgvDLud-Ba6wWM1
!gdown 1FblK2jeAC9kkZsRp9I6o-64Uqtpql-XI
!gdown 1n718TN1i5kwMEIm0Q6NTcz94D34YXgpA
!gdown 1oy-KECRHSE07R7-78D1tBdiJNbEoxI8E
!gdown 19--phk8Mf1z4QX464ZV_wVL4hfyQmSW1
!gdown 1-JzfQA2dQJe-mXrEK7fdNY1HluF_wSHQ
!gdown 16NiUOKZgspf09ztmREbffpmBKV9zBRhX
!gdown 1mb0yBP7A5seQN2naL8Sgqy54pZ5lYZAZ
!gdown 1CUxws0PZc8T_LWIKyifATBfOzFGMXBi9
!gdown 1ajESJp2rNjlmdWa31-2iIvE8lyvkGKgS
!gdown 14M9gs549r7PLS2CAjEbLoQOiaRfey6X8
!gdown 1j7YuYNeVF4Jt3F3tqG1oH4qgpGwF99jC
!gdown 1Z61vMm0J5Vjn2sNQq87FgaLbAAt8P4or
```

Metrik Evaluation

MAE

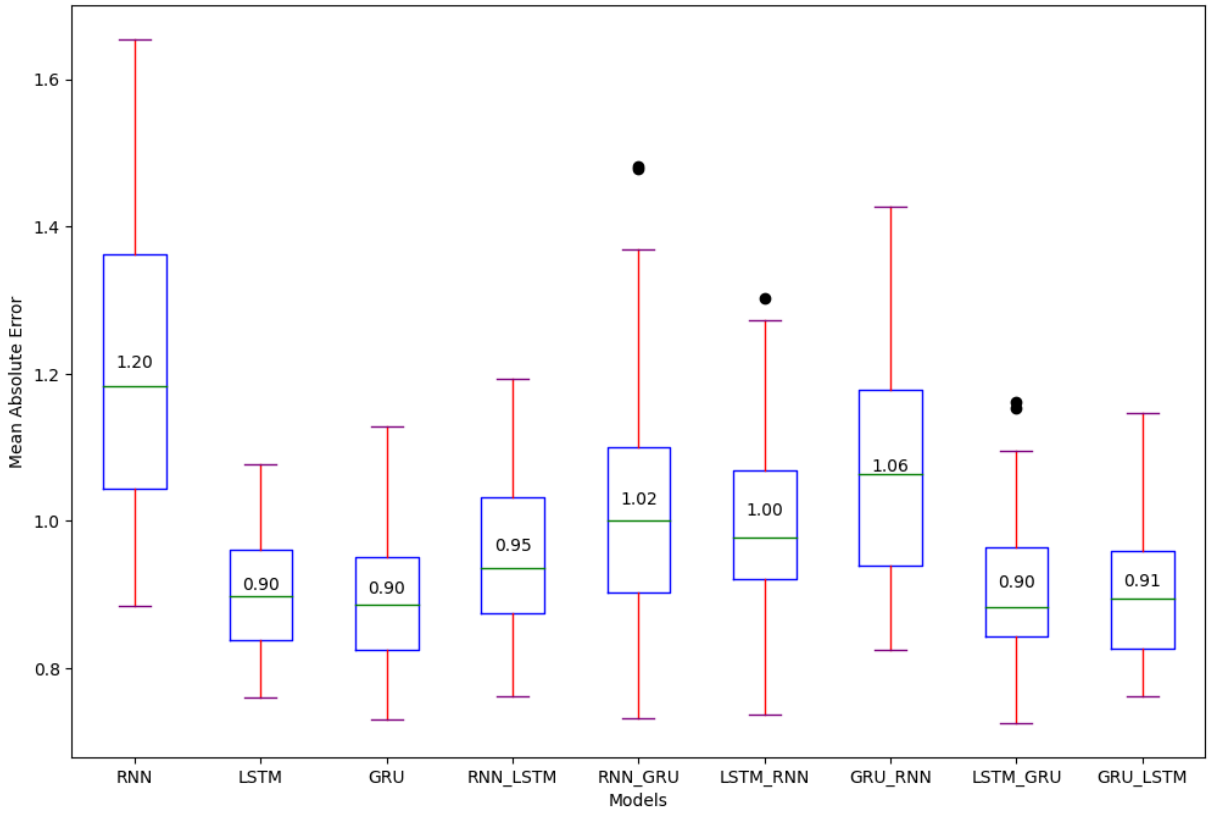
In [ ]:
```
df1 =  pd.read_excel('/content/RNN_FORECASTMae.xlsx')
df2 =  pd.read_excel('/content/LSTM_FORECASTMae.xlsx')
df3 =  pd.read_excel('/content/GRU_FORECASTMae.xlsx')
df4 =  pd.read_excel('/content/RNN_LSTM_FORECASTMae.xlsx')
df5 =  pd.read_excel('/content/RNN_GRU_FORECASTMae.xlsx')
df6 =  pd.read_excel('/content/LSTM_RNN_FORECASTMae.xlsx')
df7 =  pd.read_excel('/content/GRU_RNN_FORECASTMae.xlsx')
df8 =  pd.read_excel('/content/LSTM_GRU_FORECASTMae.xlsx')
df9 =  pd.read_excel('/content/GRU_LSTM_FORECASTMae.xlsx')
```

In [ ]:
```
df = pd.DataFrame()
df['RNN'] =  df1[0]
df['LSTM'] =  df2[0]
df['GRU'] =  df3[0]
df['RNN_LSTM'] =  df4[0]
df['RNN_GRU'] =  df5[0]
df['LSTM_RNN'] =  df6[0]
df['GRU_RNN'] =  df7[0]
df['LSTM_GRU'] =  df8[0]
df['GRU_LSTM'] =  df9[0]
```

```
# n = int(len(df) * (20 / 100))
n = int(len(df) * (2.5 / 100))
df = df.iloc[n:-n]
```

In [ ]:

```
df = pd.DataFrame()
df['RNN'] =  sorted(df1[0])
df['LSTM'] =  sorted(df2[0])
df['GRU'] =  sorted(df3[0])
df['RNN_LSTM'] =  sorted(df4[0])
df['RNN_GRU'] =  sorted(df5[0])
df['LSTM_RNN'] = sorted(df6[0])
df['GRU_RNN'] =  sorted(df7[0])
df['LSTM_GRU'] = sorted(df8[0])
df['GRU_LSTM'] = sorted(df9[0])


# n = int(len(df)*0.15)
n = int(len(df) * 0.025)
df = df[:].iloc[n:-n]
data = df[:]

box_color = 'blue'
whisker_color = 'red'
median_color = 'green'
flier_color = 'black'
cap_color = 'purple'

fig, ax = plt.subplots(figsize=(12, 8))

plt.boxplot(data.values, labels=data.columns,
            boxprops=dict(color=box_color),
            whiskerprops=dict(color=whisker_color),
            capprops=dict(color=cap_color),
            flierprops=dict(markerfacecolor=flier_color, marker='o', markersize=6),
            medianprops=dict(color=median_color))

for i, column in enumerate(data.columns):
    mean = np.mean(data[column])
    plt.text(i + 1, mean, f'{mean:.2f}', ha='center', va='bottom', color='black')

# plt.title('Predict with Boxplot')
plt.xlabel('Models')
plt.ylabel('Mean Absolute Error')
plt.savefig("Comparison_MAE_Dataset2.pdf", dpi=300)
files.download("Comparison_MAE_Dataset2.pdf")

plt.show()
```

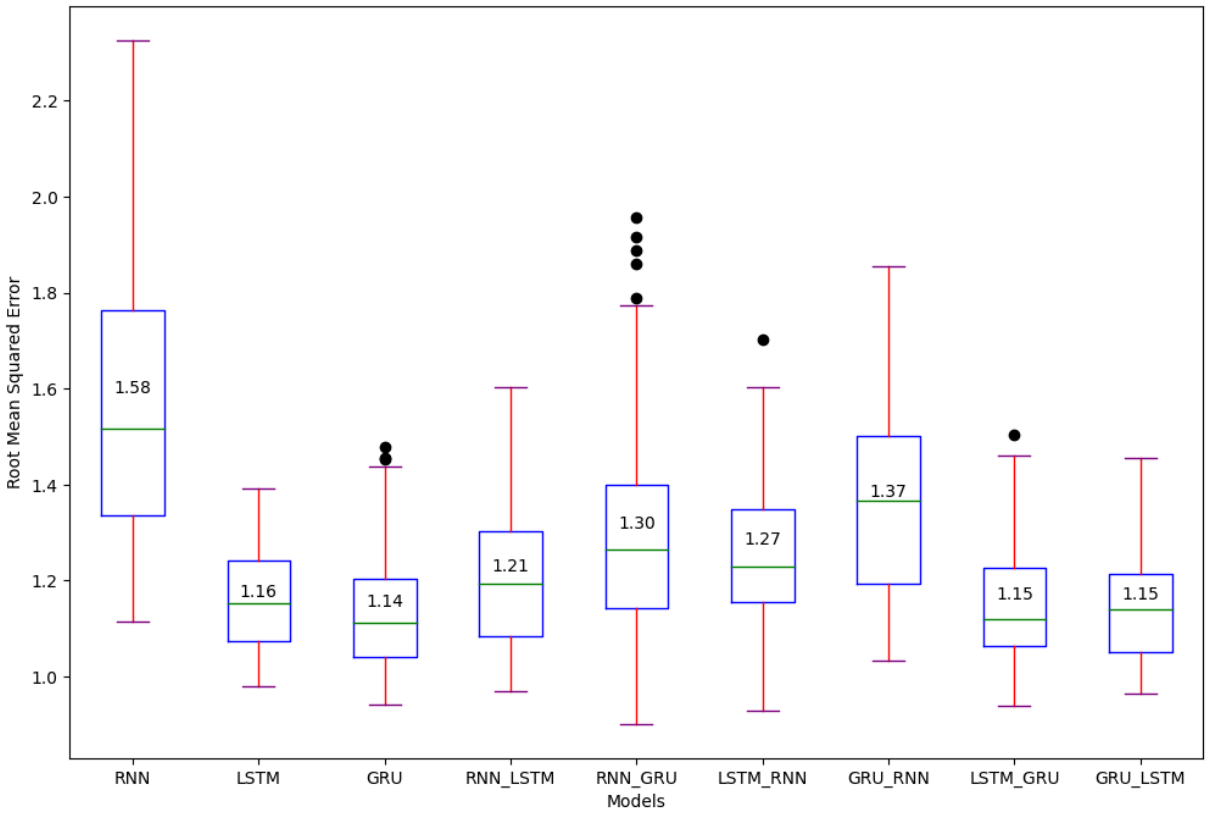


RMSE

In [ ]:
```
df1 =  pd.read_excel('/content/RNN_FORECASTRMse.xlsx')
df2 =  pd.read_excel('/content/LSTM_FORECASTRMse.xlsx')
df3 =  pd.read_excel('/content/GRU_FORECASTRMse.xlsx')
df4 =  pd.read_excel('/content/RNN-LSTM_FORECASTRMse.xlsx')
df5 =  pd.read_excel('/content/RNN-GRU_FORECASTRMse.xlsx')
df6 =  pd.read_excel('/content/LSTM-RNN_FORECASTRMse.xlsx')
df7 =  pd.read_excel('/content/GRU-RNN_FORECASTRMse.xlsx')
df8 =  pd.read_excel('/content/LSTM-GRU_FORECASTRMse.xlsx')
df9 =  pd.read_excel('/content/GRU-LSTM_FORECASTRMse.xlsx')
df = pd.DataFrame()
df['RNN'] =  df1['RMSE']
df['LSTM'] =  df2['RMSE']
df['GRU'] =  df3['RMSE']
df['RNN_LSTM'] =  df4['RMSE']
df['RNN_GRU'] =  df5['RMSE']
df['LSTM_RNN'] =  df6['RMSE']
df['GRU_RNN'] =  df7['RMSE']
df['LSTM_GRU'] =  df8['RMSE']
df['GRU_LSTM'] =  df9['RMSE']

# n = int(len(df) * (20 / 100))
n = int(len(df) * (2.5 / 100))
df = df.iloc[n:-n]
```

In [ ]:
```
df = pd.DataFrame()
df['RNN'] =  sorted(df1['RMSE'])
df['LSTM'] =  sorted(df2['RMSE'])
df['GRU'] =  sorted(df3['RMSE'])
df['RNN_LSTM'] =  sorted(df4['RMSE'])
df['RNN_GRU'] =  sorted(df5['RMSE'])
df['LSTM_RNN'] = sorted(df6['RMSE'])
df['GRU_RNN'] =  sorted(df7['RMSE'])
df['LSTM_GRU'] = sorted(df8['RMSE'])
df['GRU_LSTM'] = sorted(df9['RMSE'])


# n = int(len(df)*0.15)
n = int(len(df) * 0.025)
```

```
df = df[:].iloc[n:-n]
data = df[:]

box_color = 'blue'
whisker_color = 'red'
median_color = 'green'
flier_color = 'black'
cap_color = 'purple'

fig, ax = plt.subplots(figsize=(12, 8))

plt.boxplot(data.values, labels=data.columns,
            boxprops=dict(color=box_color),
            whiskerprops=dict(color=whisker_color),
            capprops=dict(color=cap_color),
            flierprops=dict(markerfacecolor=flier_color, marker='o', markersize=6),
            medianprops=dict(color=median_color))

for i, column in enumerate(data.columns):
    mean = np.mean(data[column])
    plt.text(i + 1, mean, f'{mean:.2f}', ha='center', va='bottom', color='black')

# plt.title('Predict with Boxplot')
plt.xlabel('Models')
plt.ylabel('Root Mean Squared Error')
plt.savefig("Comparison_RMSE_Dataset2.pdf", dpi=300)
files.download("Comparison_RMSE_Dataset2.pdf")

plt.show()
```



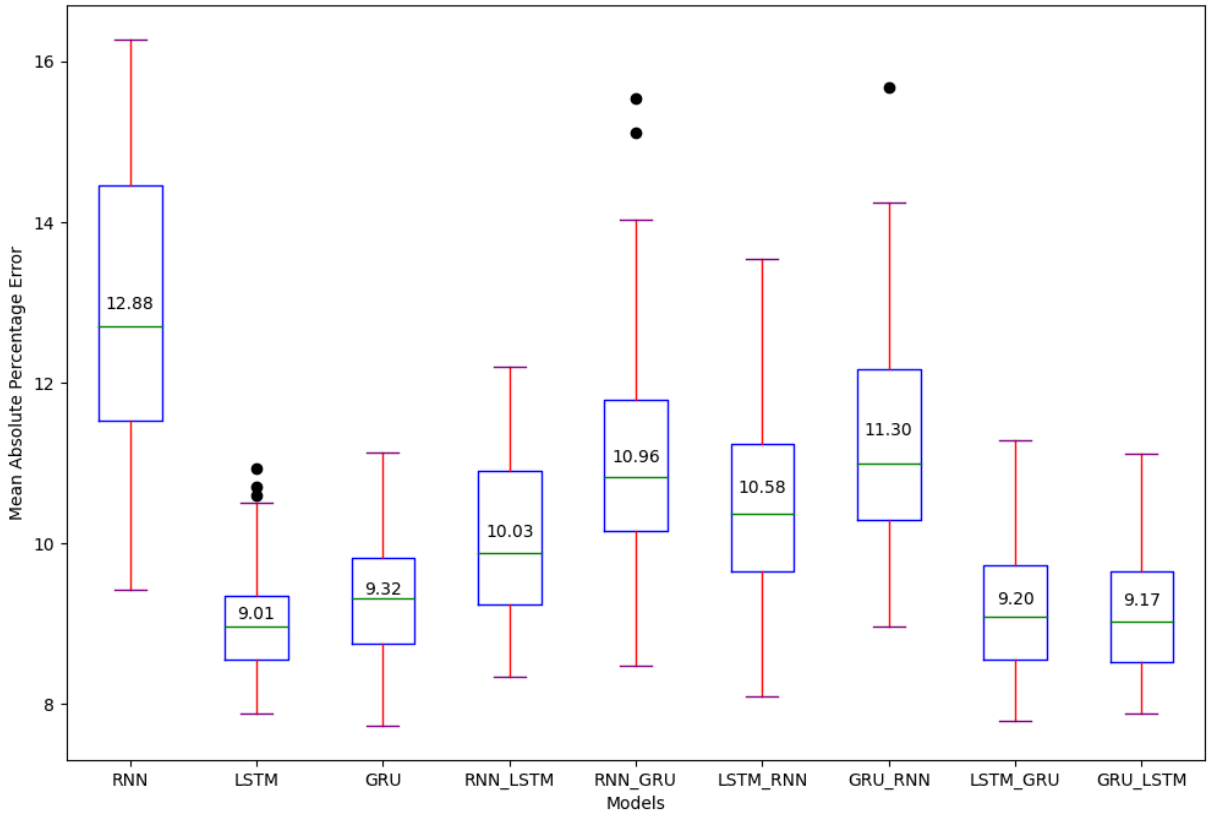
MAPE

In [ ]:
```
df1 =  pd.read_excel('/content/RNN_FORECASTMpe.xlsx')
df2 =  pd.read_excel('/content/LSTM_FORECASTMpe.xlsx')
df3 =  pd.read_excel('/content/GRU_FORECASTMpe.xlsx')
df4 =  pd.read_excel('/content/RNN-LSTM_FORECASTMpe.xlsx')
df5 =  pd.read_excel('/content/RNN-GRU_FORECASTMpe.xlsx')
df6 =  pd.read_excel('/content/LSTM-RNN_FORECASTMpe.xlsx')
df7 =  pd.read_excel('/content/GRU-RNN_FORECASTMpe.xlsx')
```

```
df8 =  pd.read_excel('/content/LSTM-GRU_FORECASTMpe.xlsx')
df9 =  pd.read_excel('/content/GRU-LSTM_FORECASTMpe.xlsx')
df = pd.DataFrame()
df['RNN'] =  df1['MAPE']
df['LSTM'] =  df2['MAPE']
df['GRU'] =  df3['MAPE']
df['RNN_LSTM'] =  df4['MAPE']
df['RNN_GRU'] =  df5['MAPE']
df['LSTM_RNN'] =  df6['MAPE']
df['GRU_RNN'] =  df7['MAPE']
df['LSTM_GRU'] =  df8['MAPE']
df['GRU_LSTM'] =  df9['MAPE']

# n = int(len(df) * (20 / 100))
n = int(len(df) * (2.5 / 100))
df = df.iloc[n:-n]
```

In [ ]:

```
df = pd.DataFrame()
df['RNN'] =  sorted(df1['MAPE'])
df['LSTM'] =  sorted(df2['MAPE'])
df['GRU'] =  sorted(df3['MAPE'])
df['RNN_LSTM'] =  sorted(df4['MAPE'])
df['RNN_GRU'] =  sorted(df5['MAPE'])
df['LSTM_RNN'] = sorted(df6['MAPE'])
df['GRU_RNN'] =  sorted(df7['MAPE'])
df['LSTM_GRU'] = sorted(df8['MAPE'])
df['GRU_LSTM'] = sorted(df9['MAPE'])


# n = int(len(df)*0.15)
n = int(len(df) * 0.025)
df = df[:].iloc[n:-n]
data = df[:]

box_color = 'blue'
whisker_color = 'red'
median_color = 'green'
flier_color = 'black'
cap_color = 'purple'

fig, ax = plt.subplots(figsize=(12, 8))

plt.boxplot(data.values, labels=data.columns,
            boxprops=dict(color=box_color),
            whiskerprops=dict(color=whisker_color),
            capprops=dict(color=cap_color),
            flierprops=dict(markerfacecolor=flier_color, marker='o', markersize=6),
            medianprops=dict(color=median_color))

for i, column in enumerate(data.columns):
    mean = np.mean(data[column])
    plt.text(i + 1, mean, f'{mean:.2f}', ha='center', va='bottom', color='black')

# plt.title('Predict with Boxplot')
plt.xlabel('Models')
plt.ylabel('Mean Absolute Percentage Error')
plt.savefig("Comparison_MAPE_Dataset2.pdf", dpi=300)
files.download("Comparison_MAPE_Dataset2.pdf")

plt.show()
```

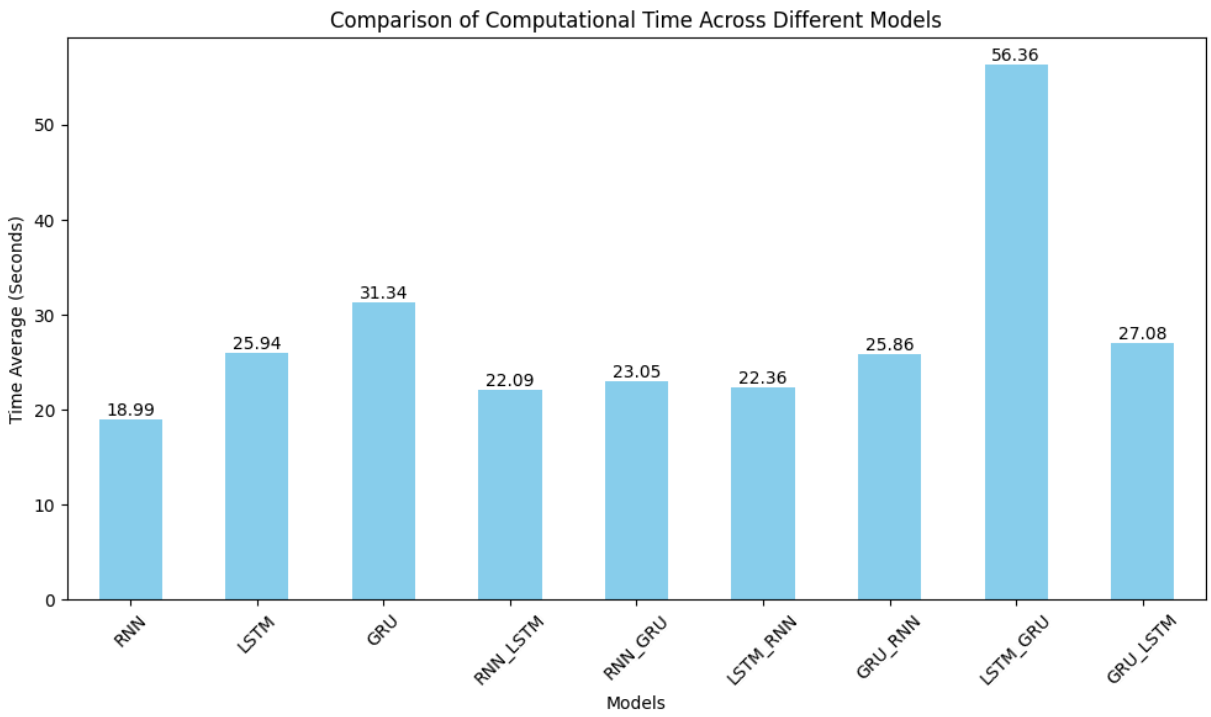


Time Comparison

In [ ]:
```
df1 =  pd.read_excel('/content/RNN_TIME.xlsx')
df2 =  pd.read_excel('/content/LSTM_TIME.xlsx')
df3 =  pd.read_excel('/content/GRU_TIME.xlsx')
df4 =  pd.read_excel('/content/RNN-LSTM_TIME.xlsx')
df5 =  pd.read_excel('/content/RNN-GRU_TIME.xlsx')
df6 =  pd.read_excel('/content/LSTM-RNN_TIME.xlsx')
df7 =  pd.read_excel('/content/GRU-RNN_TIME.xlsx')
df8 =  pd.read_excel('/content/LSTM-GRU_TIME.xlsx')
df9 =  pd.read_excel('/content/GRU-LSTM_TIME.xlsx')
df = pd.DataFrame()
df['RNN'] =  df1[0]
df['LSTM'] =  df2[0]
df['GRU'] =  df3[0]
df['RNN_LSTM'] =  df4[0]
df['RNN_GRU'] =  df5[0]
df['LSTM_RNN'] =  df6[0]
df['GRU_RNN'] =  df7[0]
df['LSTM_GRU'] =  df8[0]
df['GRU_LSTM'] =  df9[0]
```

In [ ]:
```
# Calculate the average time for each model
average_times = df.mean()

# Create a bar chart
plt.figure(figsize=(10, 6))
average_times.plot(kind='bar', color='skyblue')
plt.title('Comparison of Computational Time Across Different Models')
plt.xlabel('Models')
plt.ylabel('Time Average (Seconds)')
plt.xticks(rotation=45)
plt.tight_layout()

# Display the average times on top of the bars
for i, v in enumerate(average_times):
    plt.text(i, v, f"{v:.2f}", ha='center', va='bottom')

plt.savefig("Comparison_Time_Dataset2.pdf", dpi=300)
```

```
files.download("Comparison_Time_Dataset2.pdf")
plt.show()
```



1. Import Libraries

List of libraries that will be used.

In [1]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import csv
```

Using Utilites of Data Visualization

In [2]:
```
def plot_series(x, y, format="-", start=0, end=None,
                title=None, xlabel=None, ylabel=None, legend=None ):
    plt.figure(figsize=(10, 6))
    if type(y) is tuple:
      for y_curr in y:
        plt.plot(x[start:end], y_curr[start:end], format)
    else:
      plt.plot(x[start:end], y[start:end], format)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    if legend:
      plt.legend(legend)
    plt.title(title)
    plt.grid(True)
    plt.show()
```

2. Get DATA

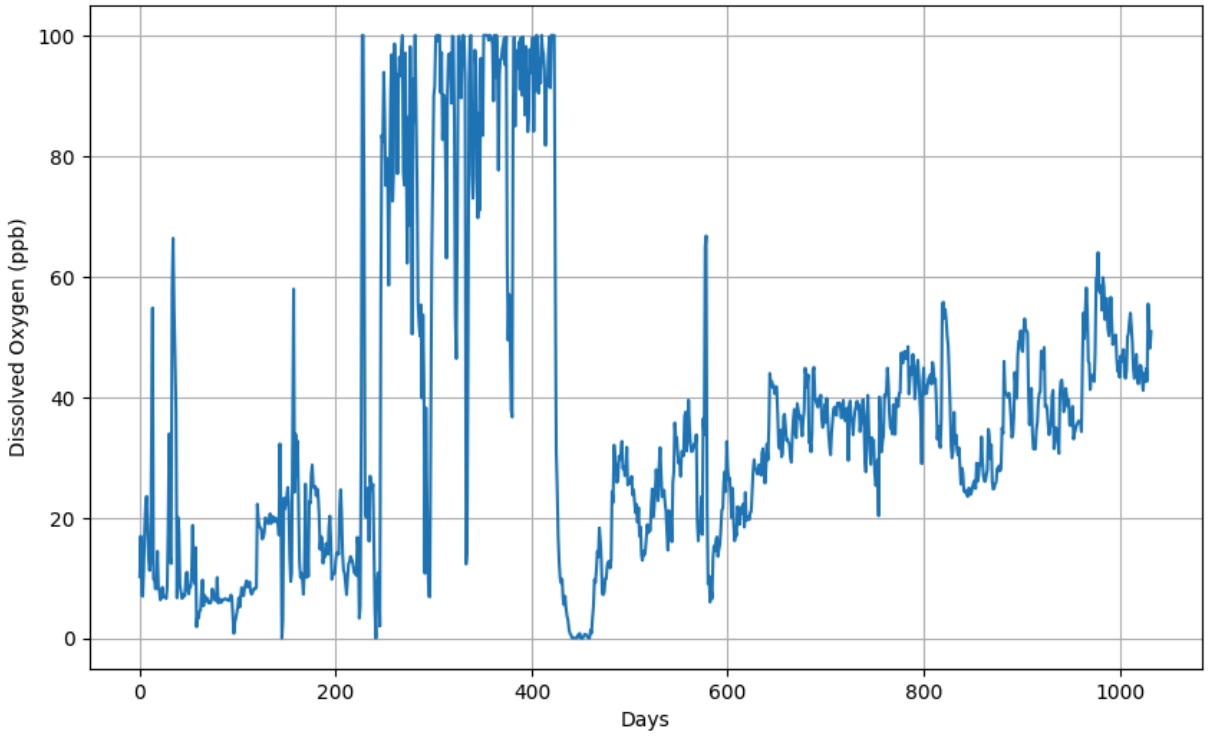
Take Data from Upstream (Local, Gdrive, Github, etc).

In [ ]:
```
!gdown 1tFt9l06v8hl2dRXRx-yMXMv6QRlwkF2p
```

Data Visualize

In [4]:
```
file = pd.read_excel('DissolvedOxygen.xlsx')
def take(file):
  time_step = []
  sunspots = []

  for i in range(len(file)):
    time_step.append(i)
    sunspots.append(file.iloc[i][1])
  time = np.array(time_step)
  series = np.array(sunspots)
  return time, series
time, series = take(file)
plot_series(time, series, xlabel='Days', ylabel='Dissolved Oxygen (ppb)')
```

```
<ipython-input-4-52403bb85ecb>:8: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  sunspots.append(file.iloc[i][1])
```



3. Data Prerocessing

Analysis and Cleaning Data -> Extract, Transform, Load.

Split Data

In [5]:
```
split_time = int(len(time)*0.8)

time_train = time[:split_time]
x_train = series[:split_time]

time_valid = time[split_time:]
x_valid = series[split_time:]
```

Prepare Features and Labels

In [6]:
```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
    dataset = dataset.map(lambda window: (window[:-1], window[-1]))
    dataset = dataset.shuffle(shuffle_buffer)
    dataset = dataset.batch(batch_size).prefetch(1)
    return dataset

window_size = 30
batch_size = 32
shuffle_buffer_size = 1000

train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)
```

4. Model Architechture

Using library from Scikit-learn, Tensorflow(?).

RNN Model

In [7]:
```
model_RNN = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                        activation="relu",
                        padding='causal',
                        input_shape=[window_size, 1]),
  tf.keras.layers.SimpleRNN(64, return_sequences=True),
  tf.keras.layers.SimpleRNN(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_RNN.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: 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__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 30, 64)	256
simple_rnn (SimpleRNN)	(None, 30, 64)	8,256
simple_rnn_1 (SimpleRNN)	(None, 64)	8,256
dense (Dense)	(None, 1)	65
lambda (Lambda)	(None, 1)	0

Total params: 16,833 (65.75 KB)

Trainable params: 16,833 (65.75 KB)

Non-trainable params: 0 (0.00 B)

LSTM Model

In [8]:
```
model_LSTM = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                        activation="relu",
                        padding='causal',
                        input_shape=[window_size, 1]),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.LSTM(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_LSTM.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 30, 64)	256
lstm (LSTM)	(None, 30, 64)	33,024
lstm_1 (LSTM)	(None, 64)	33,024
dense_1 (Dense)	(None, 1)	65
lambda_1 (Lambda)	(None, 1)	0

Total params: 66,369 (259.25 KB)

Trainable params: 66,369 (259.25 KB)

GRU Model

In [9]:
```
model_GRU = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                        activation="relu",
                        padding='causal',
                        input_shape=[window_size, 1]),
  tf.keras.layers.GRU(64, return_sequences=True),
  tf.keras.layers.GRU(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_GRU.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv1d_2 (Conv1D)	(None, 30, 64)	256
gru (GRU)	(None, 30, 64)	24,960
gru_1 (GRU)	(None, 64)	24,960
dense_2 (Dense)	(None, 1)	65
lambda_2 (Lambda)	(None, 1)	0

Total params: 50,241 (196.25 KB)

Trainable params: 50,241 (196.25 KB)

Non-trainable params: 0 (0.00 B)

RNN-LSTM Model

In [10]:
```
model_RNN_LSTM = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                         activation="relu",
                         padding='causal',
                         input_shape=[window_size, 1]),
  tf.keras.layers.SimpleRNN(64, return_sequences=True),
  tf.keras.layers.LSTM(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_RNN_LSTM.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv1d_3 (Conv1D)	(None, 30, 64)	256
simple_rnn_2 (SimpleRNN)	(None, 30, 64)	8,256
lstm_2 (LSTM)	(None, 64)	33,024
dense_3 (Dense)	(None, 1)	65
lambda_3 (Lambda)	(None, 1)	0

Total params: 41,601 (162.50 KB)

Trainable params: 41,601 (162.50 KB)

RNN-GRU Model

In [11]:
```
model_RNN_GRU = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                         activation="relu",
                         padding='causal',
                         input_shape=[window_size, 1]),
  tf.keras.layers.SimpleRNN(64, return_sequences=True),
  tf.keras.layers.GRU(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_RNN_GRU.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv1d_4 (Conv1D)	(None, 30, 64)	256
simple_rnn_3 (SimpleRNN)	(None, 30, 64)	8,256
gru_2 (GRU)	(None, 64)	24,960
dense_4 (Dense)	(None, 1)	65
lambda_4 (Lambda)	(None, 1)	0

Total params: 33,537 (131.00 KB)

Trainable params: 33,537 (131.00 KB)

Non-trainable params: 0 (0.00 B)

LSTM-RNN

In [12]:
```
model_LSTM_RNN = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                         activation="relu",
                         padding='causal',
                         input_shape=[window_size, 1]),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.SimpleRNN(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
```

```
])
model_LSTM_RNN.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
conv1d_5 (Conv1D)	(None, 30, 64)	256
lstm_3 (LSTM)	(None, 30, 64)	33,024
simple_rnn_4 (SimpleRNN)	(None, 64)	8,256
dense_5 (Dense)	(None, 1)	65
lambda_5 (Lambda)	(None, 1)	0

Total params: 41,601 (162.50 KB)

Trainable params: 41,601 (162.50 KB)

Non-trainable params: 0 (0.00 B)

LSTM-GRU Model

In [13]:
```
model_LSTM_GRU = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                         activation="relu",
                         padding='causal',
                         input_shape=[window_size, 1]),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.GRU(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_LSTM_GRU.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
conv1d_6 (Conv1D)	(None, 30, 64)	256
lstm_4 (LSTM)	(None, 30, 64)	33,024
gru_3 (GRU)	(None, 64)	24,960
dense_6 (Dense)	(None, 1)	65
lambda_6 (Lambda)	(None, 1)	0

Total params: 58,305 (227.75 KB)

Trainable params: 58,305 (227.75 KB)

Non-trainable params: 0 (0.00 B)

GRU-RNN Model

In [14]:
```
model_GRU_RNN = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                         activation="relu",
                         padding='causal',
```

```
                      input_shape=[window_size, 1]),
  tf.keras.layers.GRU(64, return_sequences=True),
  tf.keras.layers.SimpleRNN(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_GRU_RNN.summary()
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
conv1d_7 (Conv1D)	(None, 30, 64)	256
gru_4 (GRU)	(None, 30, 64)	24,960
simple_rnn_5 (SimpleRNN)	(None, 64)	8,256
dense_7 (Dense)	(None, 1)	65
lambda_7 (Lambda)	(None, 1)	0

Total params: 33,537 (131.00 KB)

Trainable params: 33,537 (131.00 KB)

Non-trainable params: 0 (0.00 B)

GRU-LSTM Model

In [15]:
```
model_GRU_LSTM = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                         strides=1,
                        activation="relu",
                        padding='causal',
                        input_shape=[window_size, 1]),
  tf.keras.layers.GRU(64, return_sequences=True),
  tf.keras.layers.LSTM(64),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])
model_GRU_LSTM  .summary()
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
conv1d_8 (Conv1D)	(None, 30, 64)	256
gru_5 (GRU)	(None, 30, 64)	24,960
lstm_5 (LSTM)	(None, 64)	33,024
dense_8 (Dense)	(None, 1)	65
lambda_8 (Lambda)	(None, 1)	0

Total params: 58,305 (227.75 KB)

Trainable params: 58,305 (227.75 KB)

Non-trainable params: 0 (0.00 B)

5. Model Training

In [16]:
```
tf.keras.backend.clear_session()
learning_rate = 8e-7
```

In [ ]:
```
optimizer_RNN = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_RNN.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_RNN,
              metrics=["mae", "mse", "mape"])
history_RNN = model_RNN.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_LSTM = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_LSTM.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_LSTM,
              metrics=["mae", "mse", "mape"])
history_LSTM = model_LSTM.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_GRU= tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_GRU.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_GRU,
              metrics=["mae", "mse", "mape"])
history_GRU = model_GRU.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_RNN_LSTM = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_RNN_LSTM.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_RNN_LSTM,
              metrics=["mae", "mse", "mape"])
history_RNN_LSTM = model_RNN_LSTM.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_RNN_GRU = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_RNN_GRU.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_RNN_GRU,
              metrics=["mae", "mse", "mape"])
history_RNN_GRU = model_RNN_GRU.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_LSTM_RNN = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_LSTM_RNN.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_LSTM_RNN,
              metrics=["mae", "mse", "mape"])
history_LSTM_RNN = model_LSTM_RNN.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_LSTM_GRU = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_LSTM_GRU.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_LSTM_GRU,
              metrics=["mae", "mse", "mape"])
history_LSTM_GRU = model_LSTM_GRU.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_GRU_RNN= tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_GRU_RNN.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_GRU_RNN,
              metrics=["mae", "mse", "mape"])
history_GRU_RNN = model_GRU_RNN.fit(train_set,epochs=100)
```

In [ ]:
```
optimizer_GRU_LSTM= tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
model_GRU_LSTM.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer_GRU_LSTM,
              metrics=["mae", "mse", "mape"])
history_GRU_LSTM = model_GRU_LSTM.fit(train_set,epochs=100)
```

6. Model Evaluation

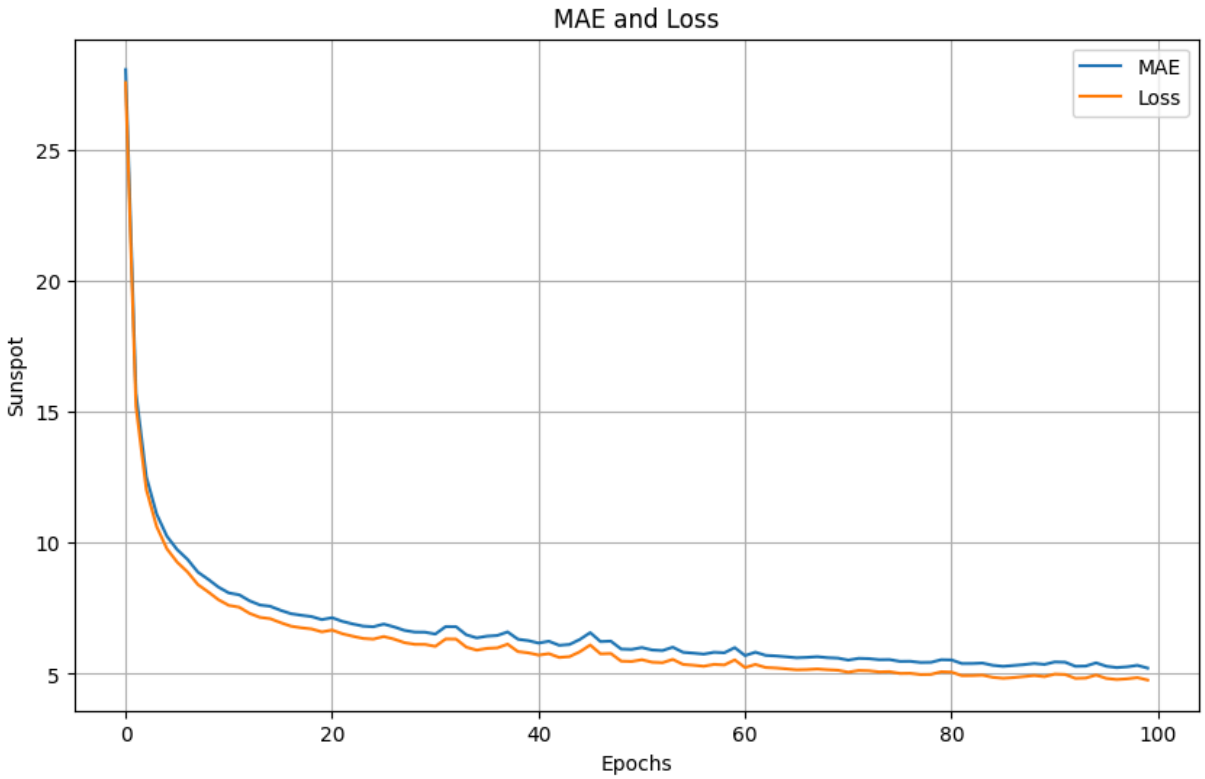
Loss evaluation using Scatter Plot

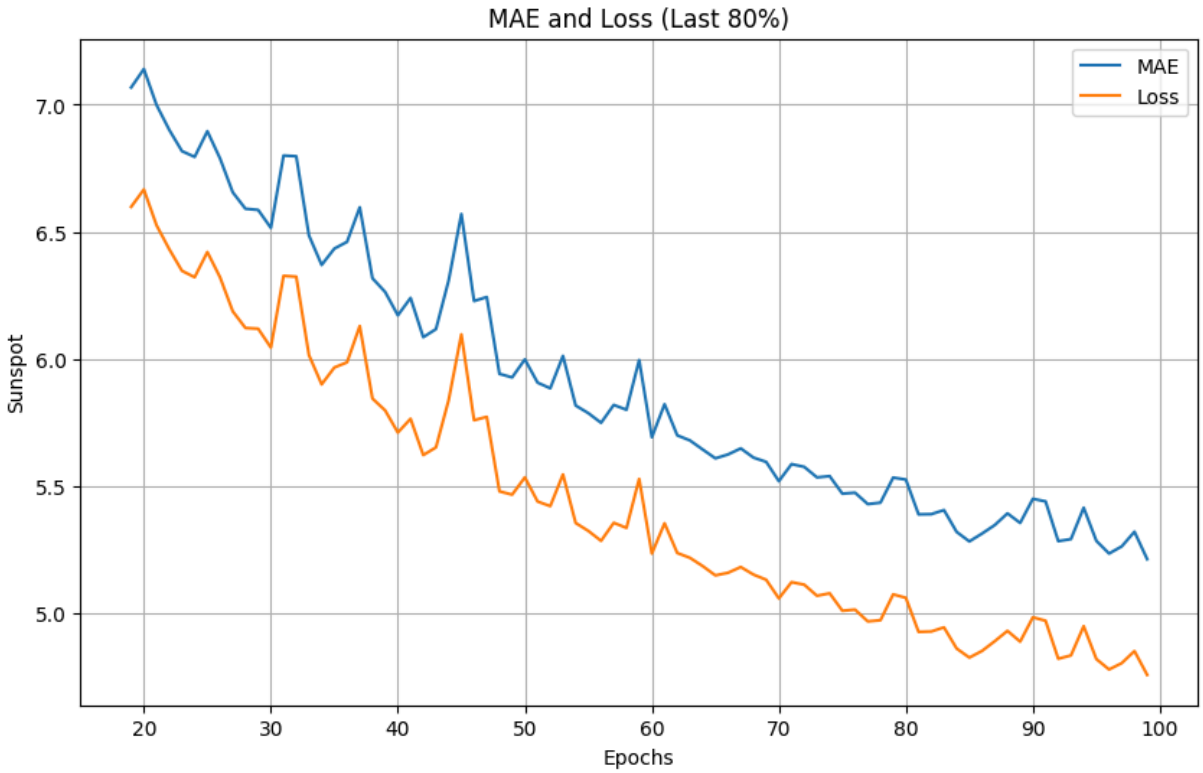
In [26]:
```
# Plot Function
def visualize_evaluation(history):
  mae=history.history['mae']
  loss=history.history['loss']
```

```
epochs=range(len(loss))

plot_series(
    x=epochs,
    y=(mae, loss),
    title='MAE and Loss',
    xlabel='Epochs',
    ylabel='Sunspot',
    legend=['MAE', 'Loss'])

zoom_split = int(epochs[-1] * 0.2)
epochs_zoom = epochs[zoom_split:]
mae_zoom = mae[zoom_split:]
loss_zoom = loss[zoom_split:]

plot_series(
    x=epochs_zoom,
    y=(mae_zoom, loss_zoom),
    title='MAE and Loss (Last 80%)',
    xlabel='Epochs',
    ylabel='Sunspot',
    legend=['MAE', 'Loss'])
```

RNN Evaluation

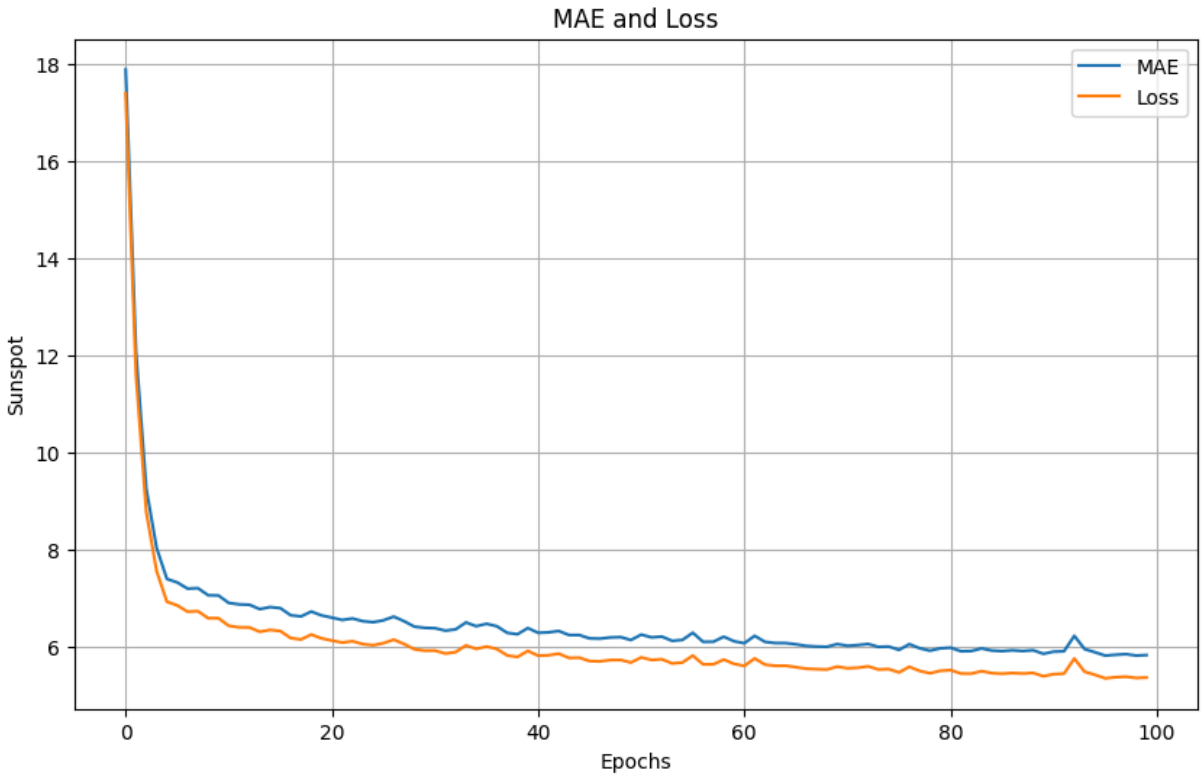
In [27]:
```
visualize_evaluation(history_RNN)
```

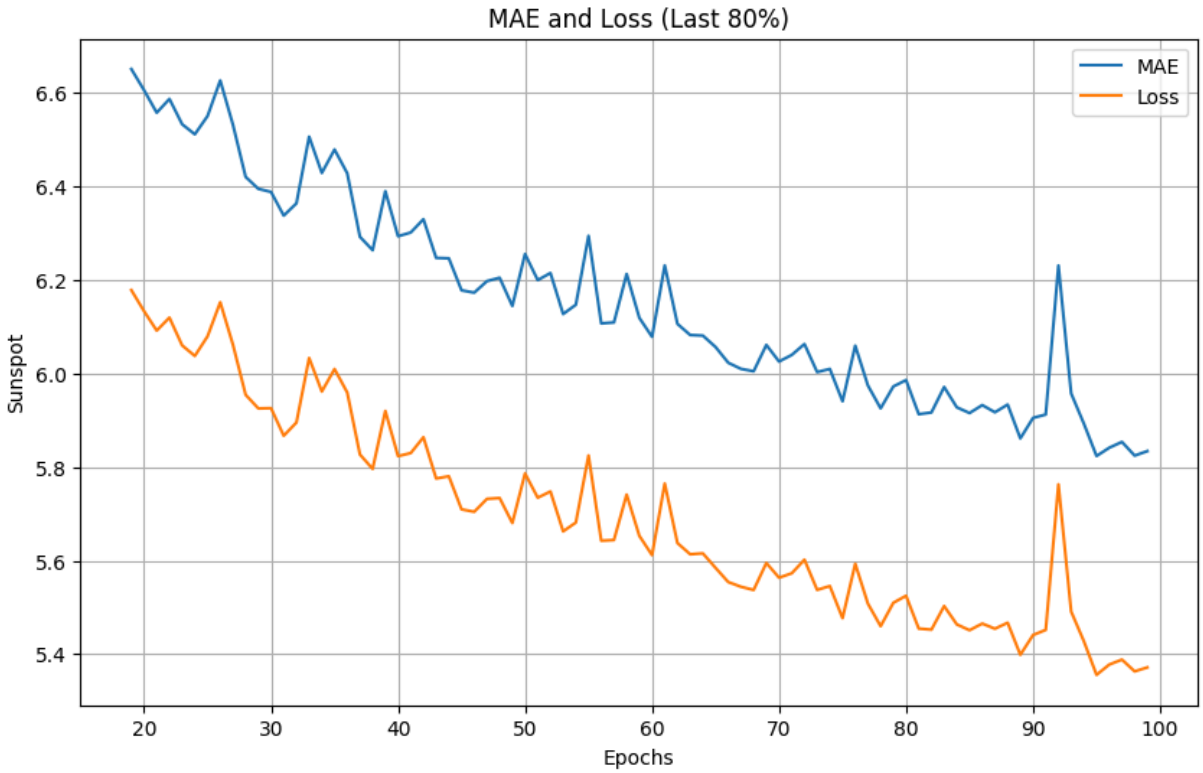




LSTM Evaluation

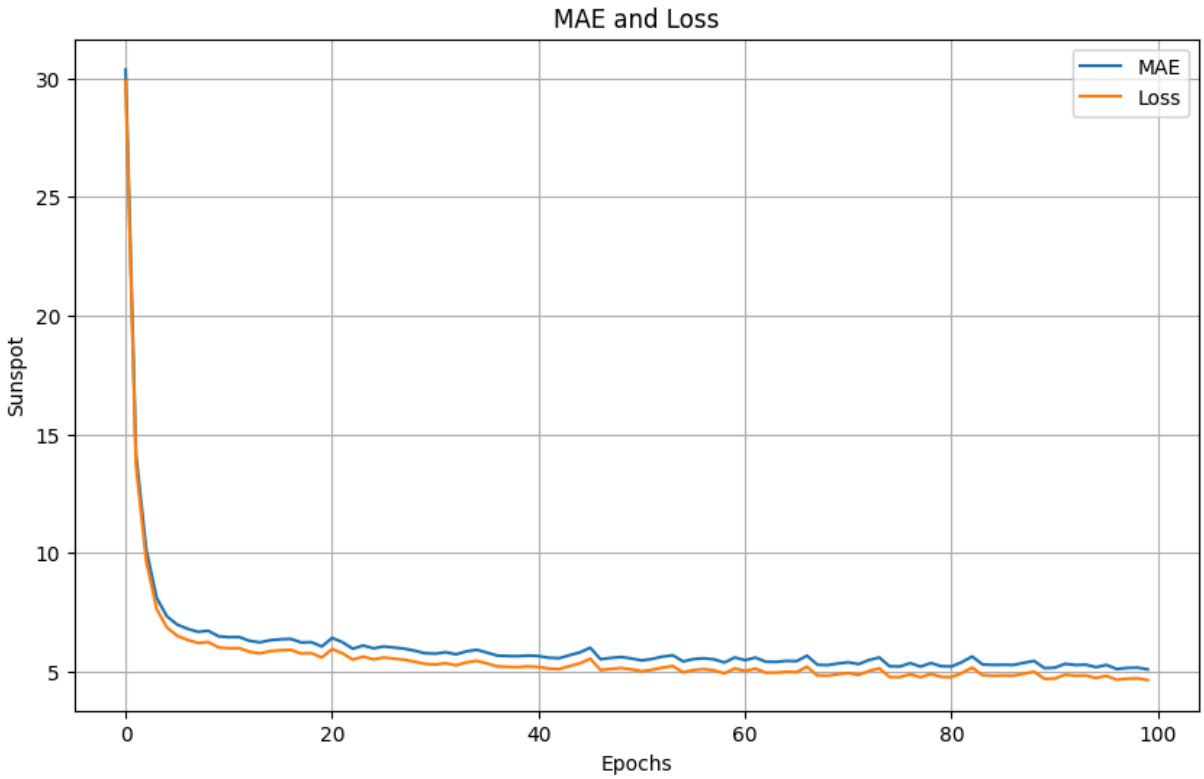
In [28]:
```
visualize_evaluation(history_LSTM)
```

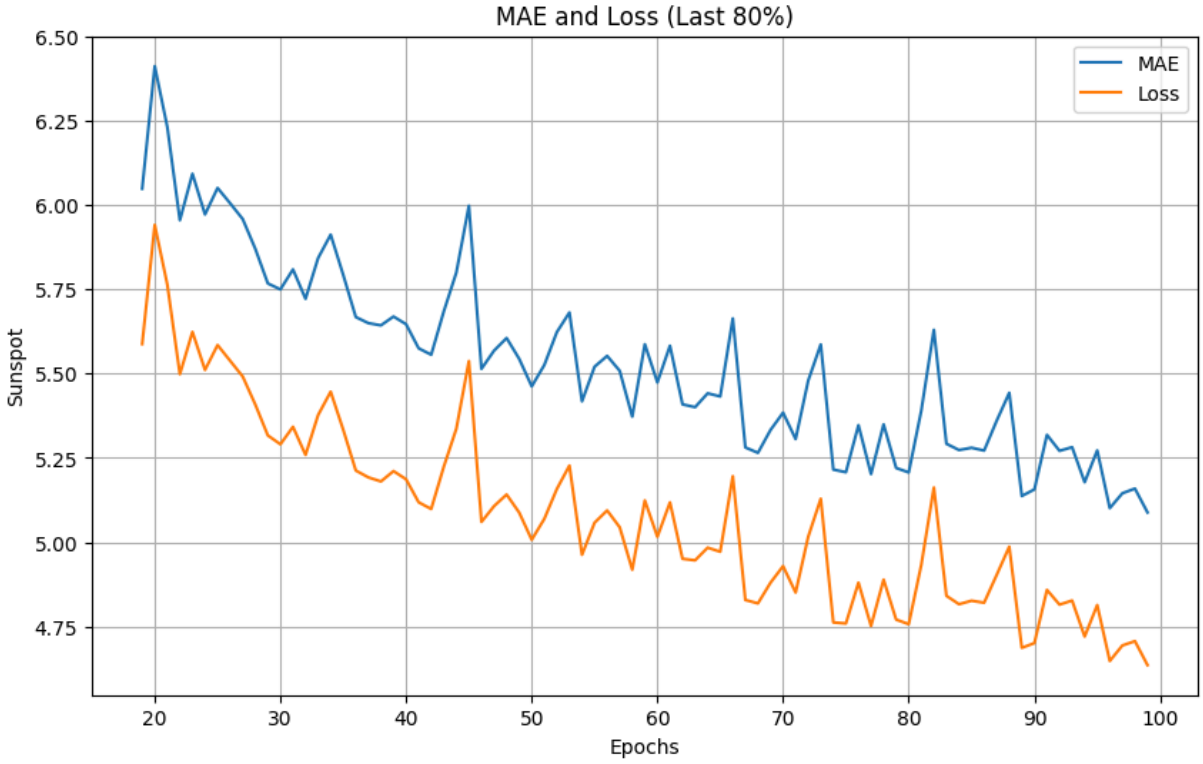




GRU Evaluation

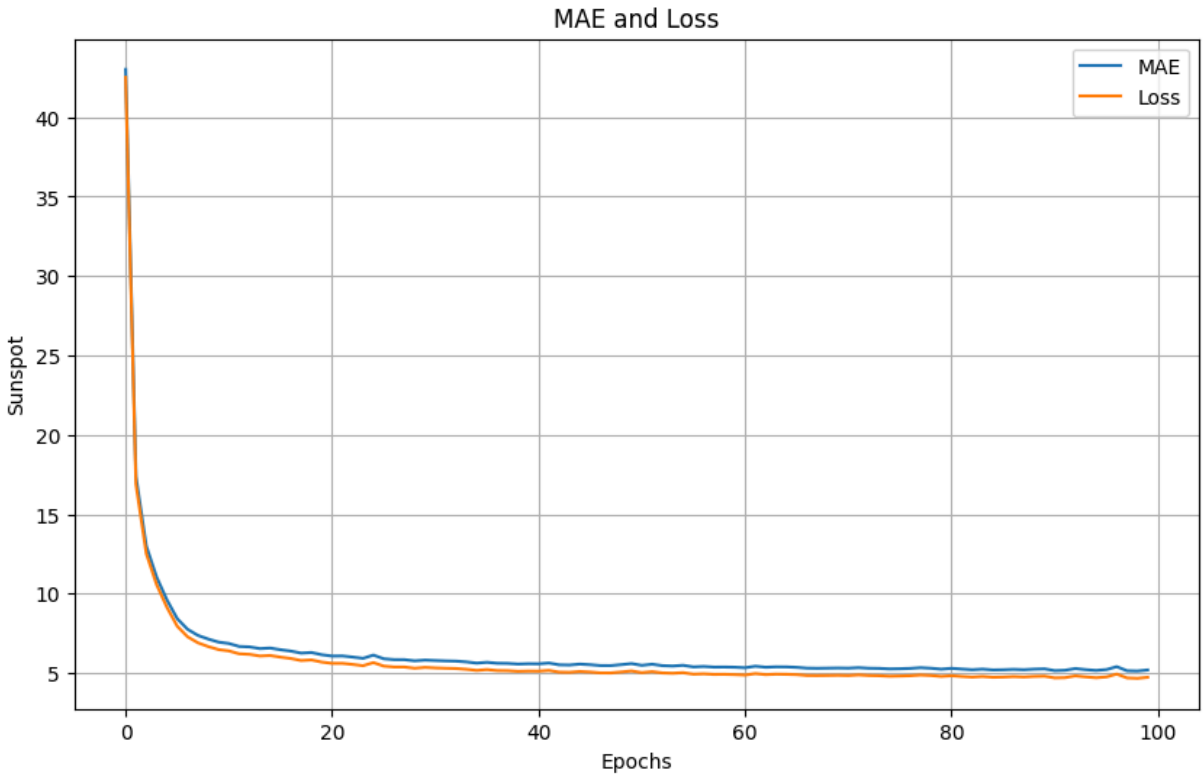
In [29]:
```
visualize_evaluation(history_GRU)
```

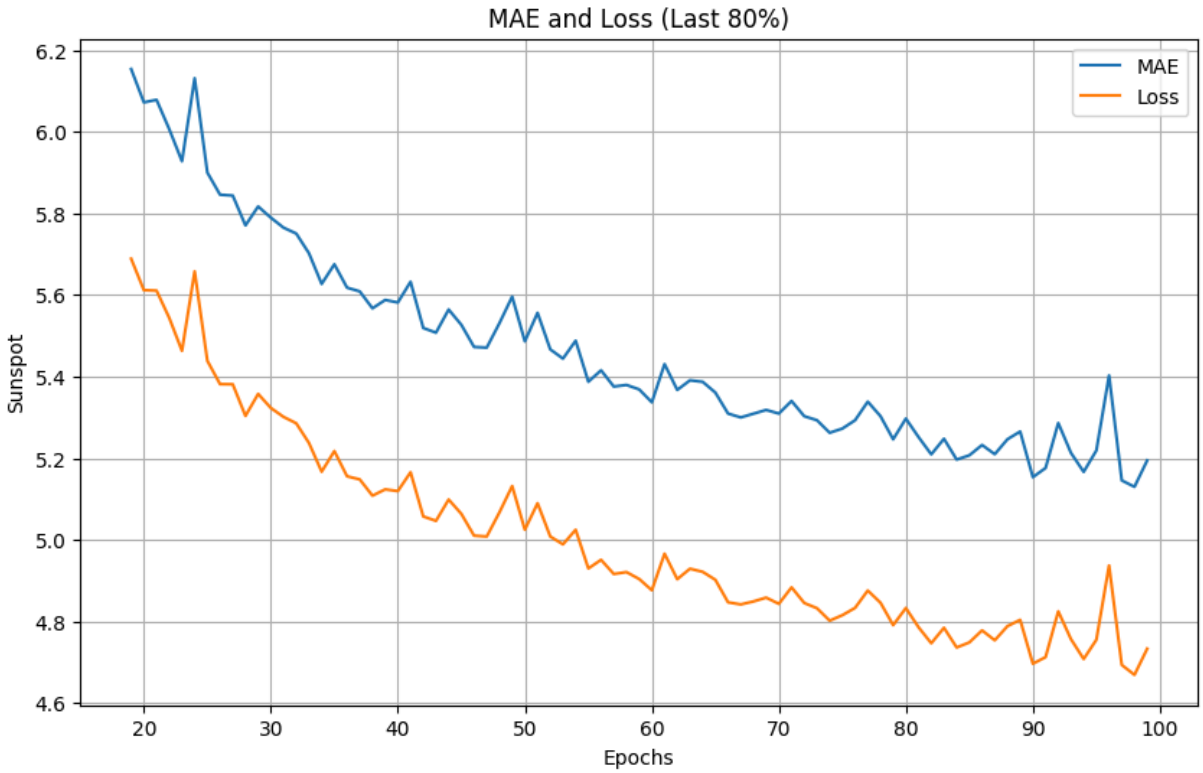




RNN-LSTM Evaluation

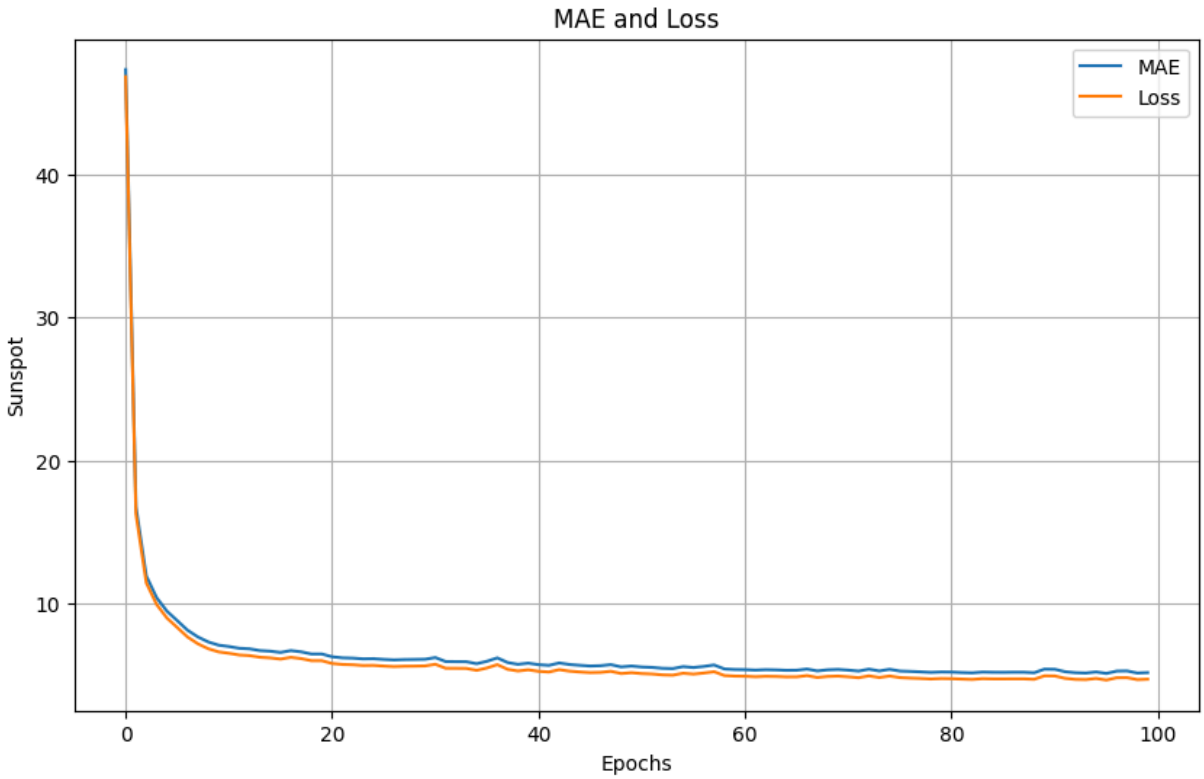
In [30]:
```
visualize_evaluation(history_RNN_LSTM)
```





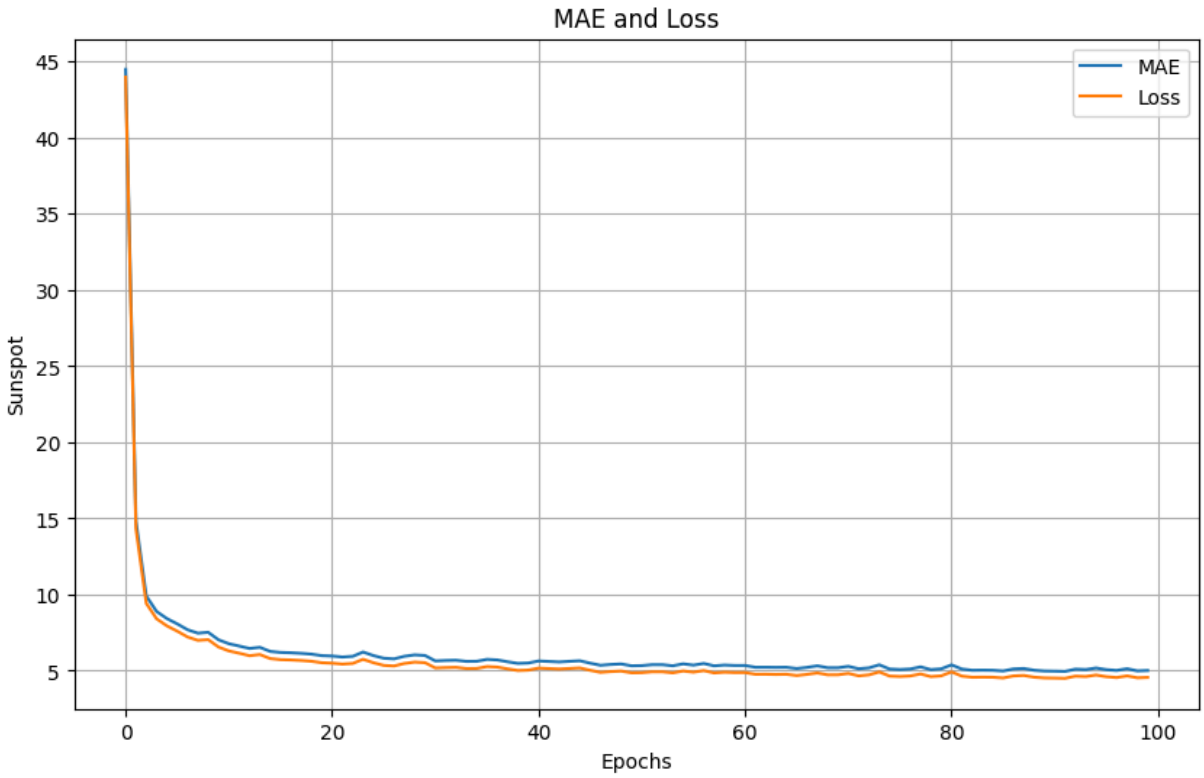
RNN-GRU Evaluation

In [31]:
```
visualize_evaluation(history_RNN_GRU)
```



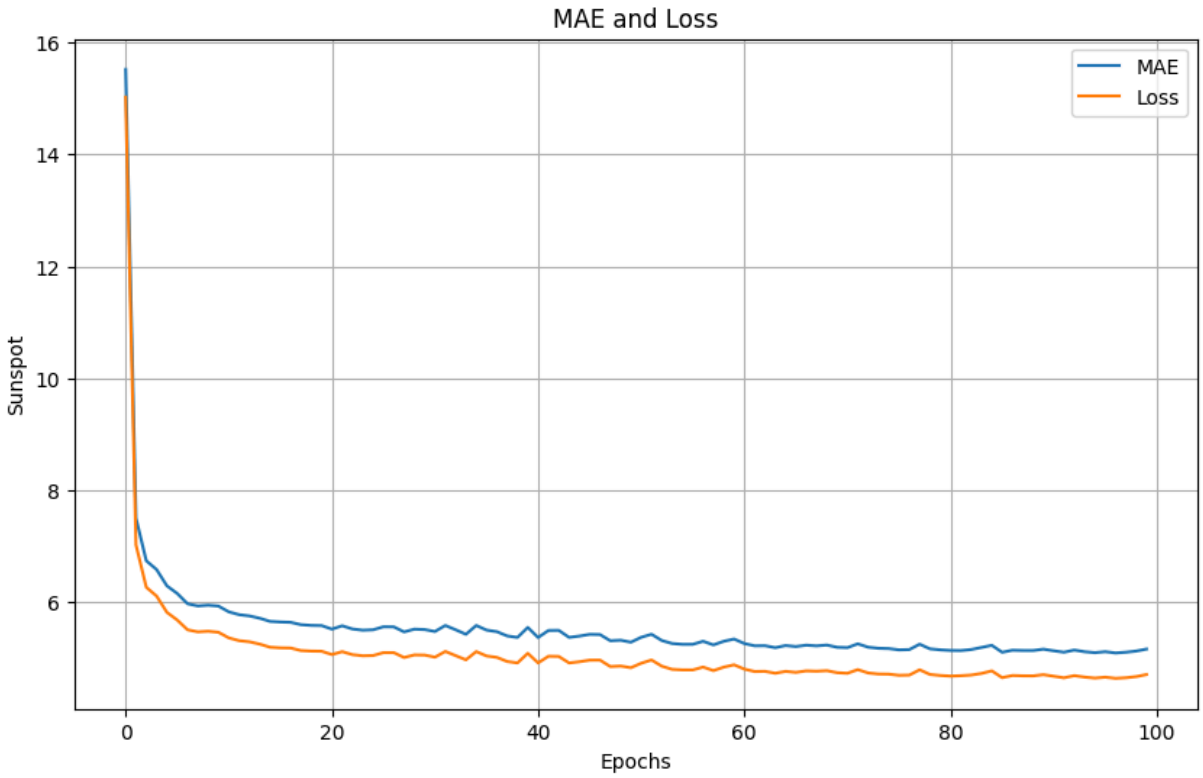
LSTM-RNN Evaluation

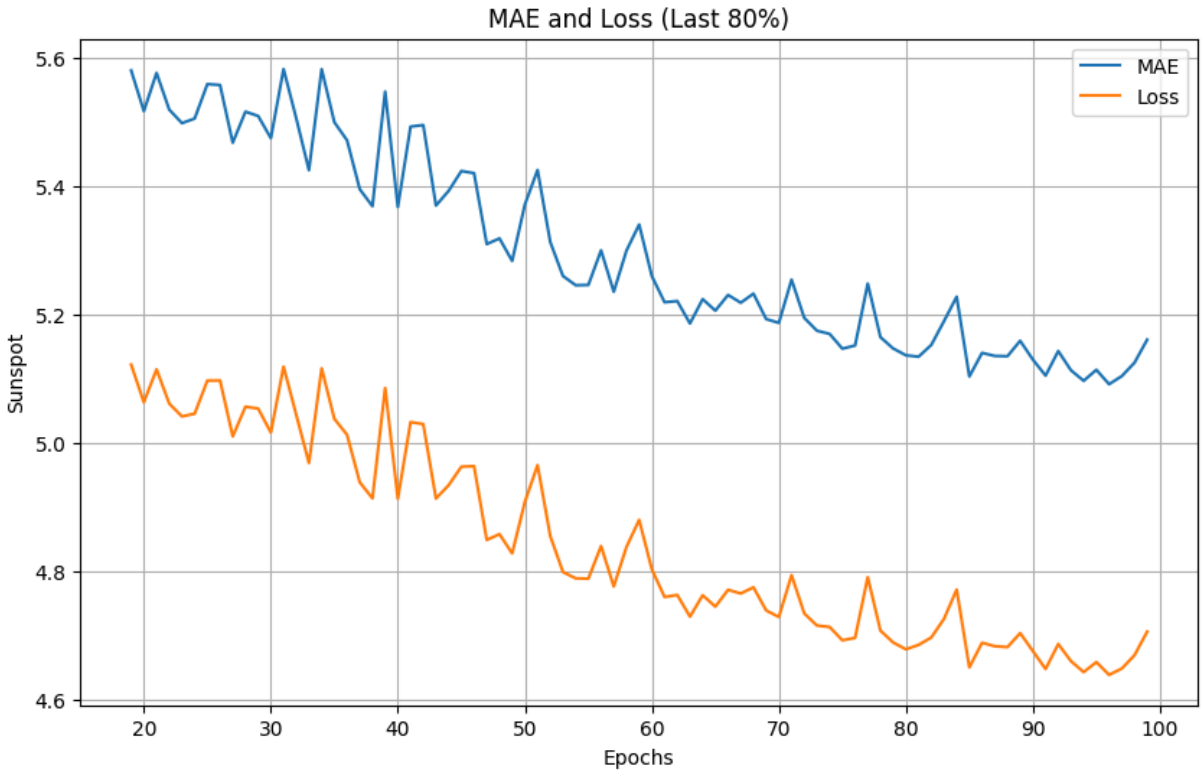
In [32]:
```
visualize_evaluation(history_LSTM_RNN)
```



LSTM-GRU Evaluation

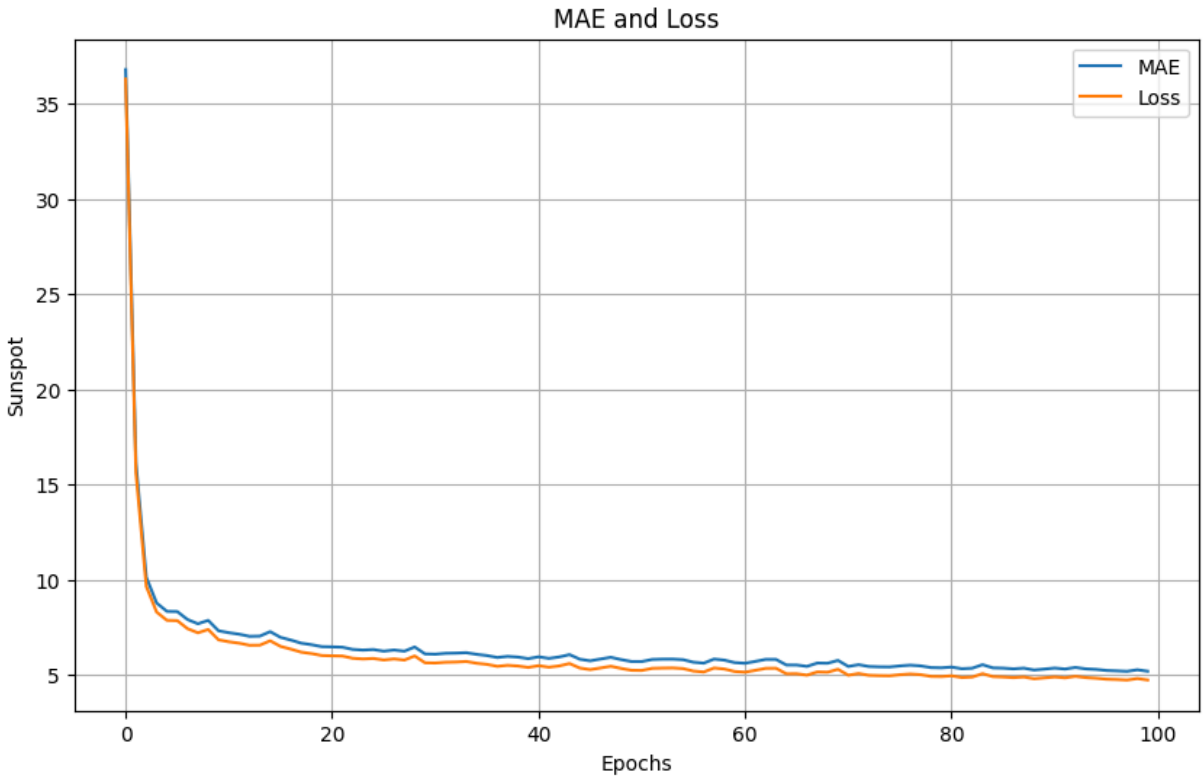
In [33]:
```
visualize_evaluation(history_LSTM_GRU)
```

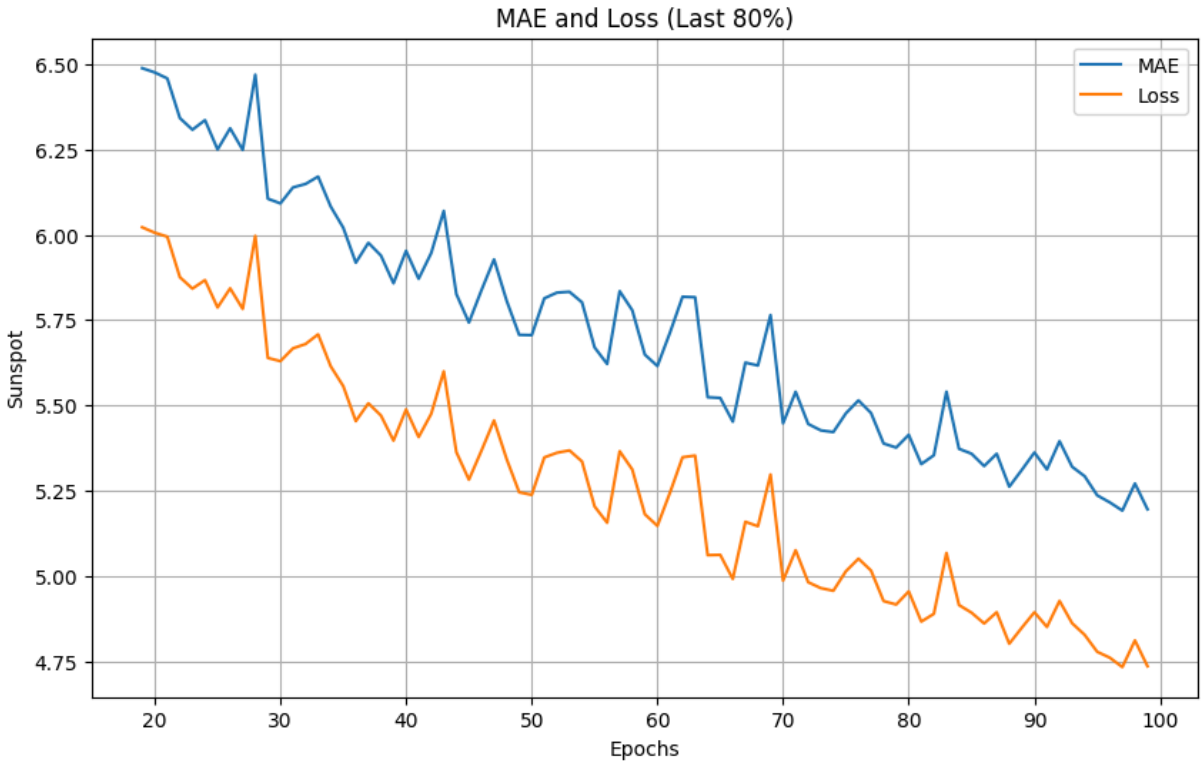




GRU-RNN Evaluation

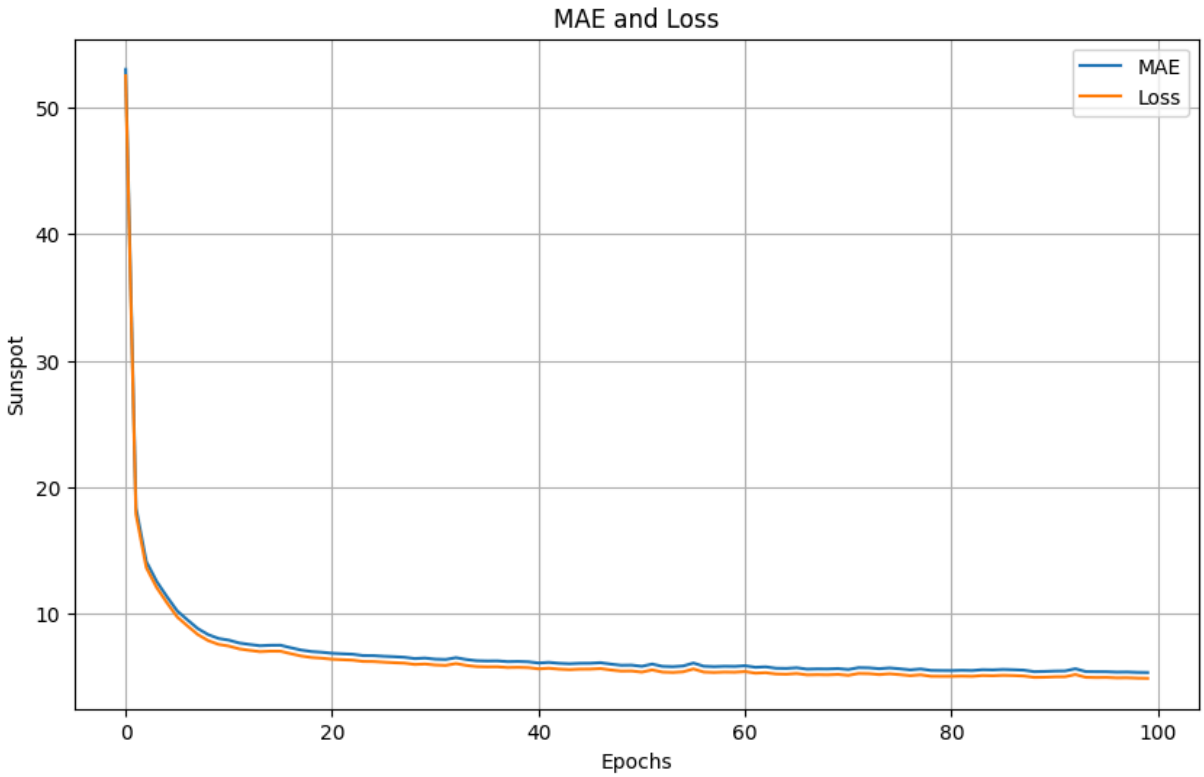
In [34]:
```
visualize_evaluation(history_GRU_RNN)
```

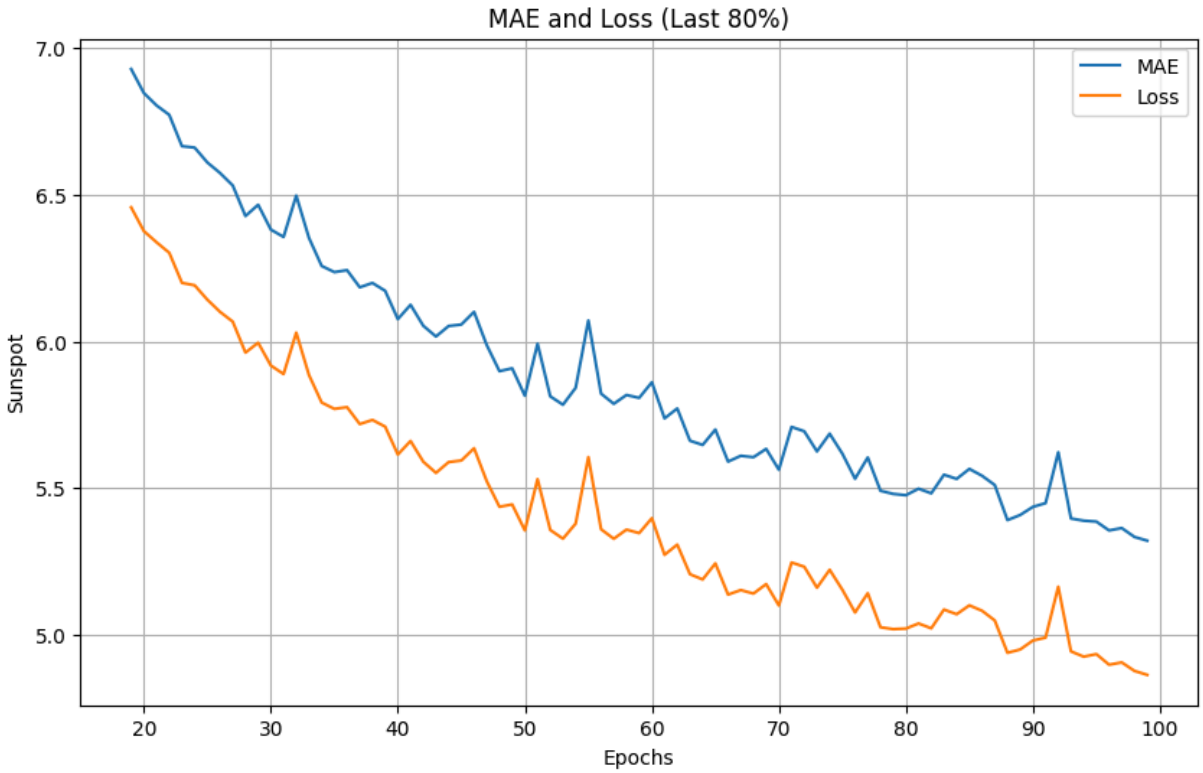




GRU-LSTM Evaluation

In [35]:
```
visualize_evaluation(history_GRU_LSTM)
```





7. Model Prediction

Analysis outcome of clustering for whale transaction characteristic

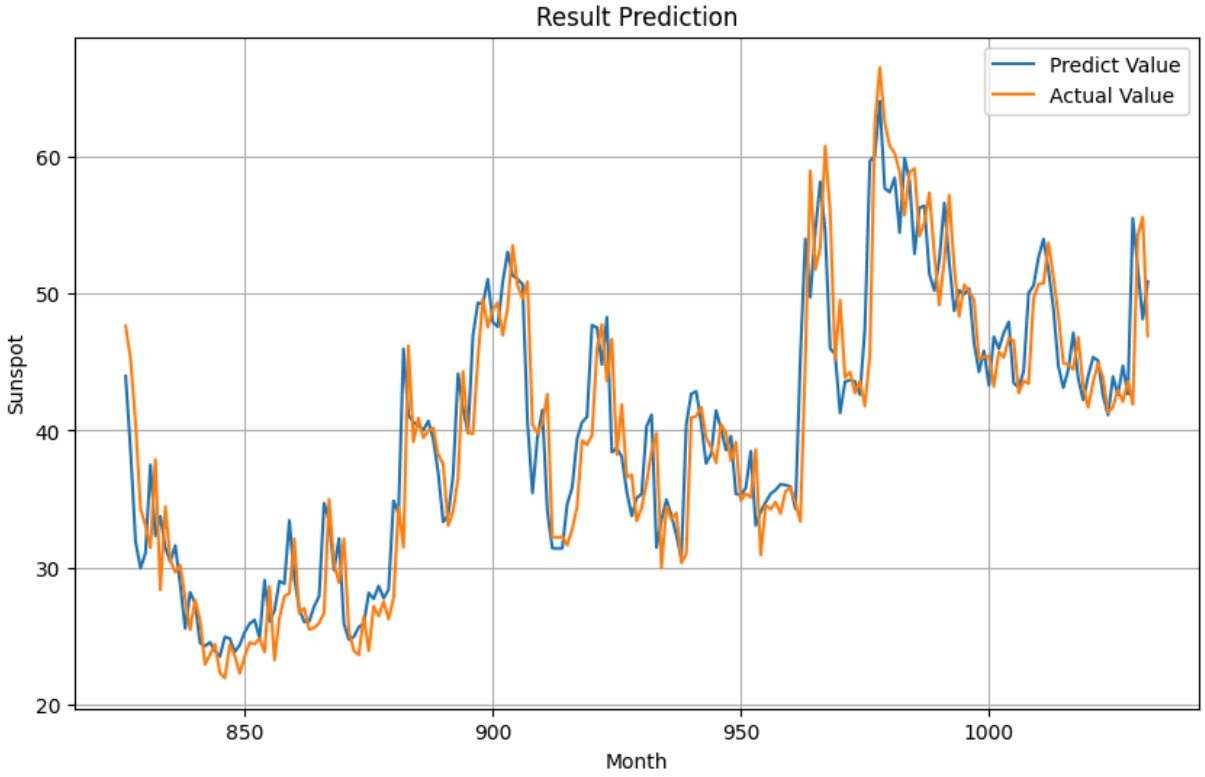
In [36]:
```
def model_forecast(model, series, window_size, batch_size):
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda w: w.batch(window_size))
    dataset = dataset.batch(batch_size).prefetch(1)
    forecast = model.predict(dataset)
    return forecast
```

In [37]:
```
def predict(model):
  forecast_series = series[split_time-window_size:-1]
  forecast = model_forecast(model, forecast_series, window_size, batch_size)
  results = forecast.squeeze()
  plot_series(time_valid, (x_valid, results, ), title='Result Prediction',
  xlabel='Month',
  ylabel='Sunspot',
  legend=['Predict Value', 'Actual Value'])
```

RNN Prediction

In [38]:
```
predict(model_RNN)
```

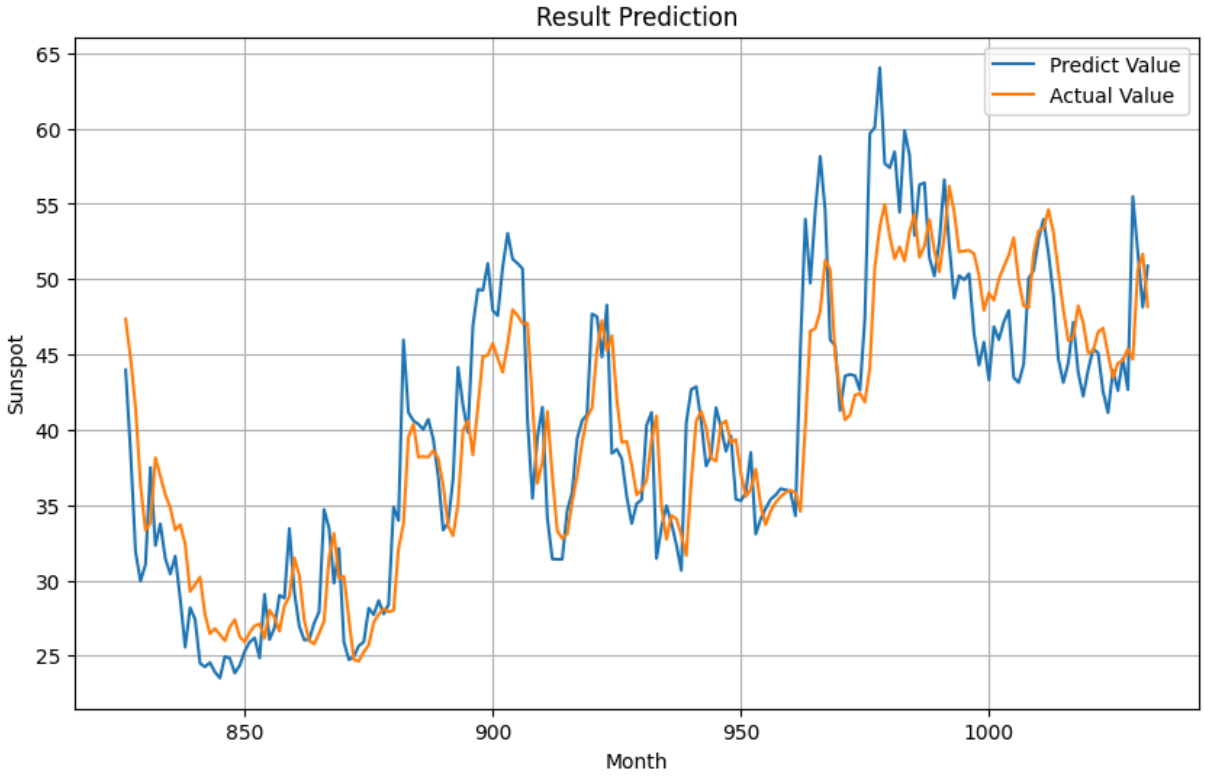
```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step
```



LSTM Prediction

In [39]:
```
predict(model_LSTM)
```

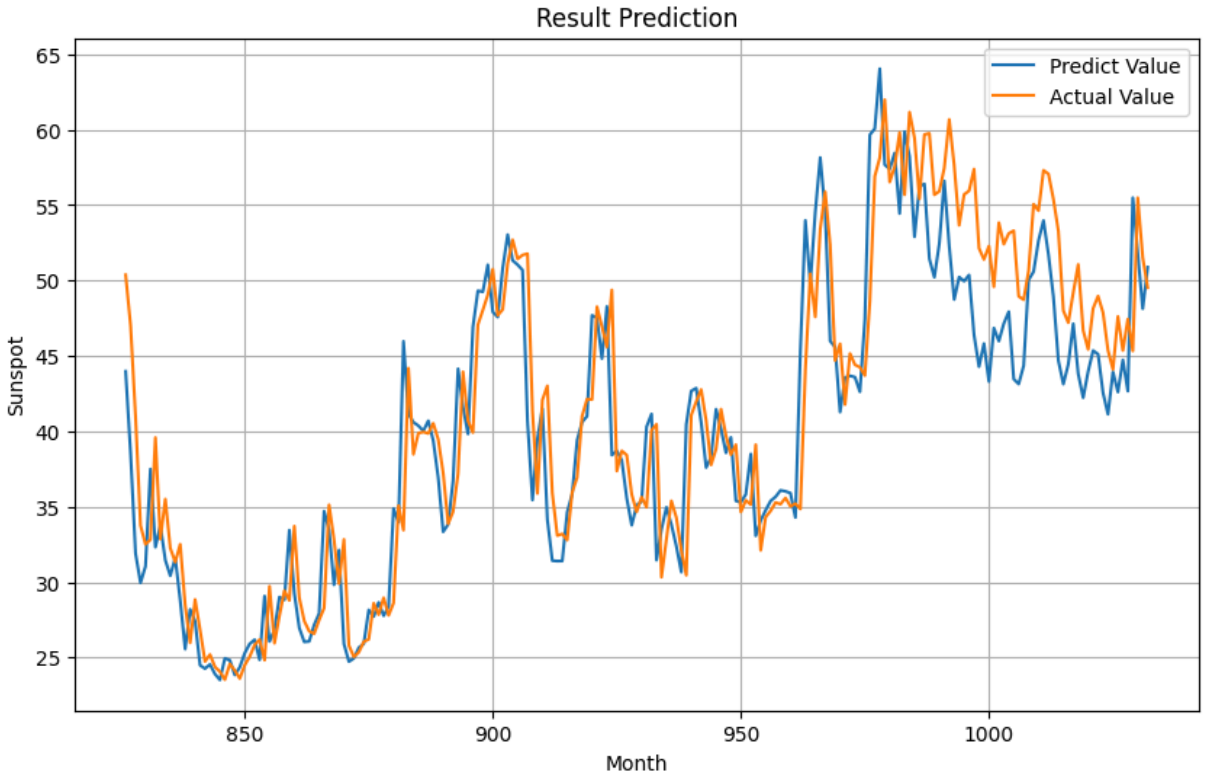
```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step
```



GRU Prediction

In [40]:
```
predict(model_GRU)
```

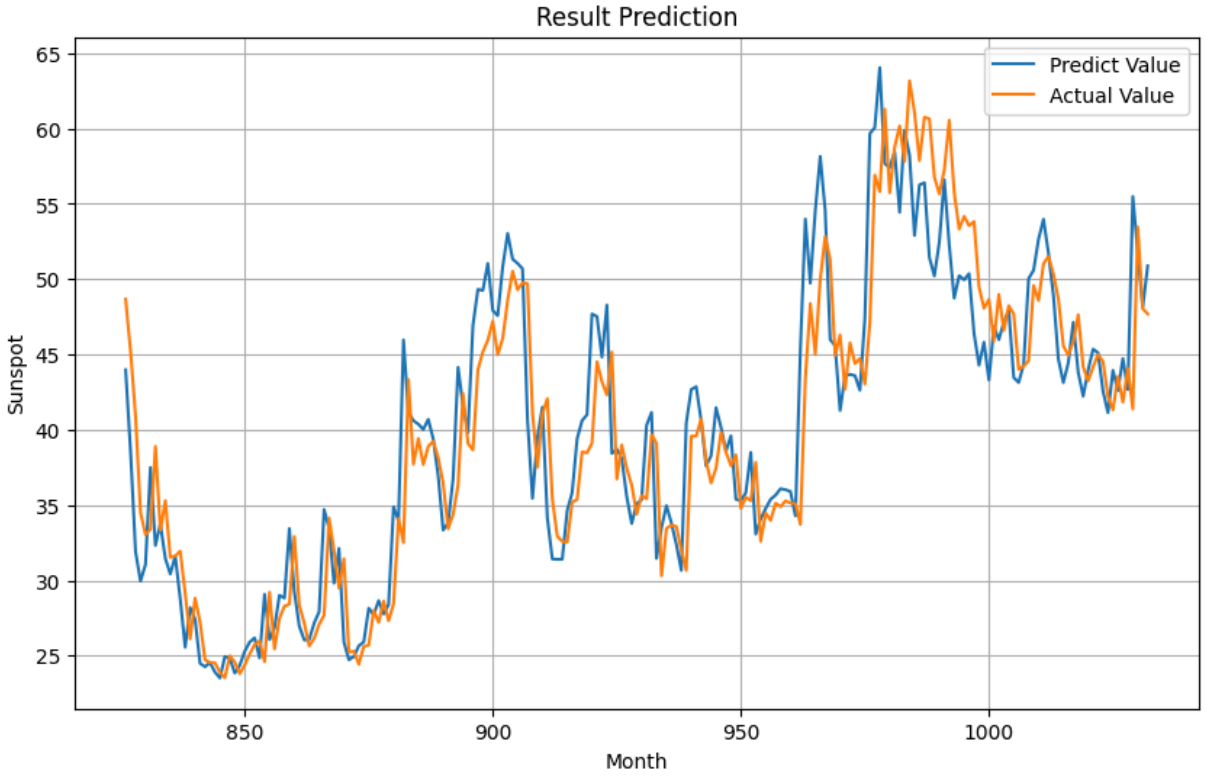
```
WARNING:tensorflow:5 out of the last 15 calls to <function TensorFlowTrainer.make_predict_funct
ion.<locals>.one_step_on_data_distributed at 0x7ef1d08dc700> triggered tf.function retracing. T
racing is expensive and the excessive number of tracings could be due to (1) creating @tf.funct
ion repeatedly 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.tensor
flow.org/api_docs/python/tf/function for  more details.
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 106ms/step
```



RNN-LSTM Prediction

In [41]:
```
predict(model_RNN_LSTM)
```

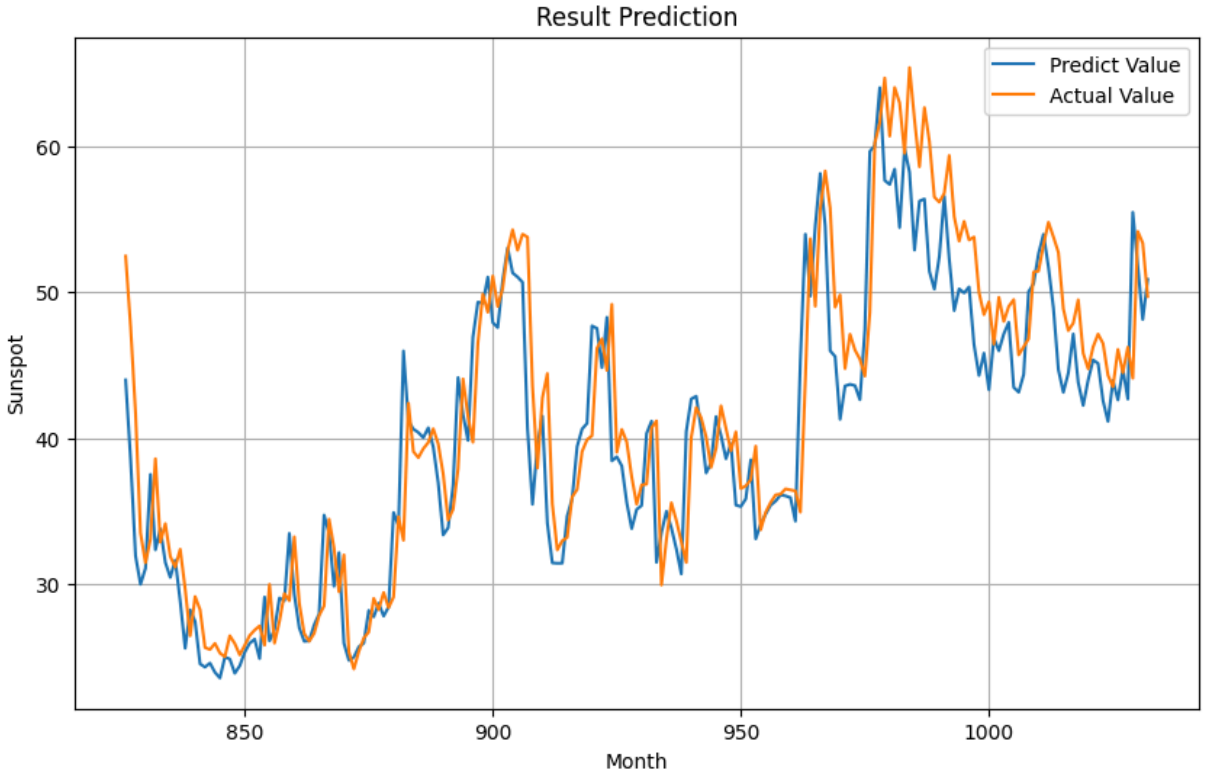
```
WARNING:tensorflow:5 out of the last 15 calls to <function TensorFlowTrainer.make_predict_funct
ion.<locals>.one_step_on_data_distributed at 0x7ef1cf659b40> triggered tf.function retracing. T
racing is expensive and the excessive number of tracings could be due to (1) creating @tf.funct
ion repeatedly 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.tensor
flow.org/api_docs/python/tf/function for  more details.
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 87ms/step
```



RNN-GRU Prediction

In [42]:
```
predict(model_RNN_GRU)
```

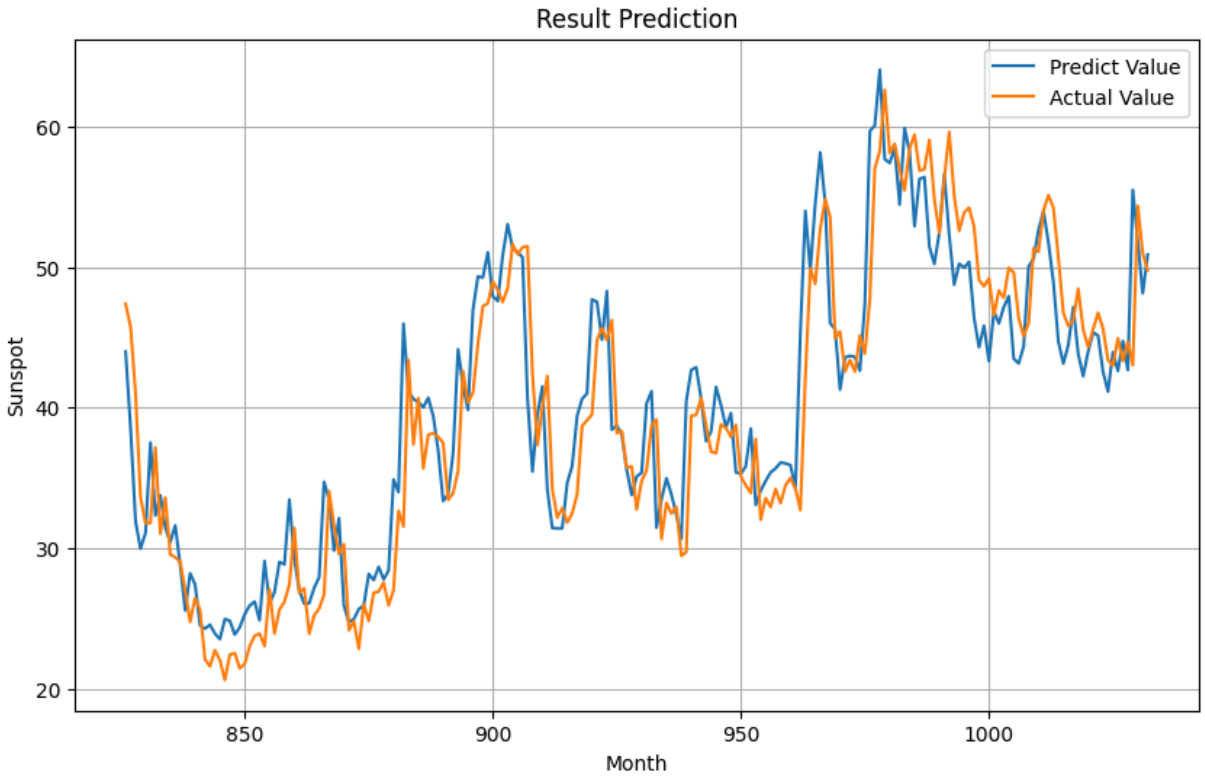
```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step
```



LSTM-RNN Prediction

In [43]:
```
predict(model_LSTM_RNN)
```

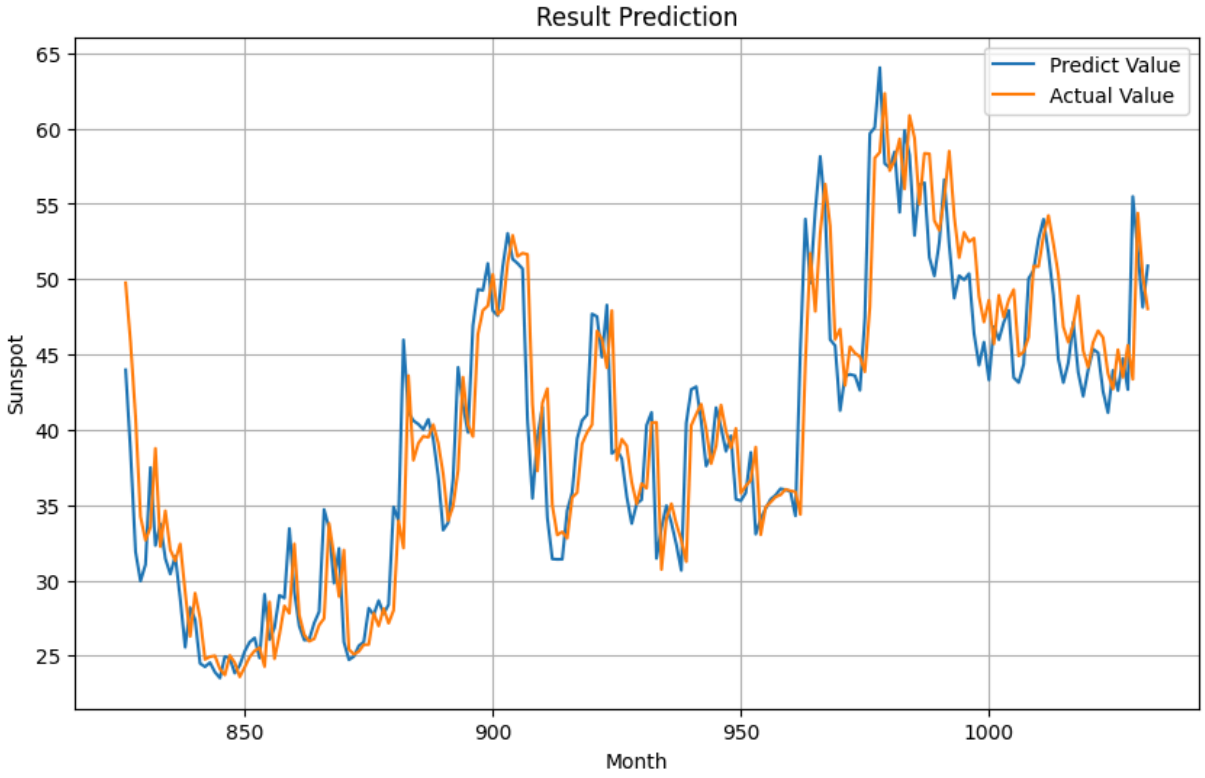
```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 57ms/step
```



LSTM-GRU Prediction

In [44]:
```
predict(model_LSTM_GRU)
```

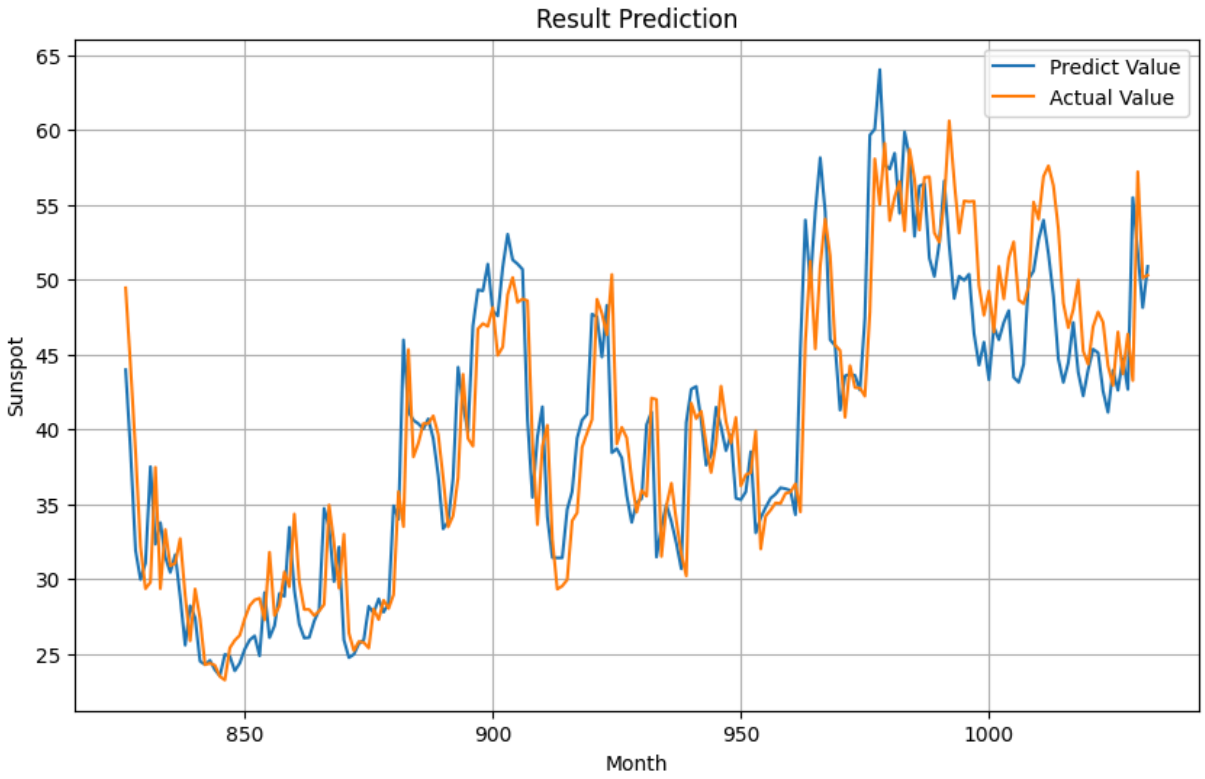
```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step
```



GRU-RNN Prediction

In [45]:
```
predict(model_GRU_RNN)
```

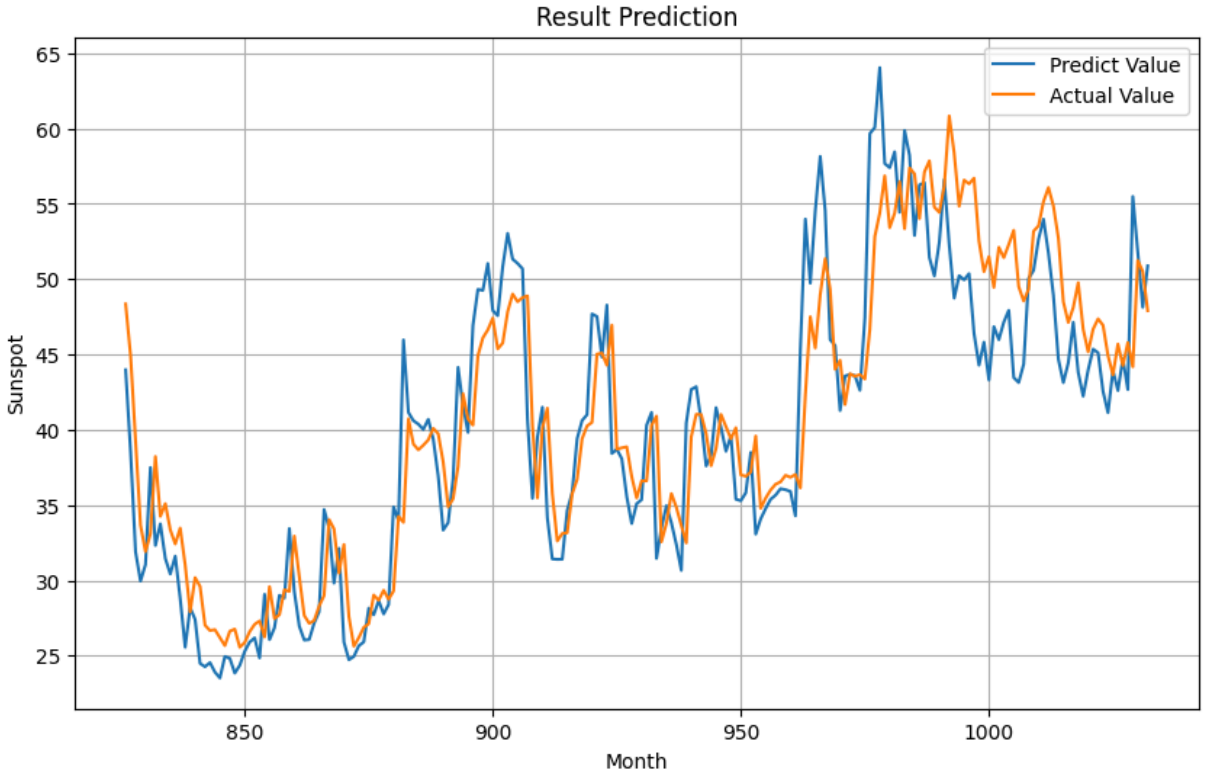
```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step
```



GRU-LSTM Prediction

In [46]:
```
predict(model_GRU_LSTM)
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 69ms/step
```



Confident Interval

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import files
import time
```

Donwload Data

RNN

In [ ]:
```
!gdown 1dy_cvZMu9GEih7XJ5_qNzw8LPyqqXg-Q
!gdown 1AkKbVgI-HwdyFZWb7uyfseR5J-F9Hnw5
!gdown 1-qA9uOMkKEumT1MhrtMKOqVEjrjn7EAV
!gdown 1RP404GspPAsdEb_xBKAf8pFnR1SRecgE
!gdown 1JOwC0UN2z6JPK20OX1VgvOdBtFjrJAVy
```

LSTM

In [ ]:
```
!gdown 1ek2LrQEmi9kH98jtLZuD1rzKYdLswrcK
!gdown 1Y8eAijF4Zehrng5tpKyMFJ1ZEu9sxtA3
!gdown 1-AVY6E8UV4KuReuexQjwckSe-gDAYeJ4
!gdown 1DOkptkv8uiE3BwguxmFjsF8YRl4U-6ht
!gdown 1IJjiJMO48CiT32bJmreBe-7PxmzCqIdh
```

GRU

In [ ]:
```
!gdown 1D6ptowJZ3Dt04QAhfkB0AEi3doutcDoh
!gdown 1CnWWdLQEICuMAzAgoMa3ZMuHYteUVNTD
!gdown 1XKIbo7yb_9a-RFqqJ4GrVGNUw-tlmGYq
!gdown 1ncY7FLHJsSNjKxipWQLg9yAZnpVEQEbh
!gdown 1mJEKKy20pmczLrcchqcZTzwZCz7r4jY0
```

RNN-LSTM

In [ ]:
```
!gdown 1Y8-y9ZQ7AbEQNnh5ABiSRXLVWt8_cZLd
!gdown 1U_DR-mizdvJ4z-pDFf8vyxCbi31CCtj0
!gdown 1WNK8MG7qILDs7vR_-h2b15vt9N8eTkZP
!gdown 1yZrI6VmJsKp22byajzVjYhCzN2QgfDwF
!gdown 1p9CEqsWM4pq2f7lTjpdhZkTx71SpyRTp
```

RNN-GRU

In [ ]:
```
!gdown 1vfag8vScvoEa_3_mxyzk4OFozMlmC4up
!gdown 19C0BgDp8LrvERVdsGSqulEkz_l9XInQH
!gdown 1WovGGEGYNoa-Pe-XQHbvoEy6dV-jKQb-
!gdown 1UopC67xwvVPX-Mc12kN-XSIPtqPSYvkH
!gdown 1cgWGQ4OPt1l4zPBEf8jBSKokqdo_Efvi
```

LSTM-RNN

In [ ]:
```
!gdown 1VomAv7QXYkBleRM3bbZkUlPHtwQ-BDJ0
!gdown 1seeOoYqCH7TJKS8sE7oirWbg1ruK4bFQ
!gdown 14ClrSSNu2e--YqvOtwsnxHZsGd-isA4j
!gdown 1wmaoWfxjLkTW4b89kovrmOWtb9GbY1OX
!gdown 1Gz7G03X8y8WD7mG3KN7_mTJVPkuMexi4
```

LSTM-GRU

In [ ]:
```
!gdown 1CwuhIWWBqYXmWWsZHs2wzbp6KW_dsnYN
!gdown 1qAjGQNxr1T66E55YmvkT4mTkEfIPKYb2
!gdown 1jszRT5McI9ukT9GBZn0vEVq_oMKO1S40
!gdown 1iYUflkad60Vmq2XU37VGHTidOnks5pNs
!gdown 1EZ7tRYNi8vG9GZo44RuZKHOGq1j2p8MH
```

Viz

In [ ]:
```
from re import X
def shade_plot(temp, x_valid='0', line='MAE', predict=''):
  import matplotlib.pyplot as plt

  if predict == '':
    redline = f'Mean {line}'
    axis = 'Epochs'
  else:
    redline = 'Actual Value'
    axis = 'Days'

  #SORT
  temp = temp.apply(lambda x: x.sort_values().reset_index(drop=True))

  #CUT 2.5%
  n = int(len(temp) * 0.025)
  temp = temp.iloc[n:-n]

  batas_atas = temp.max().tolist()
  batas_bawah = temp.min().tolist()
  rata = temp.mean().tolist()

  # Replace these lists with your actual data
  # Dataset 1
  x1 = [i for i in range(0,len(batas_atas))]
  y1 = batas_atas

  # Dataset 2
  x2 = [i for i in range(0,len(batas_bawah))]
  y2 = batas_bawah

  if(x_valid=='0'):
    # Dataset 3
    x3 = [i for i in range(0,len(rata))]
    y3 = rata
  else:
    x3 = [i for i in range(0,len(x_valid))]
    y3 = x_valid

  # Create the plot
  plt.figure(figsize=(8, 6))

  # Plot the first linear dataset
  plt.plot(x1, y1, label=f'Predicted Value (Max)', color='blue')

  # Plot the second linear dataset
  plt.plot(x2, y2, label=f'Predicted Value (Min)', color='green')

  plt.plot(x3, y3, label=f'{redline}', color='red')

  # Create a mask for the shaded area where y1 >= y2
  mask = [y1_val >= y2_val for y1_val, y2_val in zip(y1, y2)]

  # Shade the area between the two datasets
  plt.fill_between(x1, y1, y2, where=mask, interpolate=True, color='gray', alpha=0.5)

  # Add labels and legend
  plt.xlabel(f'{axis}')
```

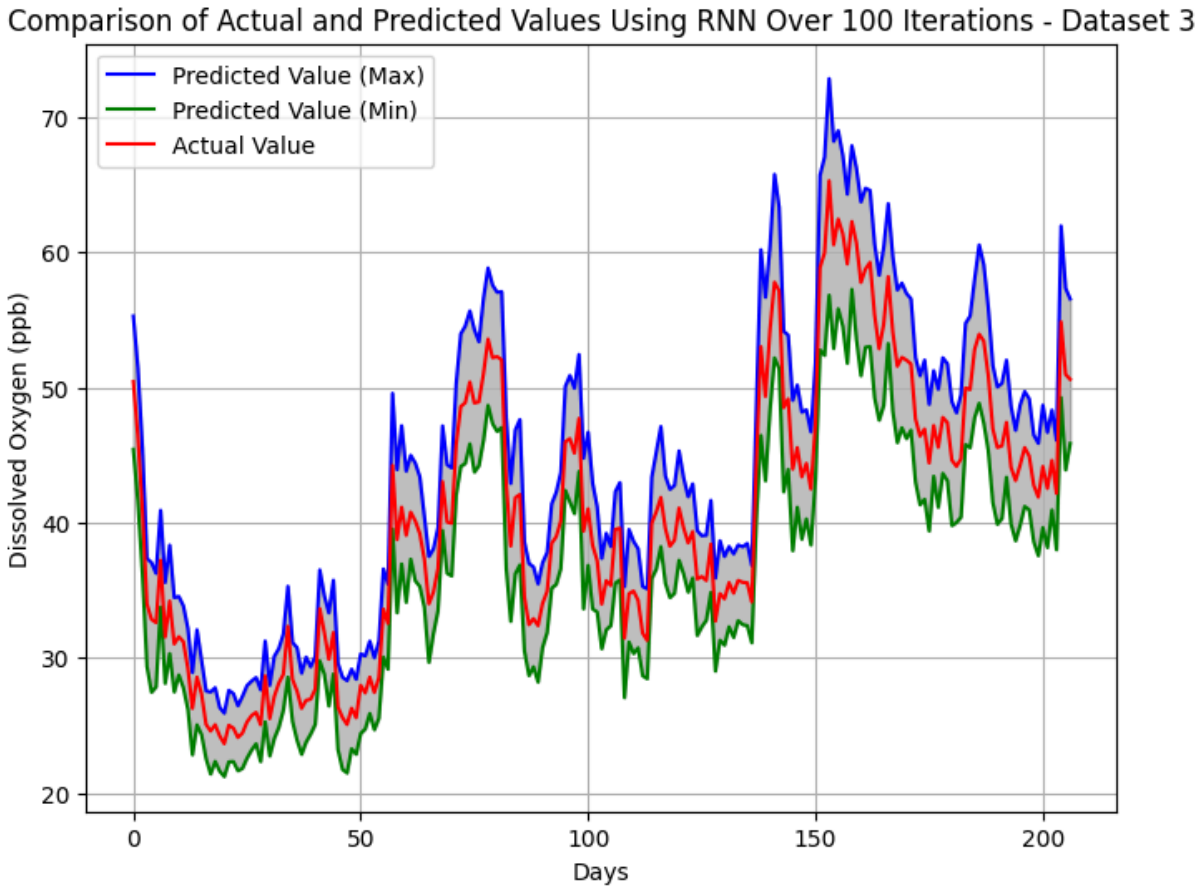
```
  plt.ylabel('Dissolved Oxygen (ppb)')
  plt.title(f'Comparison of Actual and Predicted Values Using {line} Over 100 Iterations - Data
  plt.legend()
  plt.savefig(f"Predicted_Actual_Value_Dataset3_{line}.pdf", dpi=300)
  files.download(f"Predicted_Actual_Value_Dataset3_{line}.pdf")
  # Show the plot
  plt.grid(True)
  plt.show()
```

In [ ]:
```
file2 = pd.read_excel('DissolvedOxygen.xlsx')
def take(file):
  time_step = []
  sunspots = []

  for i in range(len(file)):
    time_step.append(i)
    sunspots.append(file.iloc[i][1])
  time = np.array(time_step)
  series = np.array(sunspots)
  return time, series
time, series = take(file2)
split_time = int(len(time)*0.8)
def divide(time, series):
  split_time = int(len(time)*0.8)
  time_train = time[:split_time]
  x_train = series[:split_time]
  time_valid = time[split_time:]
  x_valid = series[split_time:]
  return x_train, time_train, time_valid, x_valid

def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
    dataset = dataset.map(lambda window: (window[:-1], window[-1]))
    dataset = dataset.shuffle(shuffle_buffer)
    dataset = dataset.batch(batch_size).prefetch(1)
    return dataset

window_size = 30
batch_size = 32
shuffle_buffer_size = 1000


window_size = 30
batch_size = 32
shuffle_buffer_size = 1000

time, series = take(file2)
x_train, time_train, time_valid, x_valid = divide(time, series)
```

```
<ipython-input-12-23662f301f4f>:8: FutureWarning: Series.__getitem__ treating keys as positions
is deprecated. In a future version, integer keys will always be treated as labels (consistent w
ith DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  sunspots.append(file.iloc[i][1])
```

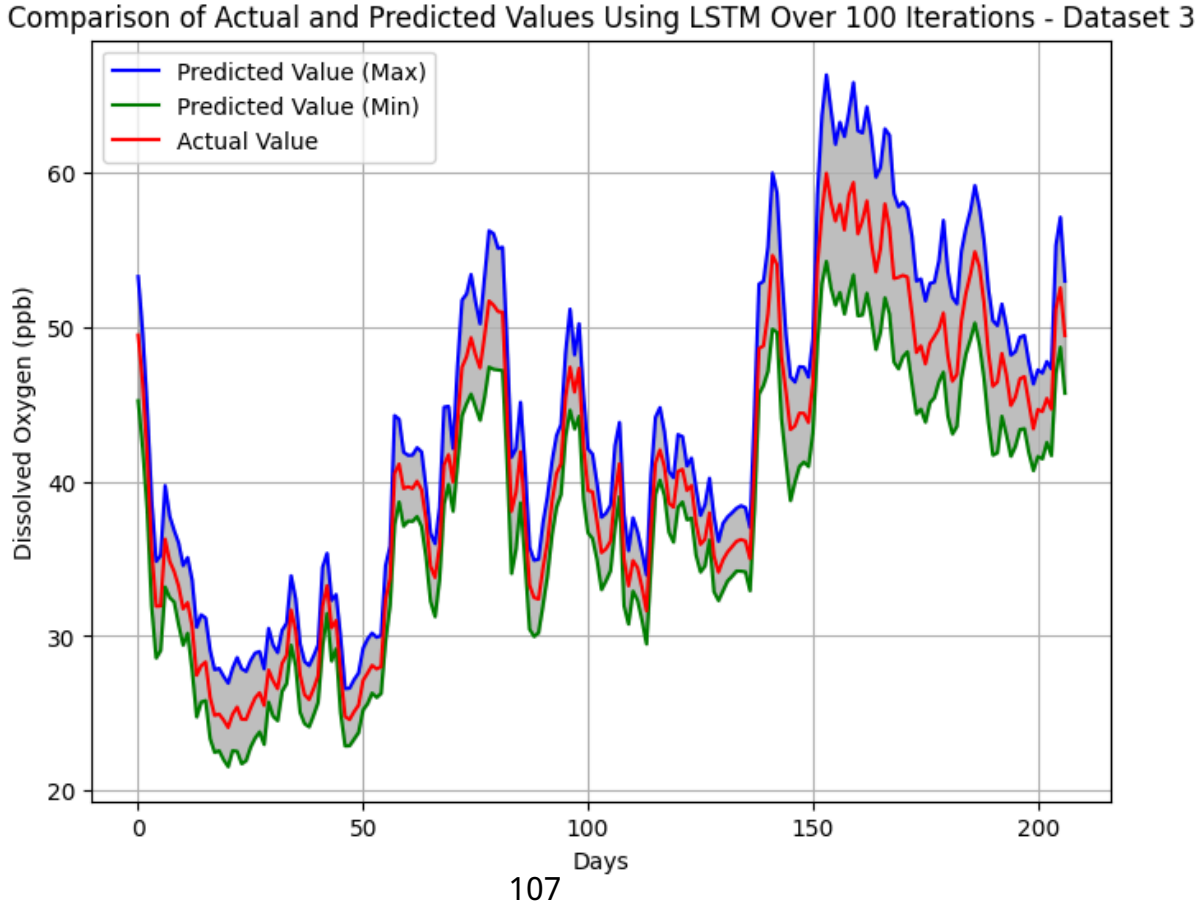
RNN

In [ ]:
```
temp = pd.read_excel('RNN_PREDICT.xlsx')
shade_plot(temp, line='RNN', predict='YES')
```



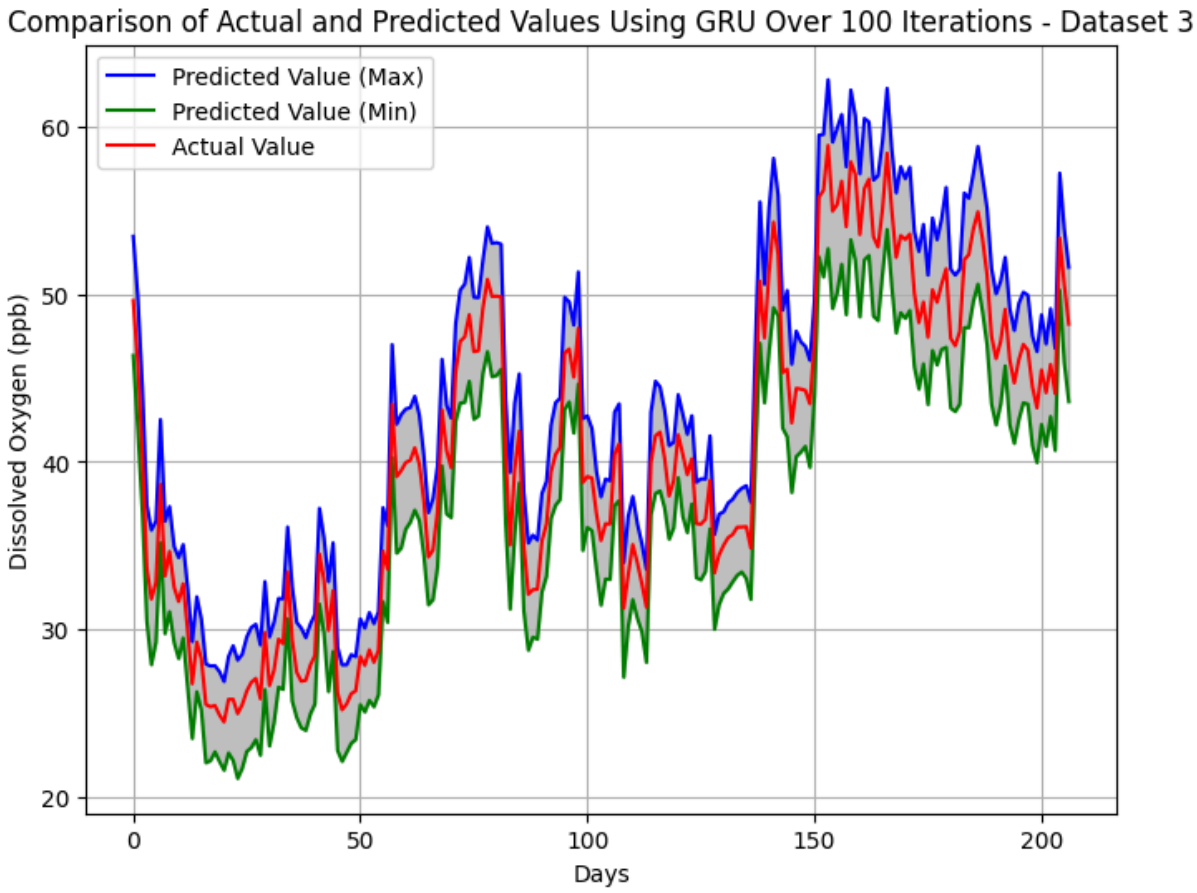
LSTM

In [ ]:
```
temp = pd.read_excel('LSTM_PREDICT.xlsx')
shade_plot(temp, line='LSTM', predict='YES')
```



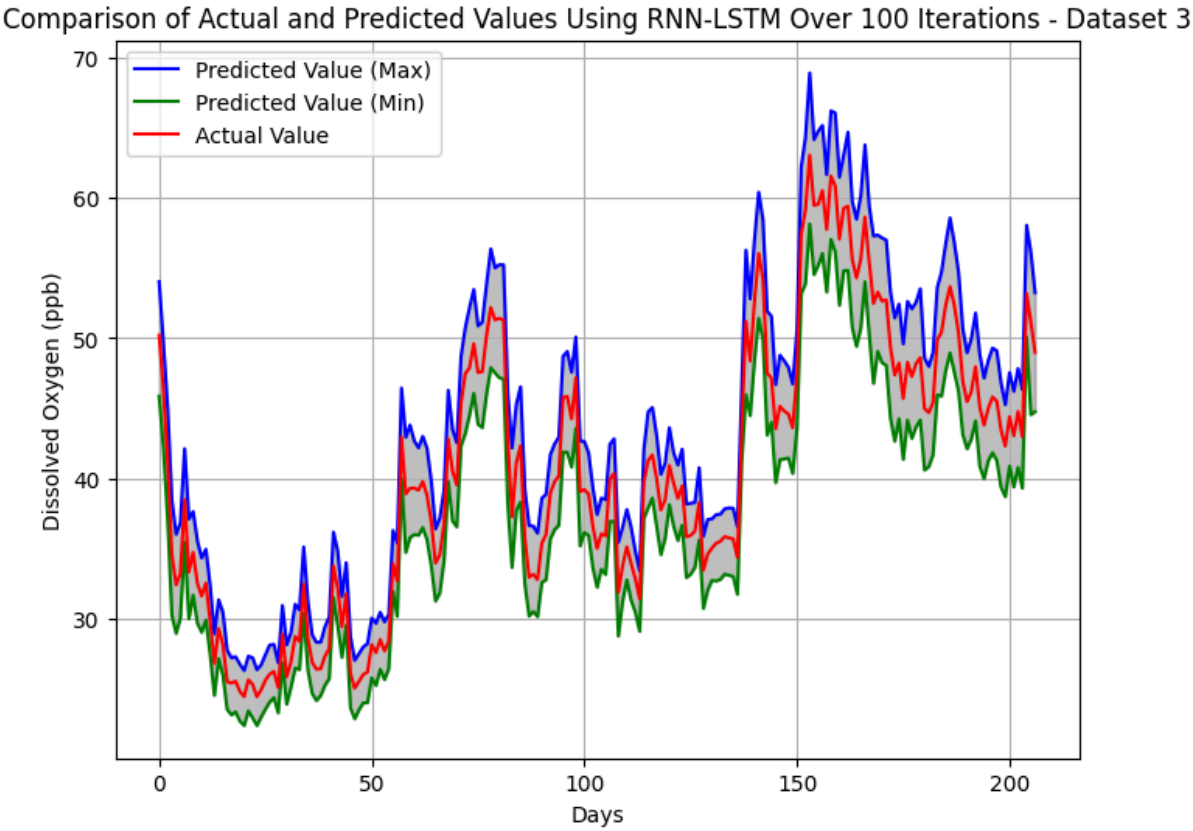
GRU

In [ ]:
```
temp = pd.read_excel('GRU_PREDICT.xlsx')
shade_plot(temp, line='GRU', predict='YES')
```



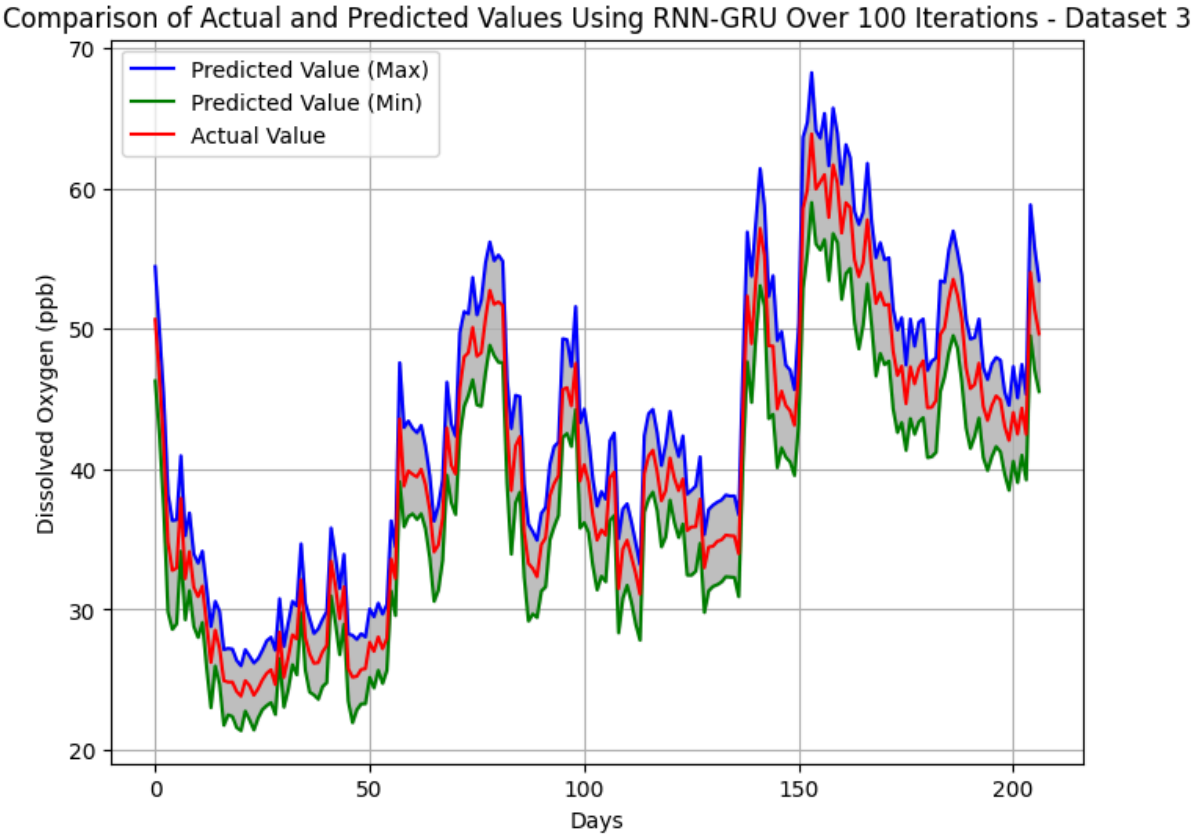
RNN-LSTM

In [ ]:
```
temp = pd.read_excel('RNN-LSTM_PREDICT.xlsx')
shade_plot(temp, line='RNN-LSTM', predict='YES')
```



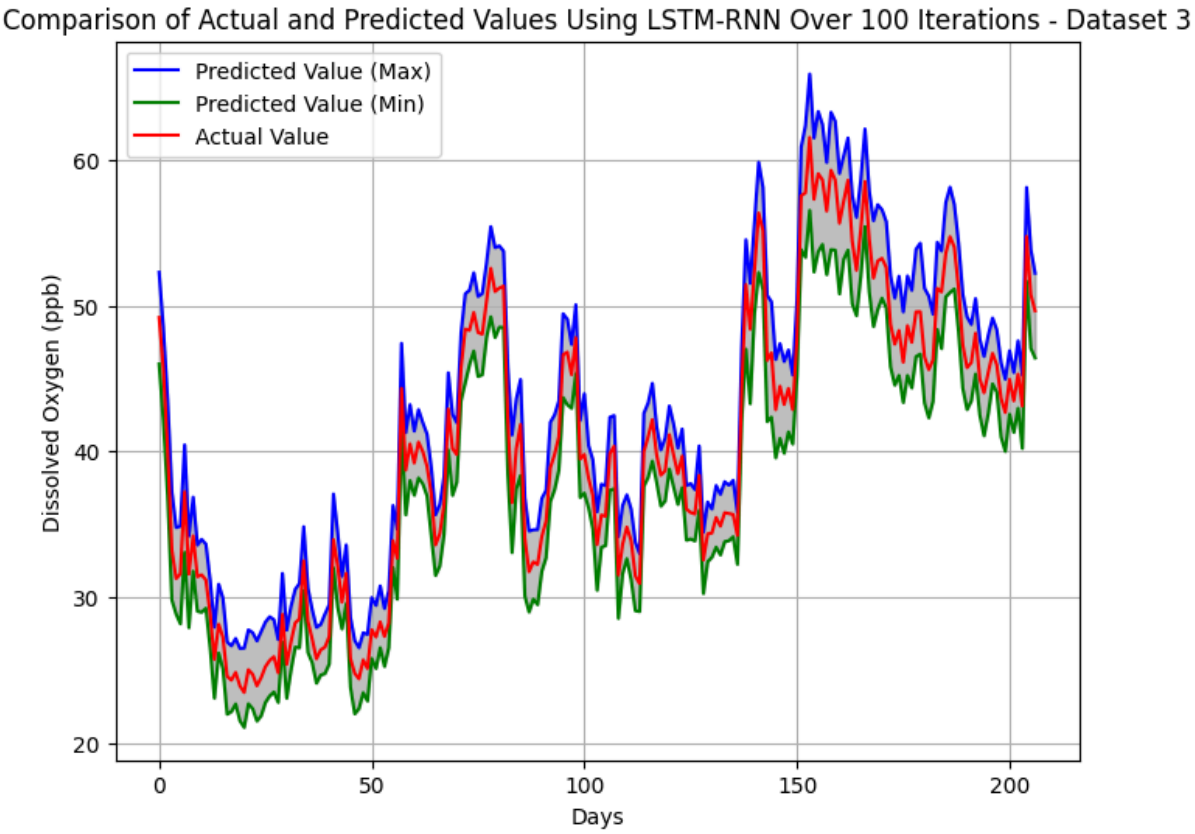
RNN-GRU

In [ ]:
```
temp = pd.read_excel('RNN-GRU_PREDICT.xlsx')
shade_plot(temp, line='RNN-GRU', predict='YES')
```



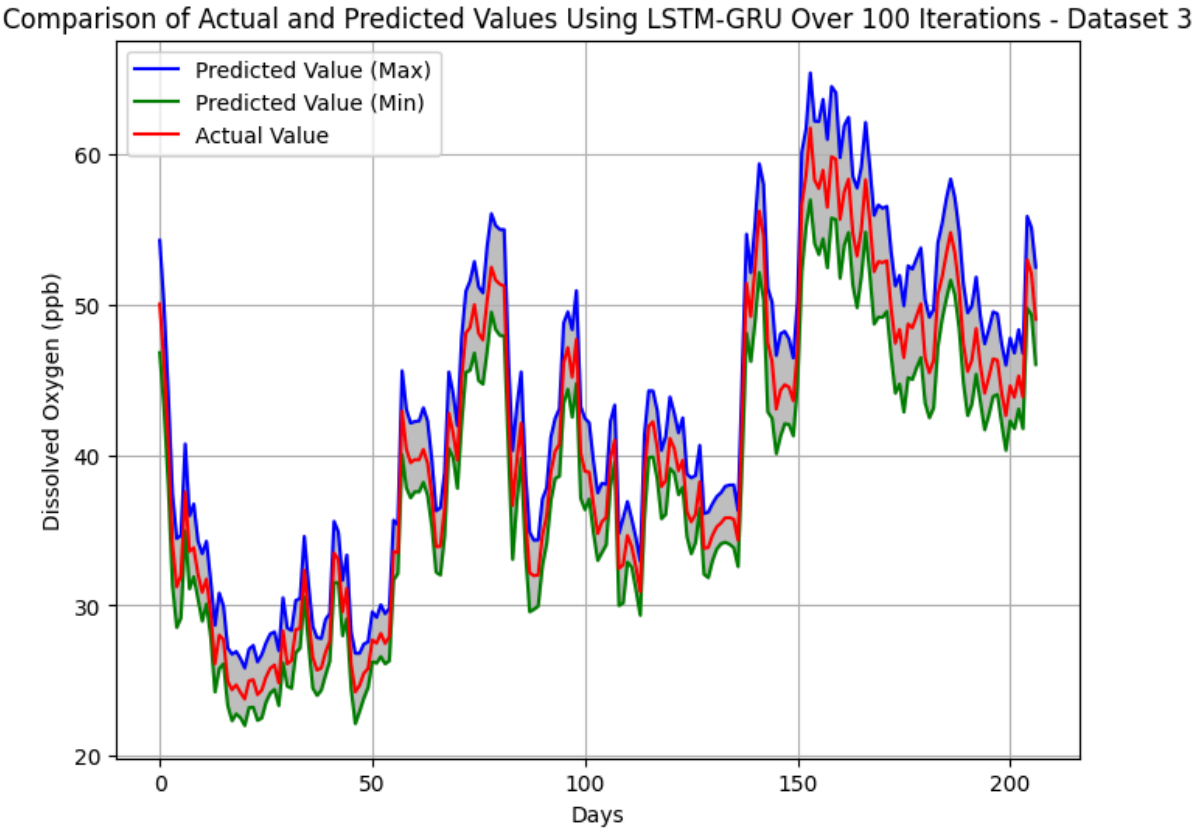
LSTM-RNN

In [ ]:
```
temp = pd.read_excel('LSTM-RNN_PREDICT.xlsx')
shade_plot(temp, line='LSTM-RNN', predict='YES')
```



LSTM-GRU

In [ ]:
```
temp = pd.read_excel('LSTM-GRU_PREDICT.xlsx')
shade_plot(temp, line='LSTM-GRU', predict='YES')
```


Comparison of Actual and Predicted Values Using LSTM-GRU Over 100 Iterations - Dataset 3
Predicted Value (Max)
Predicted Value (Min)
Actual Value
Dissolved Oxygen (ppb)
Days
20
30
40
50
60
0
50
100
150
200

Donwload Data

GRU-RNN

In [ ]:
```
!gdown 1t7-iBKRzQIu9xyVEvAGwMwtJ4_-PdtH0
!gdown 1lkWnA9uiWtsXQgCPy1vmEyODXcTUma1E
!gdown 1yr9Iaj7aC3rhcs4ZOXpGChB8DZDJh9G9
!gdown 1LlKEROW_YGadtr1r3L9Lo8OSkoLWxHpv
!gdown 1hJ3d58Un9Ym3AeCGQURp_pq5sczWU-8H
```

GRU-LSTM

In [ ]:
```
!gdown 1QdQWUlWbVoi0mT7LwJD-DYu1Db-py2Bv
!gdown 1__szgBSVNnyRh5Zw-83Aotu_YEzZxIWf
!gdown 1vdGkz58BeqRNNr6BxfMCHcNOvcvGirVD
!gdown 1_rPgnFGmk_e1NlZ4LHS9Hde_63TUucMw
!gdown 1eL4hMtU3L6wG3yf3Mf6UoCP1ZTW6Di6r
```

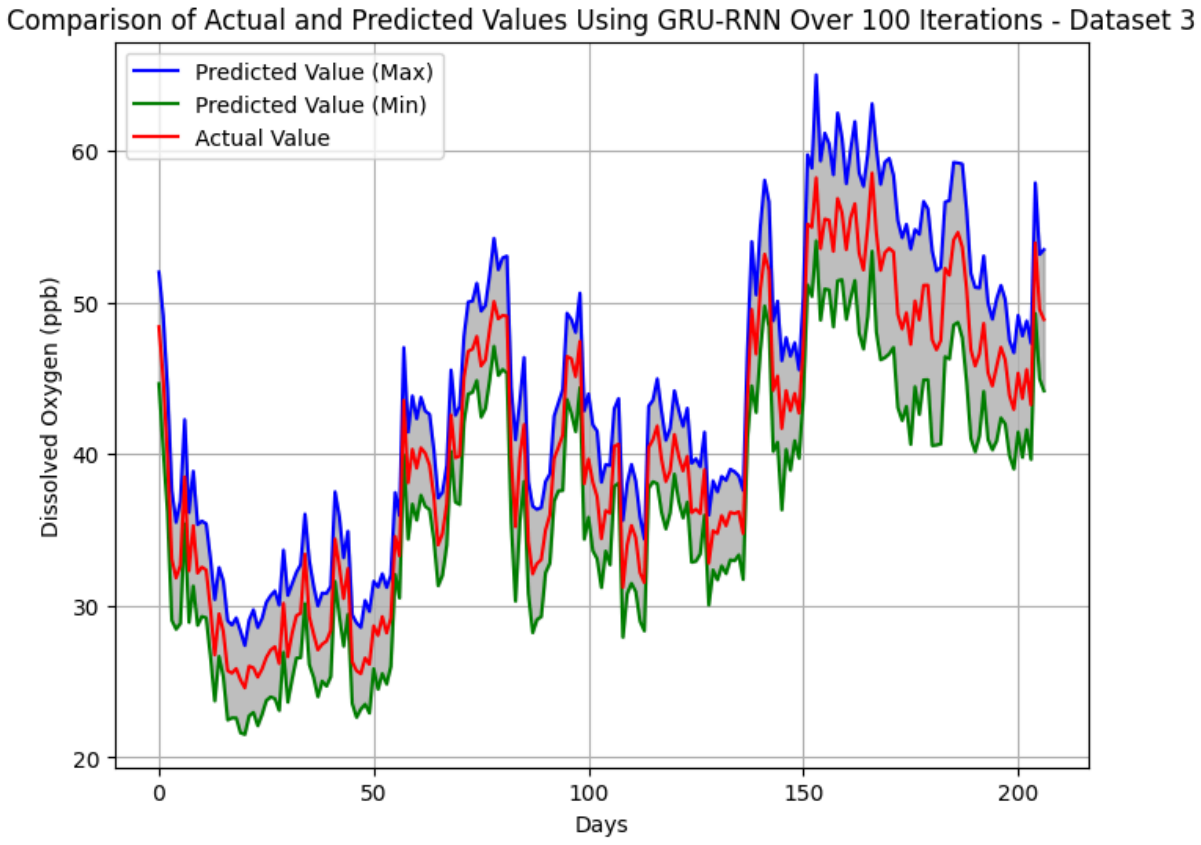
Viz

In [ ]:
```
from re import X
def shade_plot(temp, x_valid='0', line='MAE', predict=''):
  import matplotlib.pyplot as plt

  if predict == '':
    redline = f'Mean {line}'
    axis = 'Epochs'
  else:
    redline = 'Actual Value'
    axis = 'Days'

  #SORT
  temp = temp.apply(lambda x: x.sort_values().reset_index(drop=True))

  #CUT 2.5%
  n = int(len(temp) * 0.025)
  temp = temp.iloc[n:-n]

  batas_atas = temp.max().tolist()
  batas_bawah = temp.min().tolist()
  rata = temp.mean().tolist()

  # Replace these lists with your actual data
  # Dataset 1
  x1 = [i for i in range(0,len(batas_atas))]
  y1 = batas_atas

  # Dataset 2
  x2 = [i for i in range(0,len(batas_bawah))]
  y2 = batas_bawah

  if(x_valid=='0'):
    # Dataset 3
    x3 = [i for i in range(0,len(rata))]
    y3 = rata
  else:
    x3 = [i for i in range(0,len(x_valid))]
    y3 = x_valid

  # Create the plot
  plt.figure(figsize=(8, 6))

  # Plot the first linear dataset
  plt.plot(x1, y1, label=f'Predicted Value (Max)', color='blue')
```

```
# Plot the second linear dataset
plt.plot(x2, y2, label=f'Predicted Value (Min)', color='green')

plt.plot(x3, y3, label=f'{redline}', color='red')

# Create a mask for the shaded area where y1 >= y2
mask = [y1_val >= y2_val for y1_val, y2_val in zip(y1, y2)]

# Shade the area between the two datasets
plt.fill_between(x1, y1, y2, where=mask, interpolate=True, color='gray', alpha=0.5)

# Add labels and legend
plt.xlabel(f'{axis}')
plt.ylabel('Dissolved Oxygen (ppb)')
plt.title(f'Comparison of Actual and Predicted Values Using {line} Over 100 Iterations - Data
plt.legend()
plt.savefig(f"Predicted_Actual_Value_Dataset3_{line}.pdf", dpi=300)
files.download(f"Predicted_Actual_Value_Dataset3_{line}.pdf")
# Show the plot
plt.grid(True)
plt.show()
```

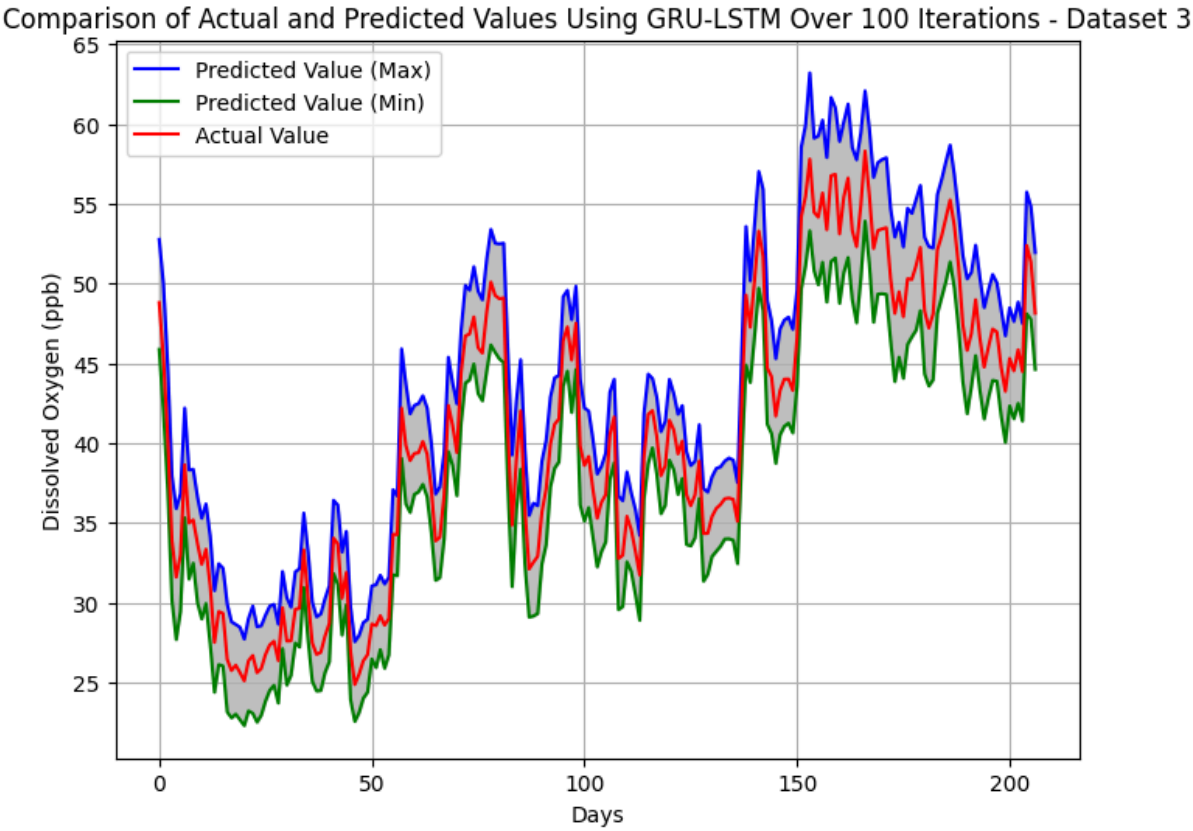
GRU-RNN

In [ ]:
```
temp = pd.read_excel('GRU-RNN_PREDICT.xlsx')
shade_plot(temp, line='GRU-RNN', predict='YES')
```



GRU-LSTM

In [ ]:
```
temp = pd.read_excel('GRU-LSTM_PREDICT.xlsx')
shade_plot(temp, line='GRU-LSTM', predict='YES')
```


Comparison of Actual and Predicted Values Using GRU-LSTM Over 100 Iterations - Dataset 3
Predicted Value (Max)
Predicted Value (Min)
Actual Value
Dissolved Oxygen (ppb)
Days
65
60
55
50
45
40
35
30
25
0
50
100
150
200

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import files
```

DONWLOAD DATA

In [ ]:
```
!gdown 1_bfo8AtzgY1aTlBMkCQTeXZJgvpya_jy
!gdown 1Yfwmvc33LCqBRolk0cht_U0z8Dm2e94r
!gdown 1SlDSpcjwyok5cvllKSBIkhmgcEMsniXw
!gdown 1keYxJghjd3PS_RKCG8pAMZ_rKED9bR1I
!gdown 1C-8SpOb3URkXvOfv2MT_8F1BXhG_z6yC
!gdown 1lWRg123gdALQdFHd8Y7YKBBuFWV9AJz3
!gdown 1ABT_xEk_8wSQ7GAUdb4_zl_P5aqao3AC
!gdown 1KhQuc5NO8DM2lSsQHFMc_Asj8AC_gCix
!gdown 1j0grY7SJMdeOUE8xJ_C8pAfzSLxWn32p
```

In [ ]:
```
!gdown 1u3QM7s9U4X1zwhXoIyMUlUHL9MP2m7lA
!gdown 1n-qg3c_F7CdKRu4J-pvosOPifeOFpyXT
!gdown 1eHWoOj9Hjnqq953qF7GnuOqBHAZO6btU
!gdown 14CvG2kBlZc0iGIN2yiE3IHbYTJp1lcbX
!gdown 11wAUYeSh6ABH5CdguGh9Db8m206b1BHy
!gdown 1Hs82CjFvveFxVVhq7khEN80zw5JgX2YW
!gdown 1Yu32XV9IWjqMmIlD58hccDKvXhfYBRz-
!gdown 18fIbFzz1IDCieyI3AzCOw51TLMD4jNE_
!gdown 1rvpQ9cuZ2oz_vJfR71wbAAiT2WAvkOaN
```

In [ ]:
```
!gdown 1pFi3Ak4tDjpP_bqFlq-TpIsY3BdN9iKB
!gdown 1ZMIRCHq8ixaqIbMNBKezMbJF_1HxufSm
!gdown 1ZWkYruyNyXxzfen4nX35EGqMhU8RqfCm
!gdown 1-R3HgWubX2WPGExnivgaDkngXCC4gwwI
!gdown 1jZRd9ZPvhqdMoqkrOSWAZF-VTnOpGVBT
!gdown 1u9v7m9DhtGDBxMRC5nIpeP2WdxDQzYCv
!gdown 1heElB_3Y-EfZ3NvRQyJfEaVtjKYbT7IJ
!gdown 1tzPKVzJQWdvOb3VlcaFb2Cl-OlbUzaoj
!gdown 1yfnad4_9xfyV07ChCwNY2QVO6CMwwY8C
```

In [ ]:
```
!gdown 1LgzOypyseSxSi6npx4uqU06qQlYiLaBW
!gdown 1UPlXMWGCECg95bikCPMQL4S74ulyDiJf
!gdown 1WC68Tph806l4Lh9M8OoC-k0mbYTCdmvg
!gdown 1FtcLm4T-Kf7NFUtaJFRt4G0XB1iHMLt3
!gdown 1_wKEcb1Y-4Ixm7kCeTwvSDPKmXDGBLMv
!gdown 19xKB1I7u4y-OeE89eKRBX6w7aVpT_ZTA
!gdown 1Wq8TYAz_HUf7ixW8k3wvALsxjnGwbMy2
!gdown 1QyL2vIpZEbo84zcZxdB0W4B3_J-CzLvG
!gdown 1hraImwtrxdcxxwPsLAiwtqXNxCc-z4Kz
```

Metrik Evaluation

MAE

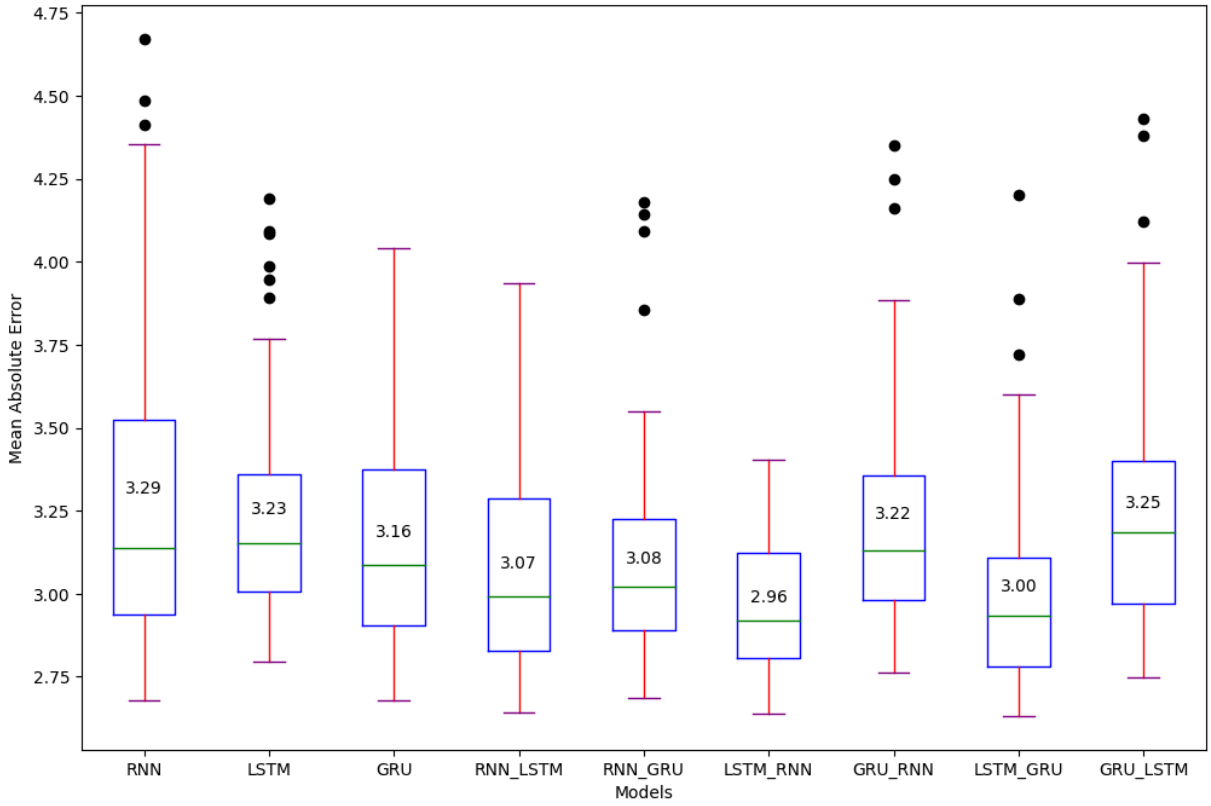
In [ ]:
```
df1 =  pd.read_excel('/content/RNN_FORECASTMae.xlsx')
df2 =  pd.read_excel('/content/LSTM_FORECASTMae.xlsx')
df3 =  pd.read_excel('/content/GRU_FORECASTMae.xlsx')
df4 =  pd.read_excel('/content/RNN-LSTM_FORECASTMae.xlsx')
df5 =  pd.read_excel('/content/RNN-GRU_FORECASTMae.xlsx')
df6 =  pd.read_excel('/content/LSTM-RNN_FORECASTMae.xlsx')
df7 =  pd.read_excel('/content/GRU-RNN_FORECASTMae.xlsx')
df8 =  pd.read_excel('/content/LSTM-GRU_FORECASTMae.xlsx')
df9 =  pd.read_excel('/content/GRU-LSTM_FORECASTMae.xlsx')
```

In [ ]:
```
df = pd.DataFrame()
df['RNN'] =  df1[0]
```

```
df['LSTM'] =  df2[0]
df['GRU'] =  df3[0]
df['RNN_LSTM'] =  df4[0]
df['RNN_GRU'] =  df5[0]
df['LSTM_RNN'] =  df6[0]
df['GRU_RNN'] =  df7[0]
df['LSTM_GRU'] =  df8[0]
df['GRU_LSTM'] =  df9[0]

# n = int(len(df) * (20 / 100))
n = int(len(df) * (2.5 / 100))
df = df.iloc[n:-n]
```

In [ ]:

```
df = pd.DataFrame()
df['RNN'] =  sorted(df1[0])
df['LSTM'] =  sorted(df2[0])
df['GRU'] =  sorted(df3[0])
df['RNN_LSTM'] =  sorted(df4[0])
df['RNN_GRU'] =  sorted(df5[0])
df['LSTM_RNN'] = sorted(df6[0])
df['GRU_RNN'] =  sorted(df7[0])
df['LSTM_GRU'] = sorted(df8[0])
df['GRU_LSTM'] = sorted(df9[0])


# n = int(len(df)*0.15)
n = int(len(df) * 0.025)
df = df[:].iloc[n:-n]
data = df[:]

box_color = 'blue'
whisker_color = 'red'
median_color = 'green'
flier_color = 'black'
cap_color = 'purple'

fig, ax = plt.subplots(figsize=(12, 8))

plt.boxplot(data.values, labels=data.columns,
            boxprops=dict(color=box_color),
            whiskerprops=dict(color=whisker_color),
            capprops=dict(color=cap_color),
            flierprops=dict(markerfacecolor=flier_color, marker='o', markersize=6),
            medianprops=dict(color=median_color))

for i, column in enumerate(data.columns):
    mean = np.mean(data[column])
    plt.text(i + 1, mean, f'{mean:.2f}', ha='center', va='bottom', color='black')

# plt.title('Predict with Boxplot')
plt.xlabel('Models')
plt.ylabel('Mean Absolute Error')
plt.savefig("Comparison_MAE_Dataset1.pdf", dpi=300)
files.download("Comparison_MAE_Dataset1.pdf")

plt.show()
```

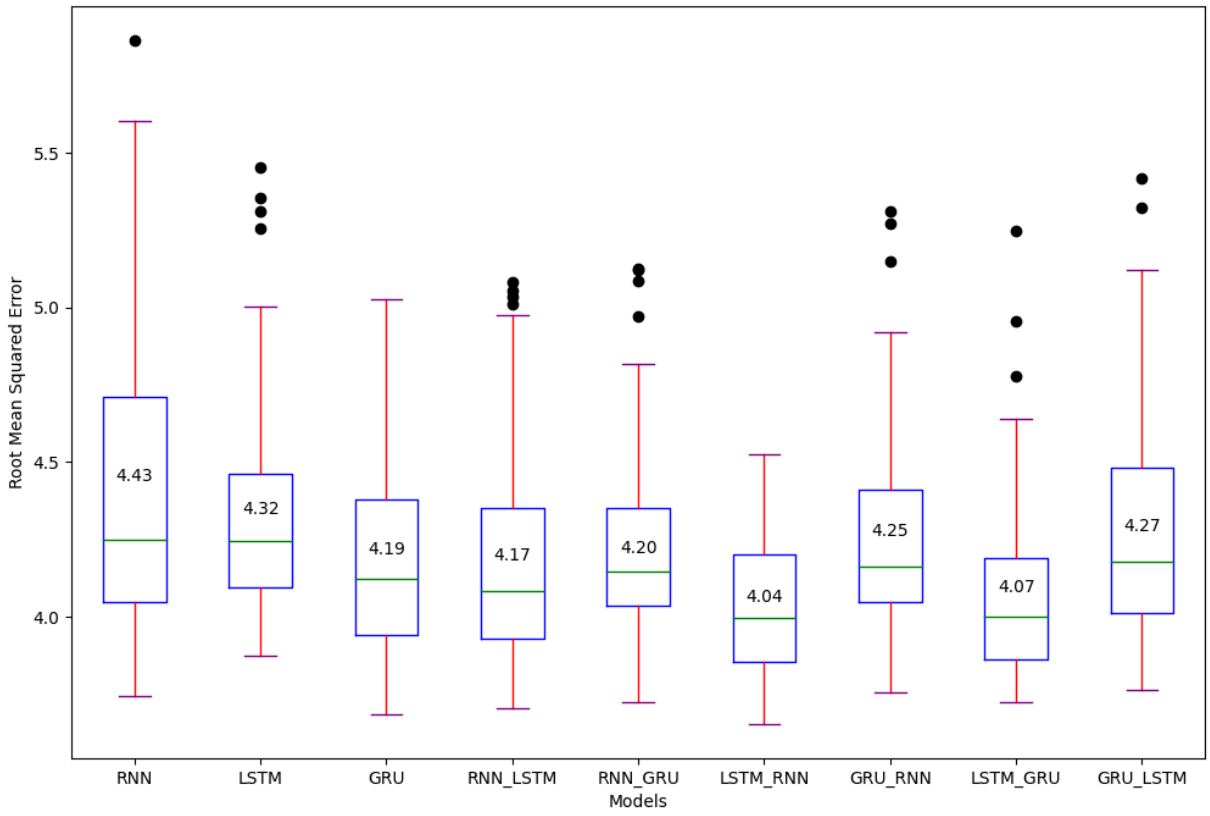


RMSE

In [ ]:
```
df1 =  pd.read_excel('/content/RNN_FORECASTRMse.xlsx')
df2 =  pd.read_excel('/content/LSTM_FORECASTRMse.xlsx')
df3 =  pd.read_excel('/content/GRU_FORECASTRMse.xlsx')
df4 =  pd.read_excel('/content/RNN-LSTM_FORECASTRMse.xlsx')
df5 =  pd.read_excel('/content/RNN-GRU_FORECASTRMse.xlsx')
df6 =  pd.read_excel('/content/LSTM-RNN_FORECASTRMse.xlsx')
df7 =  pd.read_excel('/content/GRU-RNN_FORECASTRMse.xlsx')
df8 =  pd.read_excel('/content/LSTM-GRU_FORECASTRMse.xlsx')
df9 =  pd.read_excel('/content/GRU-LSTM_FORECASTRMse.xlsx')
df = pd.DataFrame()
df['RNN'] =  df1[0]
df['LSTM'] =  df2[0]
df['GRU'] =  df3[0]
df['RNN_LSTM'] =  df4[0]
df['RNN_GRU'] =  df5[0]
df['LSTM_RNN'] =  df6[0]
df['GRU_RNN'] =  df7[0]
df['LSTM_GRU'] =  df8[0]
df['GRU_LSTM'] =  df9[0]

# n = int(len(df) * (20 / 100))
n = int(len(df) * (2.5 / 100))
df = df.iloc[n:-n]
```

In [ ]:
```
df = pd.DataFrame()
df['RNN'] =  sorted(df1[0])
df['LSTM'] =  sorted(df2[0])
df['GRU'] =  sorted(df3[0])
df['RNN_LSTM'] =  sorted(df4[0])
df['RNN_GRU'] =  sorted(df5[0])
df['LSTM_RNN'] = sorted(df6[0])
df['GRU_RNN'] =  sorted(df7[0])
df['LSTM_GRU'] = sorted(df8[0])
df['GRU_LSTM'] = sorted(df9[0])


# n = int(len(df)*0.15)
n = int(len(df) * 0.025)
```

```
df = df[:].iloc[n:-n]
data = df[:]

box_color = 'blue'
whisker_color = 'red'
median_color = 'green'
flier_color = 'black'
cap_color = 'purple'

fig, ax = plt.subplots(figsize=(12, 8))

plt.boxplot(data.values, labels=data.columns,
            boxprops=dict(color=box_color),
            whiskerprops=dict(color=whisker_color),
            capprops=dict(color=cap_color),
            flierprops=dict(markerfacecolor=flier_color, marker='o', markersize=6),
            medianprops=dict(color=median_color))

for i, column in enumerate(data.columns):
    mean = np.mean(data[column])
    plt.text(i + 1, mean, f'{mean:.2f}', ha='center', va='bottom', color='black')

# plt.title('Predict with Boxplot')
plt.xlabel('Models')
plt.ylabel('Root Mean Squared Error')
plt.savefig("Comparison_RMSE_Dataset1.pdf", dpi=300)
files.download("Comparison_RMSE_Dataset1.pdf")

plt.show()
```



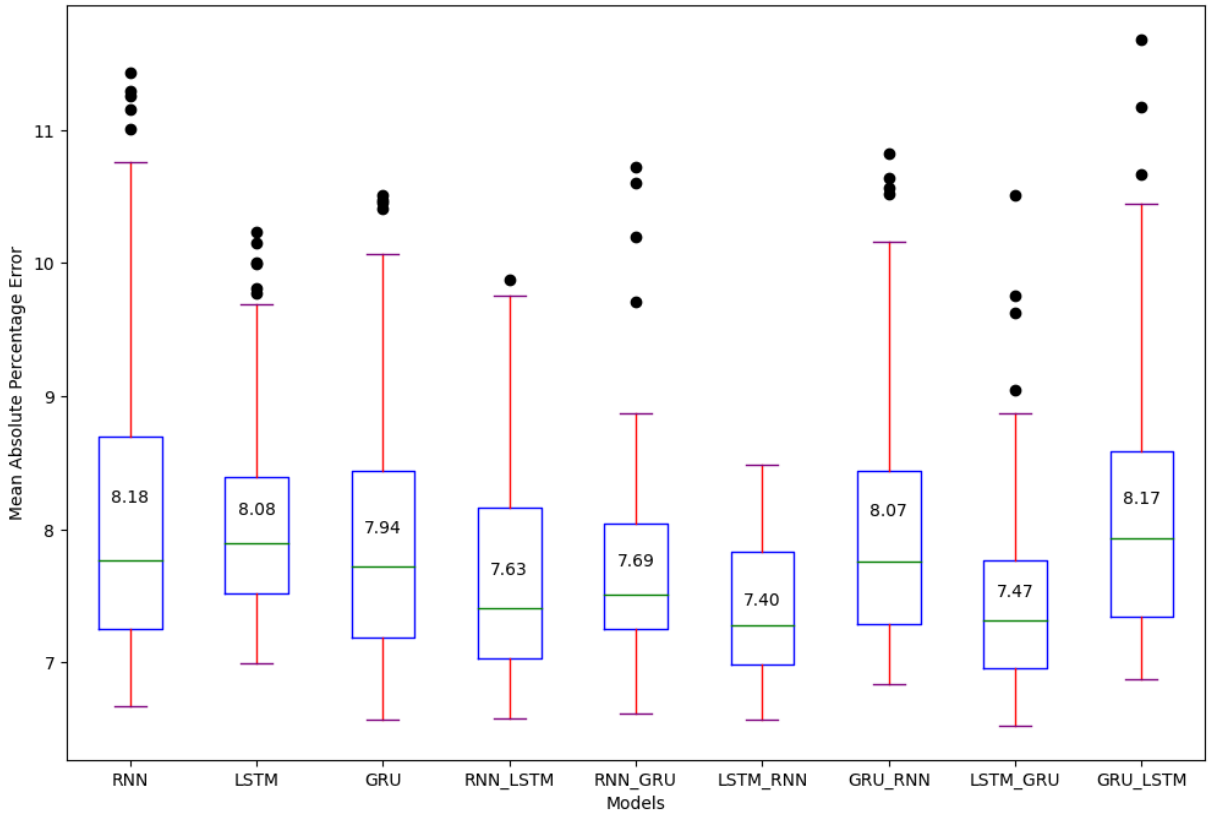
MAPE

In [ ]:
```
df1 =  pd.read_excel('/content/RNN_FORECASTMpe.xlsx')
df2 =  pd.read_excel('/content/LSTM_FORECASTMpe.xlsx')
df3 =  pd.read_excel('/content/GRU_FORECASTMpe.xlsx')
df4 =  pd.read_excel('/content/RNN-LSTM_FORECASTMpe.xlsx')
df5 =  pd.read_excel('/content/RNN-GRU_FORECASTMpe.xlsx')
df6 =  pd.read_excel('/content/LSTM-RNN_FORECASTMpe.xlsx')
df7 =  pd.read_excel('/content/GRU-RNN_FORECASTMpe.xlsx')
```

```
df8 =  pd.read_excel('/content/LSTM-GRU_FORECASTMpe.xlsx')
df9 =  pd.read_excel('/content/GRU-LSTM_FORECASTMpe.xlsx')
df = pd.DataFrame()
df['RNN'] =  df1[0]
df['LSTM'] =  df2[0]
df['GRU'] =  df3[0]
df['RNN_LSTM'] =  df4[0]
df['RNN_GRU'] =  df5[0]
df['LSTM_RNN'] =  df6[0]
df['GRU_RNN'] =  df7[0]
df['LSTM_GRU'] =  df8[0]
df['GRU_LSTM'] =  df9[0]

# n = int(len(df) * (20 / 100))
n = int(len(df) * (2.5 / 100))
df = df.iloc[n:-n]
```

In [ ]:

```
df = pd.DataFrame()
df['RNN'] =  sorted(df1[0])
df['LSTM'] =  sorted(df2[0])
df['GRU'] =  sorted(df3[0])
df['RNN_LSTM'] =  sorted(df4[0])
df['RNN_GRU'] =  sorted(df5[0])
df['LSTM_RNN'] = sorted(df6[0])
df['GRU_RNN'] =  sorted(df7[0])
df['LSTM_GRU'] = sorted(df8[0])
df['GRU_LSTM'] = sorted(df9[0])


# n = int(len(df)*0.15)
n = int(len(df) * 0.025)
df = df[:].iloc[n:-n]
data = df[:]

box_color = 'blue'
whisker_color = 'red'
median_color = 'green'
flier_color = 'black'
cap_color = 'purple'

fig, ax = plt.subplots(figsize=(12, 8))

plt.boxplot(data.values, labels=data.columns,
            boxprops=dict(color=box_color),
            whiskerprops=dict(color=whisker_color),
            capprops=dict(color=cap_color),
            flierprops=dict(markerfacecolor=flier_color, marker='o', markersize=6),
            medianprops=dict(color=median_color))

for i, column in enumerate(data.columns):
    mean = np.mean(data[column])
    plt.text(i + 1, mean, f'{mean:.2f}', ha='center', va='bottom', color='black')

# plt.title('Predict with Boxplot')
plt.xlabel('Models')
plt.ylabel('Mean Absolute Percentage Error')
plt.savefig("Comparison_MAPE_Dataset1.pdf", dpi=300)
files.download("Comparison_MAPE_Dataset1.pdf")

plt.show()
```



Time Comparison

In [ ]:
```
df1 =  pd.read_excel('/content/RNN_TIME.xlsx')
df2 =  pd.read_excel('/content/LSTM_TIME.xlsx')
df3 =  pd.read_excel('/content/GRU_TIME.xlsx')
df4 =  pd.read_excel('/content/RNN-LSTM_TIME.xlsx')
df5 =  pd.read_excel('/content/RNN-GRU_TIME.xlsx')
df6 =  pd.read_excel('/content/LSTM-RNN_TIME.xlsx')
df7 =  pd.read_excel('/content/GRU-RNN_TIME.xlsx')
df8 =  pd.read_excel('/content/LSTM-GRU_TIME.xlsx')
df9 =  pd.read_excel('/content/GRU-LSTM_TIME.xlsx')
df = pd.DataFrame()
df['RNN'] =  df1[0]
df['LSTM'] =  df2[0]
df['GRU'] =  df3[0]
df['RNN_LSTM'] =  df4[0]
df['RNN_GRU'] =  df5[0]
df['LSTM_RNN'] =  df6[0]
df['GRU_RNN'] =  df7[0]
df['LSTM_GRU'] =  df8[0]
df['GRU_LSTM'] =  df9[0]
```

In [ ]:
```
# Calculate the average time for each model
average_times = df.mean()

# Create a bar chart
plt.figure(figsize=(10, 6))
average_times.plot(kind='bar', color='skyblue')
plt.title('Comparison of Computational Time Across Different Models')
plt.xlabel('Models')
plt.ylabel('Avarage Time (Seconds)')
plt.xticks(rotation=45)
plt.tight_layout()

# Display the average times on top of the bars
for i, v in enumerate(average_times):
    plt.text(i, v, f"{v:.2f}", ha='center', va='bottom')

plt.savefig("Comparison_Time_Dataset1.pdf", dpi=300)
```

```
files.download("Comparison_Time_Dataset1.pdf")
plt.show()
```