In [1]:
```
# Variational Autoencoder for Time Series Analysis
#
# Igor Mol <igor.mol@makes.ai>
#
# The code that follows implements a Variational Autoencoder (VAE) for recon
# tructing time series data. This VAE architecture consists of an encoder, a
# reparameterization trick, and a decoder. The encoder processes the input t
# series, transforming it into a condensed representation in a latent space.
#     The reparameterization trick introduces randomness to navigate uncerta
# in this latent space. The decoder then reconstructs the time series from t
# latent representation. The VAE's performance is evaluated through a loss f
# ction, comprising cross-entropy loss and Kullback-Leibler divergence. Duri
# training, the model refines its understanding of the time series through
# stochastic gradient descent. The encoded series reveals latent insights, w
# the reconstructed series is brought back to the original scale. This proce
# exemplifies the VAE's ability to understand and reconstruct time series da
# bridging classical principles with modern neural network techniques.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Lambda
from tensorflow.keras import backend as K
from tensorflow.keras.losses import binary_crossentropy
from tensorflow.keras.optimizers import Adam
from scipy.stats import norm

# Define a class for the Variational Autoencoder (VAE)
class VAE:
    def __init__(self, original_dim, intermediate_dim, latent_dim):
        # Initialize VAE with dimensions for input, intermediate layer, and
        self.original_dim = original_dim
        self.intermediate_dim = intermediate_dim
        self.latent_dim = latent_dim
        # Build the VAE model
        self.model = self.build_model()

    def build_model(self):
        # Encoder architecture
        inputs = Input(shape=(self.original_dim,), name='encoder_input')
        h = Dense(self.intermediate_dim, activation='relu')(inputs)
        z_mean = Dense(self.latent_dim, name='z_mean')(h)
        z_log_var = Dense(self.latent_dim, name='z_log_var')(h)

        # Reparameterization trick
        def sampling(args):
            z_mean, z_log_var = args
            batch = K.shape(z_mean)[0]
            dim = K.int_shape(z_mean)[1]
            epsilon = K.random_normal(shape=(batch, dim))
            return z_mean + K.exp(0.5 * z_log_var) * epsilon

        z = Lambda(sampling, output_shape=(self.latent_dim,), name='z')([z_m

        # Decoder architecture
```

```
        decoder_h = Dense(self.intermediate_dim, activation='relu')
        decoder_mean = Dense(self.original_dim, activation='sigmoid')
        h_decoded = decoder_h(z)
        x_decoded_mean = decoder_mean(h_decoded)

        # Overall VAE model
        vae = Model(inputs, x_decoded_mean)

        # VAE loss and custom layer
        xent_loss = self.original_dim * binary_crossentropy(inputs, x_decode
        kl_loss = -0.5 * K.sum(1 + z_log_var - K.square(z_mean) - K.exp(z_lo
        vae_loss = K.mean(xent_loss + kl_loss)

        # Add the loss to the model and compile
        vae.add_loss(vae_loss)
        vae.compile(optimizer=Adam())

        return vae

    def train(self, data, epochs=50, batch_size=32, validation_split=0.1):
        # Train the VAE model on the provided data
        self.model.fit(data, epochs=epochs, batch_size=batch_size, validatio

    def predict(self, data):
        # Use the trained model to predict reconstructed data
        return self.model.predict(data)

# Function to normalize data using Min-Max scaling
def normalize_data(data):
    scaler = MinMaxScaler()
    normalized_data = scaler.fit_transform(data.reshape(-1, 1))
    return normalized_data, scaler

# Function to denormalize data
def denormalize_data(normalized_data, scaler):
    return scaler.inverse_transform(normalized_data).flatten()

# Main function
def main():
    # Load the time-series data from a CSV file
    file_path = "/Users/igormol/Desktop/time_series_data.csv"
    df = pd.read_csv(file_path)
    df['Date'] = pd.to_datetime(df['Date'])
    df = df.sort_values(by='Date')

    # Normalize the 'Value' column
    values = df['Value'].values
    normalized_values, scaler = normalize_data(values)

    # Create a VAE model
    # "In our sleep, pain which cannot forget falls drop by drop upon the he
    vae_model = VAE(original_dim=1, intermediate_dim=64, latent_dim=2)

    # Train the VAE on the normalized time series
    X_train = normalized_values.reshape(-1, 1)
    vae_model.train(X_train)

    # Encode and decode the time series
    encoded_series = vae_model.predict(X_train)
    decoded_series = denormalize_data(encoded_series, scaler)
```

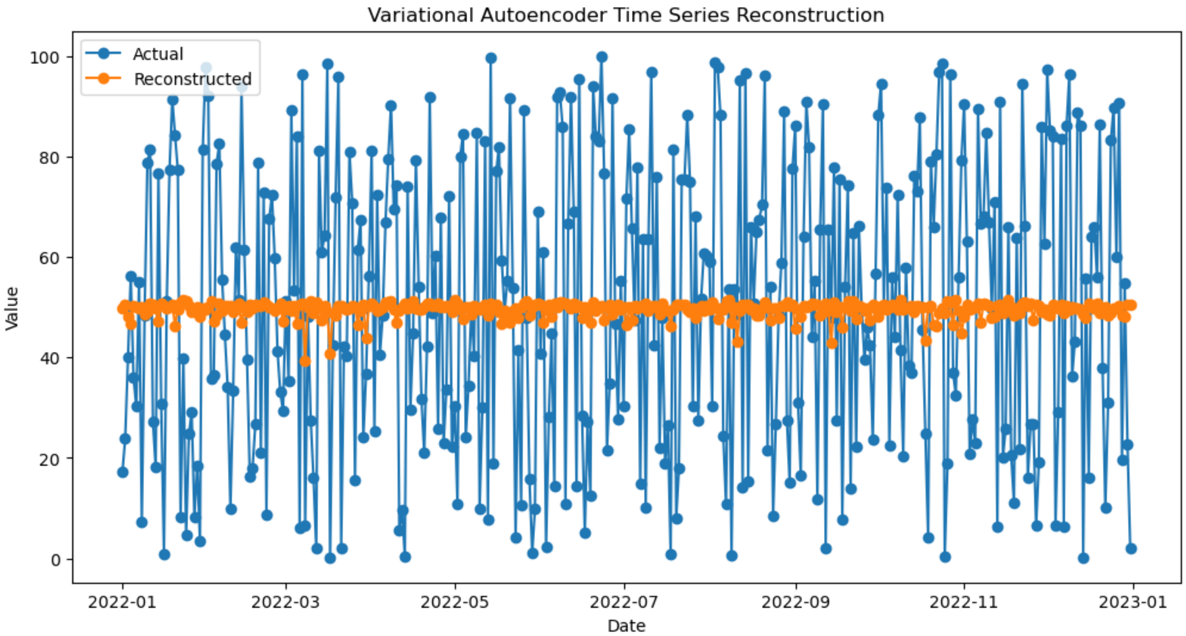
```
    decoded_series = denormalize_data(encoded_series, scaler)

    # Create a DataFrame for the results
    results = pd.DataFrame({'Date': df['Date'], 'Actual': values, 'Reconstru

    # Print the results in a formatted table
    print(results)

    # Plot the actual and reconstructed time series
    plt.figure(figsize=(12, 6))
    plt.plot(df['Date'], values, label='Actual', marker='o')
    plt.plot(df['Date'], decoded_series, label='Reconstructed', marker='o')
    plt.title('Variational Autoencoder Time Series Reconstruction')
    plt.xlabel('Date')
    plt.ylabel('Value')
    plt.legend()
    plt.show()

# Run the main function if the script is executed directly
if __name__ == "__main__":
    main()
```

```
Epoch 1/50
11/11 [==============================] - 3s 43ms/step - loss: 0.7157 - val_
loss: 0.7019
Epoch 2/50
11/11 [==============================] - 0s 10ms/step - loss: 0.6964 - val_
loss: 0.7008
Epoch 3/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6946 - val_l
oss: 0.6633
Epoch 4/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6952 - val_l
oss: 0.7068
Epoch 5/50
11/11 [==============================] - 0s 11ms/step - loss: 0.6950 - val_
loss: 0.6766
Epoch 6/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6952 - val_l
oss: 0.6925
Epoch 7/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6941 - val_l
oss: 0.7019
Epoch 8/50
11/11 [==============================] - 0s 11ms/step - loss: 0.6974 - val_
loss: 0.6886
Epoch 9/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6947 - val_l
oss: 0.6959
Epoch 10/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6939 - val_l
oss: 0.6897
Epoch 11/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6947 - val_l
oss: 0.6900
Epoch 12/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6946 - val_l
oss: 0.7012
Epoch 13/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6954 - val_l
```

```
oss: 0.6901
Epoch 14/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6952 - val_l
oss: 0.6899
Epoch 15/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6922 - val_l
oss: 0.6926
Epoch 16/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6945 - val_l
oss: 0.6956
Epoch 17/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6936 - val_l
oss: 0.7010
Epoch 18/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6923 - val_l
oss: 0.6974
Epoch 19/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6932 - val_l
oss: 0.6883
Epoch 20/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6937 - val_l
oss: 0.7074
Epoch 21/50
11/11 [==============================] - 0s 10ms/step - loss: 0.6943 - val_
loss: 0.6864
Epoch 22/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6956 - val_l
oss: 0.7010
Epoch 23/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6935 - val_l
oss: 0.6887
Epoch 24/50
11/11 [==============================] - 0s 10ms/step - loss: 0.6926 - val_
loss: 0.6915
Epoch 25/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6940 - val_l
oss: 0.6917
Epoch 26/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6953 - val_l
oss: 0.6971
Epoch 27/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6935 - val_l
oss: 0.6982
Epoch 28/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6936 - val_l
oss: 0.6909
Epoch 29/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6930 - val_l
oss: 0.6940
Epoch 30/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6932 - val_l
oss: 0.6954
Epoch 31/50
11/11 [==============================] - 0s 10ms/step - loss: 0.6933 - val_
loss: 0.6958
Epoch 32/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6957 - val_l
oss: 0.6986
Epoch 33/50
```

```
11/11 [==============================] - 0s 10ms/step - loss: 0.6926 - val_
loss: 0.6998
Epoch 34/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6934 - val_l
oss: 0.6946
Epoch 35/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6926 - val_l
oss: 0.7024
Epoch 36/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6909 - val_l
oss: 0.6974
Epoch 37/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6938 - val_l
oss: 0.6941
Epoch 38/50
11/11 [==============================] - 0s 10ms/step - loss: 0.6938 - val_
loss: 0.7045
Epoch 39/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6947 - val_l
oss: 0.6938
Epoch 40/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6924 - val_l
oss: 0.6904
Epoch 41/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6944 - val_l
oss: 0.7009
Epoch 42/50
11/11 [==============================] - 0s 8ms/step - loss: 0.6938 - val_l
oss: 0.6964
Epoch 43/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6937 - val_l
oss: 0.6963
Epoch 44/50
11/11 [==============================] - 0s 20ms/step - loss: 0.6938 - val_
loss: 0.6907
Epoch 45/50
11/11 [==============================] - 0s 19ms/step - loss: 0.6945 - val_
loss: 0.6930
Epoch 46/50
11/11 [==============================] - 0s 13ms/step - loss: 0.6931 - val_
loss: 0.6918
Epoch 47/50
11/11 [==============================] - 0s 15ms/step - loss: 0.6934 - val_
loss: 0.6916
Epoch 48/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6920 - val_l
oss: 0.6945
Epoch 49/50
11/11 [==============================] - 0s 10ms/step - loss: 0.6928 - val_
loss: 0.6938
Epoch 50/50
11/11 [==============================] - 0s 10ms/step - loss: 0.6925 - val_
loss: 0.6939
12/12 [==============================] - 0s 3ms/step
         Date     Actual  Reconstructed
0  2022-01-01  17.205673      49.698822
1  2022-01-02  23.882583      50.532482
2  2022-01-03  39.984362      48.223988
3  2022-01-04  56.131855      46.621735
```

```
4   2022-01-05  36.124378      50.266644
..         ...        ...            ...
360 2022-12-27  90.744706      50.225544
361 2022-12-28  19.675608      48.439419
362 2022-12-29  54.734194      48.054989
363 2022-12-30  22.710878      50.522205
364 2022-12-31   1.970247      50.609833

[365 rows x 3 columns]
```



In [2]:
```
# Import necessary libraries
#
# Igor Mol <igor.mol@makes.ai>
#
# The provided Python code implements a Restricted Boltzmann Machine (RBM) f
# time series data generation. It begins by defining a TimeSeriesData class
# ponsible for preprocessing the input time series data. This involves conve
# the 'Date' column to datetime, scaling the 'Value' column using MinMaxScal
# and creating sequences of data to train the RBM. The RBM is implemented in
# RBM class, which has methods for training the model using contrastive dive
# gence and generating samples. The training process involves iterating thro
# epochs, shuffling the data, and updating weights and biases based on the d
# ference between positive and negative associations. The generated samples
# then denormalized using the scaler. The main function orchestrates the ent
# process, loading the time series data, preprocessing it, training the RBM,
# generating samples, denormalizing them, and finally, creating and printing
# table that compares the actual and generated time series sequences.
#     The primary goal of this code is to showcase the use of RBMs for time
# ries data generation. RBMs are a type of unsupervised learning model, and
# this context, they are used to learn the underlying patterns and structure
# the time series data. The generated samples are denormalized and presented
# tabulated form, allowing for a direct visual comparison between the actual
# generated sequences. The training process involves iteratively adjusting t
# RBM's parameters to capture the statistical dependencies within the input
# ultimately enabling the model to generate synthetic time series sequences
# exhibit similar patterns to the original data. This approach is particular
# useful for tasks such as anomaly detection, data augmentation, or creating
# realistic synthetic data for testing machine learning models.
#     In more detail, the TimeSeriesData class handles the initial loading a
# preprocessing of time series data. It reads the data from a CSV file, tran
# forms the 'Date' column to datetime, scales the 'Value' column using MinMa
# Scaler, and creates sequences of data suitable for training the RBM. The R
# is implemented in the RBM class, where the sigmoid activation function is
# fined along with methods for sampling hidden and visible layers. The train
# process involves both positive and negative phases, where hidden layer sta
# are sampled based on visible layer states, and vice versa. The weights and
# biases are updated using contrastive divergence, a technique specific to R
#     Finally, the main function orchestrates the entire workflow, from load
# the data to training the RBM, generating denormalized samples, and creatin
# table that compares the actual and generated time series sequences using t
# 'tabulate' library for a more readable output.

import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tabulate import tabulate

# Define the TimeSeriesData class for data preprocessing
class TimeSeriesData:
    def __init__(self, file_path):
        # Initialize with file path, sequence length, MinMaxScaler, and plac
        self.time_series_df = pd.read_csv(file_path)
        self.sequence_length = 10
        self.scaler = MinMaxScaler()
        self.X_train = None
        self.X_test = None

    # Preprocess the time series data
```

```
    def preprocess_data(self):
        # Convert 'Date' column to datetime, extract 'Value' column, and sca
        self.time_series_df['Date'] = pd.to_datetime(self.time_series_df['Da
        values = self.time_series_df['Value'].values.reshape(-1, 1)
        values_scaled = self.scaler.fit_transform(values)

        # Create sequences for training the RBM
        X = []
        for i in range(len(values_scaled) - self.sequence_length):
            X.append(values_scaled[i:i + self.sequence_length].flatten())
        X = np.array(X)

        # Split the data into training and testing sets
        train_size = int(len(values_scaled) * 0.8)
        self.X_train, self.X_test = X[:train_size], X[train_size:]

# Define the RBM class for Restricted Boltzmann Machine implementation
class RBM:
    def __init__(self, visible_size, hidden_size):
        # Initialize RBM parameters: weights, visible bias, and hidden bias
        self.visible_size = visible_size
        self.hidden_size = hidden_size
        self.weights = np.random.randn(visible_size, hidden_size)
        self.visible_bias = np.zeros((1, visible_size))
        self.hidden_bias = np.zeros((1, hidden_size))

    # Sigmoid activation function
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    # Sample hidden layer probabilities and states
    def sample_hidden(self, visible_probs):
        hidden_probs = self.sigmoid(np.dot(visible_probs, self.weights) + se
        hidden_states = np.random.binomial(1, hidden_probs)
        return hidden_probs, hidden_states

    # Sample visible layer probabilities and states
    def sample_visible(self, hidden_probs):
        visible_probs = self.sigmoid(np.dot(hidden_probs, self.weights.T) +
        visible_states = np.random.binomial(1, visible_probs)
        return visible_probs, visible_states

    # Train the RBM using contrastive divergence
    def train(self, data, learning_rate=0.01, epochs=50, batch_size=32):
        num_samples = data.shape[0]

        for epoch in range(epochs):
            np.random.shuffle(data)

            for i in range(0, num_samples, batch_size):
                batch_data = data[i:i + batch_size]

                # Positive phase: Sample hidden layer states based on visibl
                positive_hidden_probs, positive_hidden_states = self.sample_
                positive_associations = np.dot(batch_data.T, positive_hidder

                # Negative phase: Sample visible and hidden layer states ite
                negative_visible_probs, negative_visible_states = self.sampl
                negative_hidden_probs, negative_hidden_states = self.sample_
```

```
                negative_associations = np.dot(negative_visible_states.T, ne

                # Update weights and biases based on contrastive divergence
                self.weights += learning_rate * (positive_associations - neg
                self.visible_bias += learning_rate * np.mean(batch_data - ne
                self.hidden_bias += learning_rate * np.mean(positive_hidden_

    # Generate samples using the trained RBM
    def generate_samples(self, num_samples):
        samples = np.random.rand(num_samples, self.visible_size)
        hidden_probs, _ = self.sample_hidden(samples)
        visible_probs, _ = self.sample_visible(hidden_probs)
        return visible_probs

# Function to create a tabulated view of actual and predicted time series se
def create_table(actual, predicted):
    table_data = {'Actual': actual.flatten(), 'Predicted': predicted.flatter
    table = pd.DataFrame(table_data)
    return tabulate(table, headers='keys', tablefmt='pretty', showindex=Fals

# Main function
def main():
    # File path for time series data
    file_path = "/Users/igormol/Desktop/time_series_data.csv"

    # Create an instance of TimeSeriesData and preprocess the data
    time_series_data = TimeSeriesData(file_path)
    time_series_data.preprocess_data()

    # Set visible and hidden layer sizes for RBM
    visible_size = time_series_data.X_train.shape[1]
    hidden_size = 50

    # Create an instance of RBM and train it on the preprocessed data
    rbm = RBM(visible_size, hidden_size)
    rbm.train(time_series_data.X_train, epochs=50, batch_size=32)

    # Generate samples using the trained RBM
    num_samples = time_series_data.X_test.shape[0]
    generated_samples = rbm.generate_samples(num_samples)

    # Denormalize generated samples
    generated_samples_denormalized = time_series_data.scaler.inverse_transfo

    # Create and print a table comparing actual and generated time series se
    table = create_table(time_series_data.X_test, generated_samples_denormal
    print(table)

# Execute main function if the script is run
if __name__ == "__main__":
    main()
```

```
+------------------------+---------------------+
|         Actual         |      Predicted      |
+------------------------+---------------------+
|   0.7904971722460135   | 90.04540002241326   |
|   0.6598653114510935   | 34.95877036146838   |
|   0.8049260492587612   | 46.02472787030583   |
|    0.968562748740495   | 54.83306181751966   |
|   0.9851986507547381   | 23.21628359464965   |
```

0.0015572327657798462	95.82227989130223
0.18693340210190945	56.86958302679335
0.9642914914648604	26.398887030824792
0.36876471834394303	12.0564343302714
0.32392408225433034	7.813125545039181
0.6598653114510935	96.24964825321756
0.8049260492587612	17.14952629486736
0.968562748740495	23.67876202596187
0.9851986507547381	76.05765890465663
0.0015572327657798462	24.70226608897078
0.18693340210190945	84.13045866794617
0.9642914914648604	79.36771630410956
0.36876471834394303	36.283342517398054
0.32392408225433034	83.87484271005465
0.5592243055017212	41.26441900263548
0.8049260492587612	74.1098428039865
0.968562748740495	90.78505224650227
0.9851986507547381	62.219727128980296
0.0015572327657798462	10.181836291874498
0.18693340210190945	60.49762211274754
0.9642914914648604	12.712566800143296
0.36876471834394303	76.2539916279044
0.32392408225433034	83.0398368446941
0.5592243055017212	3.366807711857917
0.7911760619608509	22.79914202951058
0.968562748740495	11.864270367346219
0.9851986507547381	10.308233528094924
0.0015572327657798462	5.0515213156894445
0.18693340210190945	30.973893325803928
0.9642914914648604	44.77168496812899
0.36876471834394303	73.55650265001202
0.32392408225433034	47.877837388599445
0.5592243055017212	19.733234846228996
0.7911760619608509	39.88167413499159
0.9044907330423345	85.16231817938564
0.9851986507547381	87.5113788483994
0.0015572327657798462	43.340384443674104
0.18693340210190945	13.411260465500062
0.9642914914648604	67.89495116733522
0.36876471834394303	10.17997665799572
0.32392408225433034	36.637116386849165
0.5592243055017212	30.30977156656402
0.7911760619608509	82.5035507044808
0.9044907330423345	35.56946113946434
0.62964552288955	38.6104439799596
0.0015572327657798462	24.70057520111618
0.18693340210190945	61.41449026798346
0.9642914914648604	75.8812250269575
0.36876471834394303	7.669632166656878
0.32392408225433034	4.6468006964430675
0.5592243055017212	80.20667010670469
0.7911760619608509	6.453461446285955
0.9044907330423345	79.69476678397626
0.62964552288955	23.911650407999254
0.20570324453176486	79.64301511078456
0.18693340210190945	93.55370356721313
0.9642914914648604	86.8419540254372
0.36876471834394303	66.43583708252544
0.32392408225433034	65.84804555410611

0.5592243055017212	5.723639679491393
0.7911760619608509	37.91561238862512
0.9044907330423345	29.7057380712961
0.62964552288955	60.89305200307538
0.20570324453176486	68.80373794875455
0.2746799547437327	13.540395431754108
0.9642914914648604	94.03806652990741
0.36876471834394303	50.60071558703616
0.32392408225433034	16.3723522445165
0.5592243055017212	6.218427110666857
0.7911760619608509	66.29178403597739
0.9044907330423345	92.21482095522002
0.62964552288955	88.92310870422547
0.20570324453176486	91.56647099518446
0.2746799547437327	63.22912072413207
0.22882979406431883	44.132843028216925
0.36876471834394303	4.21437333884586
0.32392408225433034	57.26989749448422
0.5592243055017212	46.6385585522187
0.7911760619608509	25.084194583822317
0.9044907330423345	73.48396013276852
0.62964552288955	19.846610650274933
0.20570324453176486	53.7779346404842
0.2746799547437327	31.881559263166032
0.22882979406431883	93.92091549686701
0.8951177076816685	17.643738793107126
0.32392408225433034	5.6029841350499945
0.5592243055017212	21.83606726888328
0.7911760619608509	16.512580700886705
0.9044907330423345	29.217062400183856
0.62964552288955	87.40923004555032
0.20570324453176486	90.5269222916379
0.2746799547437327	81.78222286433052
0.22882979406431883	31.389316762031577
0.8951177076816685	51.3090793276412
0.6662781393594458	16.588686024612787
0.5592243055017212	13.920889345888412
0.7911760619608509	58.33648811163376
0.9044907330423345	3.8501727510831665
0.62964552288955	4.879399211864279
0.20570324453176486	57.10484735671127
0.2746799547437327	58.95942912958051
0.22882979406431883	95.24206344446048
0.8951177076816685	83.3654405662647
0.6662781393594458	38.51637379088627
0.6814695041800238	87.36094907292605
0.7911760619608509	60.06209622667634
0.9044907330423345	43.22331378937832
0.62964552288955	79.52944113129288
0.20570324453176486	81.12260143225086
0.2746799547437327	88.0359146993206
0.22882979406431883	76.35419695974757
0.8951177076816685	78.76637250282641
0.6662781393594458	81.7262187481561
0.6814695041800238	46.92654537140374
0.8475537419017511	23.79370691133448
0.9044907330423345	83.72785099670529
0.62964552288955	33.12262956843587
0.20570324453176486	52.70610379456254

0.2746799547437327	91.61617721425199
0.22882979406431883	17.78084990006332
0.8951177076816685	45.18976853703549
0.6662781393594458	83.31805164236314
0.6814695041800238	56.489686040618004
0.8475537419017511	38.12377504157126
0.6677236188073693	30.180040820007218
0.62964552288955	30.44781455914195
0.20570324453176486	78.91936384147856
0.2746799547437327	58.28040608377264
0.22882979406431883	40.36852016349861
0.8951177076816685	58.87560070336037
0.6662781393594458	39.760261204597136
0.6814695041800238	90.49634018208725
0.8475537419017511	58.40683641082224
0.6677236188073693	61.84541673092603
0.497324312501742	72.46106782446984
0.20570324453176486	35.97154851800387
0.2746799547437327	28.48668152347139
0.22882979406431883	28.982545325591147
0.8951177076816685	3.9127588412826917
0.6662781393594458	77.83382243645603
0.6814695041800238	92.32826937992907
0.8475537419017511	96.03620602697485
0.6677236188073693	26.0048363877446
0.497324312501742	32.73450620878202
0.7095830762586778	23.559851934389535
0.2746799547437327	94.8539356041944
0.22882979406431883	29.95810158856205
0.8951177076816685	20.44847836868227
0.6662781393594458	80.79702807409302
0.6814695041800238	60.31459498294367
0.8475537419017511	78.57275667420822
0.6677236188073693	90.51594660977521
0.497324312501742	51.03801805199025
0.7095830762586778	94.88283496251306
0.06007433598747219	53.87399133029391
0.22882979406431883	78.19742026266726
0.8951177076816685	17.565368595206415
0.6662781393594458	27.312853781282676
0.6814695041800238	91.69688617629616
0.8475537419017511	57.98920349519712
0.6677236188073693	41.26080855453011
0.497324312501742	46.07802504596576
0.7095830762586778	90.61952470102142
0.06007433598747219	24.53937016906238
0.9099184602480401	66.04890671081539
0.8951177076816685	55.7738064376575
0.6662781393594458	81.79070979367198
0.6814695041800238	87.1966770095543
0.8475537419017511	91.1727873840995
0.6677236188073693	73.72635861158122
0.497324312501742	52.27532199047107
0.7095830762586778	5.378322074696297
0.06007433598747219	56.591299783241126
0.9099184602480401	51.337277835375204
0.19956863426895669	12.349289265303772
0.6662781393594458	90.21733638036353
0.6814695041800238	93.0850934671026

0.8475537419017511	95.42781989063104
0.6677236188073693	61.439525880241916
0.497324312501742	5.311505866429838
0.7095830762586778	93.39401363861234
0.06007433598747219	42.252404885052115
0.9099184602480401	84.0105489086433
0.19956863426895669	2.5054548844564275
0.25653434037160106	3.0564318196515776
0.6814695041800238	11.663247626716853
0.8475537419017511	80.12473742385583
0.6677236188073693	54.44256028025659
0.497324312501742	17.51558828411341
0.7095830762586778	43.489878288183746
0.06007433598747219	66.82894184610564
0.9099184602480401	52.868960766608026
0.19956863426895669	36.27166784436807
0.25653434037160106	76.04202241819203
0.6594584631902157	17.11712399140216
0.8475537419017511	86.07610889526056
0.6677236188073693	69.2483653602239
0.497324312501742	23.17153210948295
0.7095830762586778	56.12710346660259
0.06007433598747219	27.936489762760733
0.9099184602480401	64.55645361891743
0.19956863426895669	50.82469460187363
0.25653434037160106	78.94762804776262
0.6594584631902157	69.61555138984127
0.20316371642745362	48.40172238769788
0.6677236188073693	1.5400689227422868
0.497324312501742	5.722602874614269
0.7095830762586778	5.978414219926671
0.06007433598747219	23.94905938960215
0.9099184602480401	77.880270903524
0.19956863426895669	5.471125597978013
0.25653434037160106	22.95494657369312
0.6594584631902157	36.782715270176496
0.20316371642745362	13.834595843037286
0.10907798298364516	97.19559638740002
0.497324312501742	81.61794803894314
0.7095830762586778	50.649192066397376
0.06007433598747219	88.97193397580088
0.9099184602480401	47.78496388711546
0.19956863426895669	4.282699064066786
0.25653434037160106	80.16108380979179
0.6594584631902157	7.498038300254854
0.20316371642745362	55.05146116911669
0.10907798298364516	16.409721438404
0.6362806510722646	60.402965502338866
0.7095830762586778	18.75372730366511
0.06007433598747219	57.15425661444268
0.9099184602480401	37.86942982438983
0.19956863426895669	96.42694556625958
0.25653434037160106	64.34605592156642
0.6594584631902157	24.674912476526274
0.20316371642745362	11.492811203574748
0.10907798298364516	10.958598255907486
0.6362806510722646	97.4307407768647
0.21592590850142365	35.38887000983349
0.06007433598747219	1.2762084643829203

0.9099184602480401	17.30905791130799
0.19956863426895669	17.0285596979581
0.25653434037160106	95.82573888915401
0.6594584631902157	71.03276058887499
0.20316371642745362	31.203691637930763
0.10907798298364516	50.53001538188872
0.6362806510722646	16.043170080984055
0.21592590850142365	34.66231635713982
0.945157721942989	97.69901453481921
0.9099184602480401	64.75600163742877
0.19956863426895669	89.54648086325952
0.25653434037160106	92.57829786208208
0.6594584631902157	19.494340903117234
0.20316371642745362	12.678110363242068
0.10907798298364516	91.09231108335848
0.6362806510722646	8.431606765297193
0.21592590850142365	26.19568645588105
0.945157721942989	63.659463636121316
0.6607366537961368	57.90410209547924
0.19956863426895669	72.11339949725377
0.25653434037160106	8.915443635932549
0.6594584631902157	2.946942621806882
0.20316371642745362	15.104115966571177
0.10907798298364516	87.86827311866249
0.6362806510722646	93.8169191792712
0.21592590850142365	31.921731649321206
0.945157721942989	92.02544487119171
0.6607366537961368	55.42855277249699
0.15880948890829222	15.714106832326696
0.25653434037160106	94.94359805296268
0.6594584631902157	28.43704029637569
0.20316371642745362	44.68054094346943
0.10907798298364516	86.82380038230517
0.6362806510722646	63.49093017835173
0.21592590850142365	81.31978855582408
0.945157721942989	85.37036183223933
0.6607366537961368	65.66167919536933
0.15880948890829222	87.98125840616963
0.26490457885041674	63.279729009253685
0.6594584631902157	3.804046648378986
0.20316371642745362	25.406810763194255
0.10907798298364516	42.49516980736982
0.6362806510722646	21.672804225165123
0.21592590850142365	26.946854732788694
0.945157721942989	7.715543055180758
0.6607366537961368	84.01960365108565
0.15880948890829222	25.804278389549868
0.26490457885041674	26.64314836411301
0.26717057991638865	94.7588938916365
0.20316371642745362	18.715161830421444
0.10907798298364516	63.43243459950015
0.6362806510722646	11.152714153421561
0.21592590850142365	10.561576303291833
0.945157721942989	51.309516532574975
0.6607366537961368	22.830974418904944
0.15880948890829222	47.20173136671369
0.26490457885041674	90.32180810409457
0.26717057991638865	17.133085178604478
0.06335828662112462	79.71133866445722

0.10907798298364516	1.2263317382214034
0.6362806510722646	11.649724625769425
0.21592590850142365	13.963644891435166
0.945157721942989	83.26210191161837
0.6607366537961368	85.94906754234479
0.15880948890829222	8.762701373762392
0.26490457885041674	9.285336162141057
0.26717057991638865	13.540037392336
0.06335828662112462	99.37548373585992
0.19053416474478202	55.695261289563284
0.6362806510722646	23.088351151135566
0.21592590850142365	14.568236290204958
0.945157721942989	74.81418760633419
0.6607366537961368	91.44742426079883
0.15880948890829222	27.947427745118297
0.26490457885041674	78.57346475179797
0.26717057991638865	14.958797448709985
0.06335828662112462	42.83955176216985
0.19053416474478202	96.69303916781267
0.8587084621965111	94.63247810914628
0.21592590850142365	64.37236776150985
0.945157721942989	17.480412367760145
0.6607366537961368	3.493218032475207
0.15880948890829222	5.40699319128539
0.26490457885041674	31.082923725018496
0.26717057991638865	29.72104842137201
0.06335828662112462	92.94052507774684
0.19053416474478202	94.3681292522118
0.8587084621965111	11.609816102549292
0.6249132307609485	15.566359765331162
0.945157721942989	81.55770416549628
0.6607366537961368	41.73451193011981
0.15880948890829222	11.317142421363545
0.26490457885041674	29.480259953627833
0.26717057991638865	83.66696098648607
0.06335828662112462	90.5556622339031
0.19053416474478202	28.051146695493333
0.8587084621965111	35.87930728964329
0.6249132307609485	97.39881349438899
0.9728289883740522	78.61323732027621
0.6607366537961368	78.68434002432288
0.15880948890829222	83.24508168505167
0.26490457885041674	95.86197027004299
0.26717057991638865	43.53413403062131
0.06335828662112462	3.9197218871295894
0.19053416474478202	87.34118237659598
0.8587084621965111	46.4997022694768
0.6249132307609485	9.65622131968125
0.9728289883740522	36.533948896612635
0.8506017693566247	12.81645107801345
0.15880948890829222	38.617130196373026
0.26490457885041674	8.874210398846955
0.26717057991638865	11.403531607928645
0.06335828662112462	78.3338849971587
0.19053416474478202	53.789727932116996
0.8587084621965111	23.318112322750956
0.6249132307609485	24.20791743507615
0.9728289883740522	79.42202579889414
0.8506017693566247	31.226619397806182

0.8394948740642394	95.63187754589505
0.26490457885041674	80.53214551257777
0.26717057991638865	31.97370103993087
0.06335828662112462	28.885262333514454
0.19053416474478202	31.59107881500231
0.8587084621965111	61.86389399721294
0.6249132307609485	46.86620448219555
0.9728289883740522	90.44109334735774
0.8506017693566247	83.413614256855
0.8394948740642394	88.66642750791812
0.06358789717148432	70.58396781807492
0.26717057991638865	17.4382314930237
0.06335828662112462	39.59906642296759
0.19053416474478202	10.204652771625852
0.8587084621965111	36.64452951098088
0.6249132307609485	14.169352749983238
0.9728289883740522	76.83678553014052
0.8506017693566247	38.55303841377553
0.8394948740642394	7.865458119726503
0.06358789717148432	97.83908040457794
0.2904688305420574	60.38513097111393
0.06335828662112462	3.7638251280219994
0.19053416474478202	31.470201011523876
0.8587084621965111	74.58911006743612
0.6249132307609485	95.34722309876952
0.9728289883740522	71.62546216665767
0.8506017693566247	54.26831348897638
0.8394948740642394	15.383868823355867
0.06358789717148432	68.6111667578041
0.2904688305420574	7.359121948231103
0.8348063108093235	88.61144280453257
0.19053416474478202	80.15826856652
0.8587084621965111	59.03876599060206
0.6249132307609485	62.54312738620865
0.9728289883740522	66.8102992836475
0.8506017693566247	49.39980297705999
0.8394948740642394	15.901733091452542
0.06358789717148432	72.00804277352239
0.2904688305420574	63.742630460744564
0.8348063108093235	13.165238195153316
0.061970190749695044	26.99915413722644
0.8587084621965111	30.125357055293964
0.6249132307609485	91.77306072036079
0.9728289883740522	63.754236216335464
0.8506017693566247	5.875791517854176
0.8394948740642394	7.02466920061422
0.06358789717148432	11.061308962250736
0.2904688305420574	86.32890053774237
0.8348063108093235	10.867262813308457
0.061970190749695044	5.2857538900025896
0.861982880801505	72.02042001583769
0.6249132307609485	22.02020781998454
0.9728289883740522	14.70454178831804
0.8506017693566247	54.60820466511935
0.8394948740642394	64.29447608919268
0.06358789717148432	87.91194904839107
0.2904688305420574	17.019070772793125
0.8348063108093235	72.3241933039573
0.061970190749695044	15.4541941679707

0.861982880801505	97.09809319906782
0.9626203763709649	86.12620337380072
0.9728289883740522	92.41995463845716
0.8506017693566247	41.81534695583351
0.8394948740642394	56.39366805094962
0.06358789717148432	18.424939746574864
0.2904688305420574	82.84594710049042
0.8348063108093235	55.71045475748297
0.061970190749695044	74.56211773379933
0.861982880801505	93.8334195933317
0.9626203763709649	29.397117904663226
0.3605406724877948	26.939733012119092
0.8506017693566247	22.54058926808944
0.8394948740642394	87.31370572340957
0.06358789717148432	73.10299714238661
0.2904688305420574	71.0219246088889
0.8348063108093235	8.215115702393538
0.061970190749695044	57.496088940657785
0.861982880801505	58.00253595146541
0.9626203763709649	66.03980801287324
0.3605406724877948	1.425875648981532
0.431397602293037	8.507597070981664
0.8394948740642394	52.999231784246845
0.06358789717148432	29.261968481458034
0.2904688305420574	22.901323398519892
0.8348063108093235	94.8177709223602
0.061970190749695044	39.2705390712944
0.861982880801505	36.32304106318748
0.9626203763709649	18.609474446602867
0.3605406724877948	60.56362072763753
0.431397602293037	32.927306290613174
0.8881046729508704	72.46488221145468
0.06358789717148432	83.1449388952832
0.2904688305420574	84.03940483715466
0.8348063108093235	71.03095603184552
0.061970190749695044	53.241677688595566
0.861982880801505	68.25219020665638
0.9626203763709649	63.01838664721427
0.3605406724877948	84.54620245132264
0.431397602293037	83.05132795819569
0.8881046729508704	5.856058747205132
0.8613821978061533	12.122936223672951
0.2904688305420574	92.7798675491253
0.8348063108093235	27.18639302481522
0.061970190749695044	90.42840901283358
0.861982880801505	91.57636736027303
0.9626203763709649	18.69168356164964
0.3605406724877948	87.8779309465808
0.431397602293037	27.12143390690496
0.8881046729508704	12.8797355436217
0.8613821978061533	37.751433299930724
0.0	71.02099205816741
0.8348063108093235	94.98555825972035
0.061970190749695044	75.3408238218192
0.861982880801505	77.78930448536045
0.9626203763709649	26.72477736508772
0.3605406724877948	13.349919836319678
0.431397602293037	85.49210494146035
0.8881046729508704	66.53342377457268

0.8613821978061533	70.81326232782878
0.0	47.66055008190003
0.5565536599068617	74.72829464570367
0.061970190749695044	91.69710037088078
0.861982880801505	92.42587710522899
0.9626203763709649	89.71104004183245
0.3605406724877948	30.66772629812456
0.431397602293037	3.466864485549514
0.8881046729508704	56.08826427765562
0.8613821978061533	57.65422239732662
0.0	48.19346676692146
0.5565536599068617	1.6423681113458808
0.15858527314156567	4.217413020951654
0.861982880801505	86.51144707952996
0.9626203763709649	83.20822900780779
0.3605406724877948	33.699372668876045
0.431397602293037	2.857510603569204
0.8881046729508704	11.28470049852308
0.8613821978061533	84.3766113855171
0.0	92.39800232441105
0.5565536599068617	72.82042664594535
0.15858527314156567	15.221326744594426
0.6387033965750694	9.988398169076072
0.9626203763709649	36.044331893348655
0.3605406724877948	31.062210016122975
0.431397602293037	73.8340759277321
0.8881046729508704	75.45083012929624
0.8613821978061533	77.68420050612394
0.0	36.60823057460021
0.5565536599068617	80.62147629975475
0.15858527314156567	68.7887419907682
0.6387033965750694	71.2770896305159
0.659688544301449	90.67999386626806
0.3605406724877948	83.40852513165946
0.431397602293037	39.8510599280451
0.8881046729508704	90.2237249222642
0.8613821978061533	22.421635363256545
0.0	16.82503556471162
0.5565536599068617	55.38695847557075
0.15858527314156567	10.074569595281623
0.6387033965750694	91.12128531962132
0.659688544301449	3.9037983023407237
0.5596997632997004	4.132227536033842
0.431397602293037	84.10971370015739
0.8881046729508704	10.254848720665198
0.8613821978061533	9.066416845130936
0.0	51.27473427047341
0.5565536599068617	35.32598241911686
0.15858527314156567	83.45111198784217
0.6387033965750694	94.8637200132555
0.659688544301449	73.6161988736206
0.5596997632997004	4.00843784517038
0.8624590835032977	68.44365646878181
0.8881046729508704	15.27401822830656
0.8613821978061533	11.625563889498684
0.0	5.7567823970038985
0.5565536599068617	27.768278295244876
0.15858527314156567	60.76404030165521
0.6387033965750694	85.60535061410259

0.659688544301449	87.45923686399898
0.5596997632997004	18.150815333147563
0.8624590835032977	72.66612536306927
0.3771564036246608	40.52568239025232
0.8613821978061533	49.77061295067282
0.0	22.62327683448471
0.5565536599068617	11.38427787636566
0.15858527314156567	84.84827228239013
0.6387033965750694	48.64622356890011
0.659688544301449	82.73115718817984
0.5596997632997004	79.34976958093272
0.8624590835032977	70.9309540086267
0.3771564036246608	88.5538817069083
0.09994518027903934	61.95517832452555
0.0	43.16236328652587
0.5565536599068617	80.50950645027147
0.15858527314156567	87.98103472812905
0.6387033965750694	85.01938190141482
0.659688544301449	20.16765351020876
0.5596997632997004	9.825653318849243
0.8624590835032977	43.952297563892024
0.3771564036246608	6.834327955627347
0.09994518027903934	96.96732602547695
0.3085129734030654	65.72723617063687
0.5565536599068617	93.86344043449863
0.15858527314156567	40.35635700981192
0.6387033965750694	18.01337639278276
0.659688544301449	6.969066329386852
0.5596997632997004	22.963359270949685
0.8624590835032977	40.39260295362405
0.3771564036246608	38.31455771882707
0.09994518027903934	96.5959063474675
0.3085129734030654	4.077226882698203
0.8315980540556376	11.422331197389171
0.15858527314156567	59.227192042030794
0.6387033965750694	10.809856757088566
0.659688544301449	2.5024497694162697
0.5596997632997004	5.744701572205647
0.8624590835032977	16.805505162188943
0.3771564036246608	95.86577576539744
0.09994518027903934	91.31897171671696
0.3085129734030654	33.632519996414295
0.8315980540556376	57.401955350302494
0.8965763850206687	64.53166353364608
0.6387033965750694	7.894509591988841
0.659688544301449	54.22678045141544
0.5596997632997004	86.73582155328225
0.8624590835032977	57.7569980936124
0.3771564036246608	91.38462370780002
0.09994518027903934	73.53223326073768
0.3085129734030654	60.647174043115626
0.8315980540556376	17.09349903889412
0.8965763850206687	60.99324810470512
0.6002680575694376	48.061244101846455
0.659688544301449	24.736819324895926
0.5596997632997004	70.57292507938425
0.8624590835032977	5.807527476984833
0.3771564036246608	3.4868813579325786
0.09994518027903934	51.804408603905195

```
|  0.3085129734030654   | 94.32721985062864  |
|  0.8315980540556376   | 10.085108015030249 |
|  0.8965763850206687   | 91.63178259818524  |
|  0.6002680575694376   | 32.96595042435237  |
|  0.9075551685431381   | 5.509795301580483  |
|  0.5596997632997004   | 54.74265045097957  |
|  0.8624590835032977   | 29.522429652445354 |
|  0.3771564036246608   | 14.018349081910738 |
|  0.09994518027903934  | 82.74724028278091  |
|  0.3085129734030654   | 59.64157527499423  |
|  0.8315980540556376   | 78.99486208932441  |
|  0.8965763850206687   |  87.5552284409556  |
|  0.6002680575694376   | 79.05250998454527  |
|  0.9075551685431381   | 27.570787876987243 |
|  0.1950037835883633   | 61.948277323588364 |
|  0.8624590835032977   | 17.178958907522116 |
|  0.3771564036246608   | 27.70511456295158  |
|  0.09994518027903934  | 61.650192225699435 |
|  0.3085129734030654   | 65.65049587995638  |
|  0.8315980540556376   |  13.0471052772396  |
|  0.8965763850206687   | 72.44399070726519  |
|  0.6002680575694376   | 22.909540255677832 |
|  0.9075551685431381   | 82.60557934228095  |
|  0.1950037835883633   | 10.253683557444957 |
|  0.5465073870614098   | 92.55525437984339  |
|  0.3771564036246608   | 3.955928142575823  |
|  0.09994518027903934  | 43.12738302860457  |
|  0.3085129734030654   | 85.19374116083718  |
|  0.8315980540556376   | 30.657256767372278 |
|  0.8965763850206687   | 5.255080512738626  |
|  0.6002680575694376   | 53.53177341429004  |
|  0.9075551685431381   | 10.515089817857284 |
|  0.1950037835883633   | 26.181582734396883 |
|  0.5465073870614098   | 65.43085862830317  |
|   0.225435937328899   | 70.99405554191213  |
+-----------------------+--------------------+
```

In [3]:
```
# Next-Frame Approach to Model Time Series
#
# Igor Mol <igor.mol@makes.ai>
#
# This program implements a class called TimeSeriesModel to perform time ser
# modeling according to the so-called Next-Frame Technique. We employ Long S
# Term Memory (LSTM) neural networks. The program reads time series data fro
# CSV file, normalizes it using Min-Max scaling, and then converts it into
# sequences of input-output pairs suitable for training an LSTM model. The L
# model is built and trained using the Keras library, aiming to predict futu
# values in the time series. After training, the model generates predictions
# a test set. These predictions and the actual values are denormalized to th
# original scale. The program prints a table comparing the predicted and
# actual values and plots the time series to visually assess the model's per
# mance. This approach aids in understanding and forecasting patterns in tim
# series data, providing insights into potential future trends. The use of L
# networks allows the model to capture long-term dependencies and patterns i
# time series, making it effective for various prediction tasks.

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tabulate import tabulate

# Define a class 'TimeSeriesModel' to encapsulate the functionality of time
class TimeSeriesModel:
    def __init__(self, file_path, sequence_length=10):
        self.file_path = file_path
        self.sequence_length = sequence_length
        self.scaler = MinMaxScaler()

    # Load time series data from a CSV file and return the 'Value' column as
    def load_data(self):
        time_series_df = pd.read_csv(self.file_path)
        time_series_df['Date'] = pd.to_datetime(time_series_df['Date'])
        values = time_series_df['Value'].values.reshape(-1, 1)
        return values

    # Normalize the input data using Min-Max scaling
    def normalize_data(self, data):
        return self.scaler.fit_transform(data)

    # Create input-output sequences for the LSTM model
    def create_sequences(self, data):
        X, y = [], []
        for i in range(len(data) - self.sequence_length):
            X.append(data[i:i + self.sequence_length])
            y.append(data[i + self.sequence_length])
        return np.array(X), np.array(y)

    # Build and compile an LSTM model with specified architecture
    def build_lstm_model(self):
        model = Sequential()
        model.add(LSTM(50, activation='relu', input_shape=(self.sequence_ler
        model.add(Dense(1))
```

```
        model.compile(optimizer='adam', loss='mean_squared_error')
        return model

    # Train the LSTM model with training data
    def train_model(self, model, X_train, y_train, epochs=50, batch_size=32)
        model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, ve
        return model

    # Denormalize the normalized data to obtain the original scale
    def denormalize_data(self, normalized_data):
        return self.scaler.inverse_transform(normalized_data)

    # Generate predictions using the trained LSTM model on test data
    def generate_predictions(self, model, X_test):
        return model.predict(X_test)

    # Print a table comparing predicted and actual values
    def print_results_table(self, y_test_denormalized, y_pred_denormalized):
        results_table = pd.DataFrame({
            'Actual': y_test_denormalized.flatten(),
            'Predicted': y_pred_denormalized.flatten()
        })

        table = tabulate(results_table, headers='keys', tablefmt='fancy_gric
        print("\nTable with Predicted vs Actual Values:")
        print(table)

    # Plot the actual and predicted time series values
    def plot_time_series(self, y_test_denormalized, y_pred_denormalized):
        plt.figure(figsize=(10, 6))
        plt.plot(y_test_denormalized, label='Actual')
        plt.plot(y_pred_denormalized, label='Predicted')
        plt.legend()
        plt.title('Actual vs. Predicted Time Series')
        plt.xlabel('Time')
        plt.ylabel('Value')
        plt.show()

# Define the main function to execute the time series modeling
def main():
    file_path = "/Users/igormol/Desktop/time_series_data.csv"  # Update with
    time_series_model = TimeSeriesModel(file_path)

    # Load and preprocess the time series data
    values = time_series_model.load_data()
    values_scaled = time_series_model.normalize_data(values)

    # Convert the time series data into input-output sequences
    X, y = time_series_model.create_sequences(values_scaled)

    # Split the data into training and testing sets
    train_size = int(len(values_scaled) * 0.8)
    X_train, X_test, y_train, y_test = X[:train_size], X[train_size:], y[:tr

    # Build and train the LSTM model
    lstm_model = time_series_model.build_lstm_model()
    trained_lstm_model = time_series_model.train_model(lstm_model, X_train,

    # Generate predictions on the test set
    y_pred = time_series_model.generate_predictions(trained_lstm_model, X_te
```

```
    y_pred = time_series_model.generate_predictions(trained_lstm_model, X_te

    # Denormalize the predictions and actual values
    y_pred_denormalized = time_series_model.denormalize_data(y_pred)
    y_test_denormalized = time_series_model.denormalize_data(y_test)

    # Print results table and plot time series
    time_series_model.print_results_table(y_test_denormalized, y_pred_denorm
    time_series_model.plot_time_series(y_test_denormalized, y_pred_denormali

# Execute the main function if the script is run
if __name__ == "__main__":
    main()
```

```
Epoch 1/50
10/10 - 2s - loss: 0.3057 - 2s/epoch - 248ms/step
Epoch 2/50
10/10 - 0s - loss: 0.1949 - 93ms/epoch - 9ms/step
Epoch 3/50
10/10 - 0s - loss: 0.1129 - 109ms/epoch - 11ms/step
Epoch 4/50
10/10 - 0s - loss: 0.0934 - 110ms/epoch - 11ms/step
Epoch 5/50
10/10 - 0s - loss: 0.0912 - 93ms/epoch - 9ms/step
Epoch 6/50
10/10 - 0s - loss: 0.0918 - 97ms/epoch - 10ms/step
Epoch 7/50
10/10 - 0s - loss: 0.0912 - 116ms/epoch - 12ms/step
Epoch 8/50
10/10 - 0s - loss: 0.0901 - 91ms/epoch - 9ms/step
Epoch 9/50
10/10 - 0s - loss: 0.0898 - 98ms/epoch - 10ms/step
Epoch 10/50
10/10 - 0s - loss: 0.0898 - 102ms/epoch - 10ms/step
Epoch 11/50
10/10 - 0s - loss: 0.0896 - 89ms/epoch - 9ms/step
Epoch 12/50
10/10 - 0s - loss: 0.0890 - 91ms/epoch - 9ms/step
Epoch 13/50
10/10 - 0s - loss: 0.0899 - 100ms/epoch - 10ms/step
Epoch 14/50
10/10 - 0s - loss: 0.0889 - 94ms/epoch - 9ms/step
Epoch 15/50
10/10 - 0s - loss: 0.0889 - 88ms/epoch - 9ms/step
Epoch 16/50
10/10 - 0s - loss: 0.0885 - 98ms/epoch - 10ms/step
Epoch 17/50
10/10 - 0s - loss: 0.0880 - 94ms/epoch - 9ms/step
Epoch 18/50
10/10 - 0s - loss: 0.0892 - 90ms/epoch - 9ms/step
Epoch 19/50
10/10 - 0s - loss: 0.0873 - 136ms/epoch - 14ms/step
Epoch 20/50
10/10 - 0s - loss: 0.0882 - 182ms/epoch - 18ms/step
Epoch 21/50
10/10 - 0s - loss: 0.0871 - 112ms/epoch - 11ms/step
Epoch 22/50
10/10 - 0s - loss: 0.0868 - 146ms/epoch - 15ms/step
Epoch 23/50
10/10 - 0s - loss: 0.0866 - 218ms/epoch - 22ms/step
```

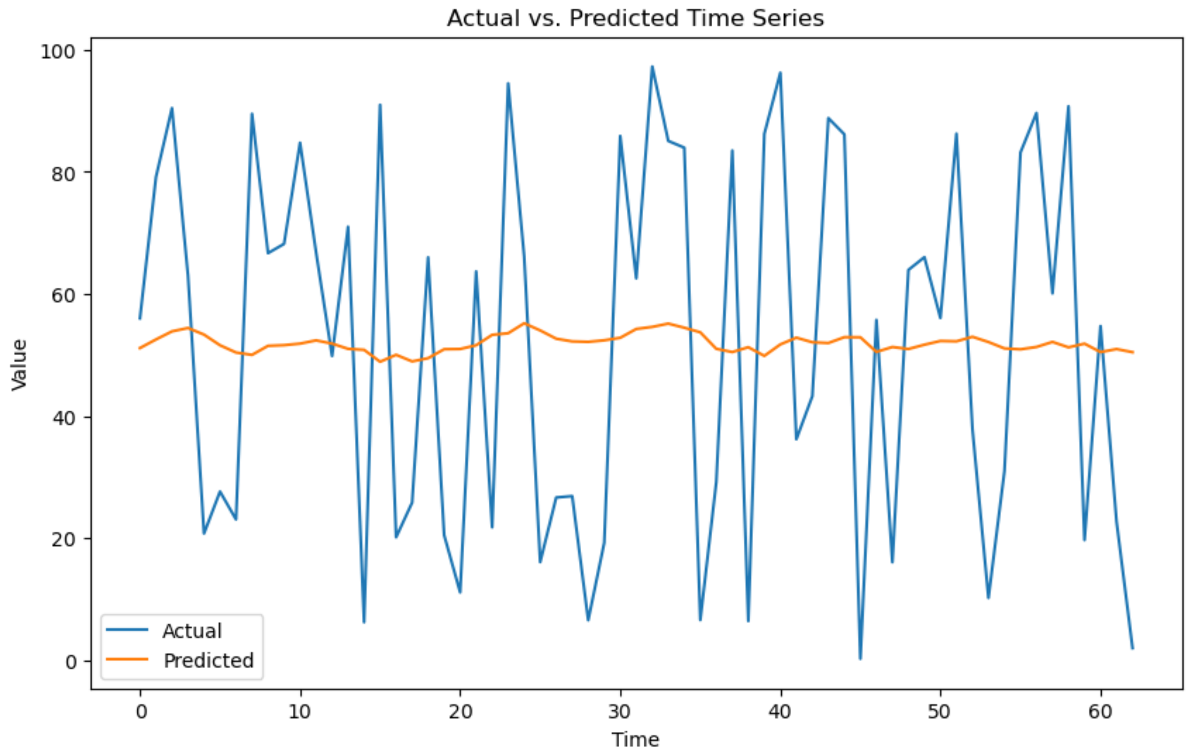
```
Epoch 24/50
10/10 - 0s - loss: 0.0875 - 148ms/epoch - 15ms/step
Epoch 25/50
10/10 - 0s - loss: 0.0876 - 121ms/epoch - 12ms/step
Epoch 26/50
10/10 - 0s - loss: 0.0860 - 184ms/epoch - 18ms/step
Epoch 27/50
10/10 - 0s - loss: 0.0870 - 186ms/epoch - 19ms/step
Epoch 28/50
10/10 - 0s - loss: 0.0876 - 174ms/epoch - 17ms/step
Epoch 29/50
10/10 - 0s - loss: 0.0855 - 128ms/epoch - 13ms/step
Epoch 30/50
10/10 - 0s - loss: 0.0854 - 164ms/epoch - 16ms/step
Epoch 31/50
10/10 - 0s - loss: 0.0866 - 197ms/epoch - 20ms/step
Epoch 32/50
10/10 - 0s - loss: 0.0865 - 96ms/epoch - 10ms/step
Epoch 33/50
10/10 - 0s - loss: 0.0857 - 86ms/epoch - 9ms/step
Epoch 34/50
10/10 - 0s - loss: 0.0860 - 93ms/epoch - 9ms/step
Epoch 35/50
10/10 - 0s - loss: 0.0854 - 144ms/epoch - 14ms/step
Epoch 36/50
10/10 - 0s - loss: 0.0864 - 116ms/epoch - 12ms/step
Epoch 37/50
10/10 - 0s - loss: 0.0854 - 132ms/epoch - 13ms/step
Epoch 38/50
10/10 - 0s - loss: 0.0856 - 116ms/epoch - 12ms/step
Epoch 39/50
10/10 - 0s - loss: 0.0850 - 83ms/epoch - 8ms/step
Epoch 40/50
10/10 - 0s - loss: 0.0865 - 79ms/epoch - 8ms/step
Epoch 41/50
10/10 - 0s - loss: 0.0843 - 81ms/epoch - 8ms/step
Epoch 42/50
10/10 - 0s - loss: 0.0838 - 80ms/epoch - 8ms/step
Epoch 43/50
10/10 - 0s - loss: 0.0842 - 80ms/epoch - 8ms/step
Epoch 44/50
10/10 - 0s - loss: 0.0840 - 77ms/epoch - 8ms/step
Epoch 45/50
10/10 - 0s - loss: 0.0837 - 87ms/epoch - 9ms/step
Epoch 46/50
10/10 - 0s - loss: 0.0845 - 84ms/epoch - 8ms/step
Epoch 47/50
10/10 - 0s - loss: 0.0834 - 80ms/epoch - 8ms/step
Epoch 48/50
10/10 - 0s - loss: 0.0837 - 78ms/epoch - 8ms/step
Epoch 49/50
10/10 - 0s - loss: 0.0834 - 79ms/epoch - 8ms/step
Epoch 50/50
10/10 - 0s - loss: 0.0832 - 81ms/epoch - 8ms/step
2/2 [==============================] - 0s 6ms/step

Table with Predicted vs Actual Values:
```

Actual	Predicted

56.0026	51.1305
79.1372	52.5373
90.4391	53.8631
63.0263	54.4159
20.7428	53.3342
27.6224	51.5881
23.0494	50.3997
89.5042	50.0137
66.68	51.4952
68.1952	51.6012
84.7602	51.8574
66.8242	52.3867
49.8287	51.8573
70.9992	51.0011
6.21789	50.8362
90.9804	48.8853
20.1309	50.0231
25.8126	48.9425
65.9998	49.4666
20.4895	50.9405
11.1055	50.9714
63.6881	51.5884
21.7624	53.2942
94.4951	53.5562
66.1273	55.1885
16.0656	54.0168
26.6474	52.662
26.8734	52.2182
6.54543	52.145

19.2298	52.3988
85.8728	52.8112
62.5543	54.257
97.255	54.5977
85.0642	55.1348
83.9564	54.4624
6.56833	53.7091
29.1972	51.0066
83.4888	50.4713
6.40698	51.2685
86.1994	49.8466
96.2368	51.7246
36.1861	52.8337
43.2533	52.0791
88.8047	51.9547
86.1395	52.91
0.226143	52.8928
55.7362	50.5038
16.0433	51.2995
63.9297	50.9751
66.0228	51.6771
56.05	52.2912
86.2469	52.2338
37.8433	52.9751
10.1946	52.1188
30.9969	51.0692
83.1688	50.9219
89.6497	51.3088
60.0962	52.1351
90.7447	51.2484

19.6756	51.8475
54.7342	50.4701
22.7109	50.9669
1.97025	50.4511



In [4]:
```
# Monte Carlo Method to Detect Time Series Anomalies
#
# Igor Mol <igor.mol@makes.ai>
#
# In Python program, the Monte Carlo method is applied to analyze time serie
# data for detecting and handling anomalies. The code begins by creating a c
# called MonteCarloAnomalyDetection with key parameters like the file path t
# CSV containing time series data, a threshold for anomaly detection, and th
# number of simulations for anomaly replacement. After loading and normalizi
# the time series data, the code uses Z-scores to identify anomalies. Z-scor
# measure how far each data point deviates from the mean, and if this deviat
# exceeds a set threshold, the point is marked as an anomaly.
#     The anomalies are then replaced using Monte Carlo simulations. For eac
# anomaly, simulated values are generated from a normal distribution based o
# the mean and standard deviation of the data, and the anomaly is replaced w
# the mean of these simulated values.
#     The main function of the code executes the entire anomaly detection
# process, displaying the results in a formatted table that includes the dat
# original values, replaced values, and an indicator for anomalies. Addition
# ly, the program creates a time series plot highlighting the original and r
# eplaced values, emphasizing the detected anomalies in red. The Monte Carlo
# technique utilized in this code is a statistical approach that leverages r
# domness through simulations. By replacing anomalies with simulated values
# based on the data's statistical properties, the Monte Carlo method offers
# flexible and effective way to handle outliers and uncertainties in time se
# data, showcasing its versatility in practical applications.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# The `MonteCarloAnomalyDetection' class is initiated with essential paramet
# the path to a CSV file containing time series data, a threshold for anomal
# detection, and the number of simulations for anomaly replacement. Default
# lues for the threshold and number of simulations are set at 1.5 and 1000,
# pectively. The class holds attributes such as:
# - file_path,
# - threshold,
# - num_simulations,
# - df (DataFrame for time series data),
# - anomalies, and
# - replaced_values.

class MonteCarloAnomalyDetection:

    def __init__(self, file_path, threshold=1.5, num_simulations=1000):
        self.file_path = file_path
        self.threshold = threshold
        self.num_simulations = num_simulations
        self.df = None
        self.anomalies = None
        self.replaced_values = None

    def load_data(self):
        self.df = pd.read_csv(self.file_path)
        self.df['Date'] = pd.to_datetime(self.df['Date'])
        self.df = self.df.sort_values(by='Date')

# detect anomalies:
```

```
# detect_anomalies:
# - Detects anomalies in the time series based on Z-scores.
# - Z-scores are calculated as the difference between data and the mean, div
# by the standard deviation.
# - Anomalies are identified if the Z-score exceeds the defined threshold.

    def detect_anomalies(self, data):
        mean_val = np.mean(data)
        std_val = np.std(data)
        z_scores = np.abs((data - mean_val) / std_val)
        anomalies = z_scores > self.threshold
        return anomalies

# replace_anomalies:
# - Replaces anomalies in the time series using Monte Carlo simulations.
# - For each data point identified as an anomaly, simulated values are gener
# from a normal distribution based on the mean and standard deviation of the
# - The anomaly value is replaced by the mean of the simulated values.

    def replace_anomalies(self, data):
        replaced_data = data.copy()
        for i in range(len(data)):
            if self.anomalies[i]:
                simulation_values = np.random.normal(np.mean(data), np.std(d
                replaced_data[i] = np.mean(simulation_values)
        return replaced_data

# run_anomaly_detection:
# - Executes the anomaly detection process.
# - Calls load_data, detect_anomalies, and replace_anomalies to populate and
# lies and replaced_values.

    def run_anomaly_detection(self):
        self.load_data()
        self.anomalies = self.detect_anomalies(self.df['Value'])
        self.replaced_values = self.replace_anomalies(self.df['Value'])

# display_results:
# - Displays results in a formatted table using a DataFrame.
# - Displays 'Date', 'Original' time series values, 'Replaced' values after
# maly replacement, and 'Anomaly' flag.

    def display_results(self):
        results = pd.DataFrame({'Date': self.df['Date'], 'Original': self.df
                                'Replaced': self.replaced_values, 'Anomaly':
        pd.set_option('display.max_rows', None)
        pd.set_option('display.max_columns', None)
        print(results)
        pd.reset_option('display.max_rows')
        pd.reset_option('display.max_columns')

# plot_time_series:
# - Plots the original time series, the series with replaced anomalies, and
# highlights anomaly points.
# - Uses matplotlib to create a time series plot.

    def plot_time_series(self):
        plt.figure(figsize=(12, 6))
        plt.plot(self.df['Date'], self.df['Value'], label='Original', marker
```

```
        plt.plot(self.df['Date'], self.replaced_values, label='Replaced', ma
        plt.scatter(self.df['Date'][self.anomalies], self.df['Value'][self.a
        plt.title('Monte Carlo Anomaly Detection and Replacement')
        plt.xlabel('Date')
        plt.ylabel('Value')
        plt.legend()
        plt.show()

def main():
    anomaly_detector = MonteCarloAnomalyDetection(file_path="/Users/igormol/
    anomaly_detector.run_anomaly_detection()
    anomaly_detector.display_results()
    anomaly_detector.plot_time_series()

if __name__ == "__main__":
    main()
```

	Date	Original	Replaced	Anomaly
0	2022-01-01	17.205673	17.205673	False
1	2022-01-02	23.882583	23.882583	False
2	2022-01-03	39.984362	39.984362	False
3	2022-01-04	56.131855	56.131855	False
4	2022-01-05	36.124378	36.124378	False
5	2022-01-06	30.250211	30.250211	False
6	2022-01-07	55.043937	55.043937	False
7	2022-01-08	7.400941	7.400941	False
8	2022-01-09	48.476184	48.476184	False
9	2022-01-10	78.648685	78.648685	False
10	2022-01-11	81.300903	81.300903	False
11	2022-01-12	27.328824	27.328824	False
12	2022-01-13	18.239756	18.239756	False
13	2022-01-14	76.547443	76.547443	False
14	2022-01-15	30.689008	30.689008	False
15	2022-01-16	0.958607	50.646826	True
16	2022-01-17	51.138039	51.138039	False
17	2022-01-18	77.246803	77.246803	False
18	2022-01-19	91.421612	91.421612	False
19	2022-01-20	84.154631	84.154631	False
20	2022-01-21	77.347057	77.347057	False
21	2022-01-22	8.236078	8.236078	False
22	2022-01-23	39.757374	39.757374	False
23	2022-01-24	4.565118	50.145036	True
24	2022-01-25	24.750312	24.750312	False
25	2022-01-26	29.149634	29.149634	False
26	2022-01-27	8.272093	8.272093	False
27	2022-01-28	18.452323	18.452323	False
28	2022-01-29	3.472170	51.712662	True
29	2022-01-30	81.266011	81.266011	False
30	2022-01-31	97.880334	50.415827	True
31	2022-02-01	92.026172	92.026172	False
32	2022-02-02	35.802006	35.802006	False
33	2022-02-03	36.395385	36.395385	False
34	2022-02-04	78.625231	78.625231	False
35	2022-02-05	82.484448	82.484448	False
36	2022-02-06	55.560241	55.560241	False
37	2022-02-07	44.502025	44.502025	False
38	2022-02-08	34.140065	34.140065	False
39	2022-02-09	9.914127	9.914127	False
40	2022-02-10	33.471826	33.471826	False
41	2022-02-11	61.985206	61.985206	False

42	2022-02-12	51.508424	51.508424	False
43	2022-02-13	93.994475	51.579686	True
44	2022-02-14	61.489319	61.489319	False
45	2022-02-15	39.644319	39.644319	False
46	2022-02-16	16.253762	16.253762	False
47	2022-02-17	17.990712	17.990712	False
48	2022-02-18	26.656962	26.656962	False
49	2022-02-19	78.792661	78.792661	False
50	2022-02-20	20.987891	20.987891	False
51	2022-02-21	72.791894	72.791894	False
52	2022-02-22	8.743812	8.743812	False
53	2022-02-23	67.491099	67.491099	False
54	2022-02-24	72.393459	72.393459	False
55	2022-02-25	59.699182	59.699182	False
56	2022-02-26	41.311351	41.311351	False
57	2022-02-27	33.093305	33.093305	False
58	2022-02-28	29.380650	29.380650	False
59	2022-03-01	51.249703	51.249703	False
60	2022-03-02	35.362935	35.362935	False
61	2022-03-03	89.118747	89.118747	False
62	2022-03-04	53.392011	53.392011	False
63	2022-03-05	84.081062	84.081062	False
64	2022-03-06	6.073966	50.562192	True
65	2022-03-07	96.241722	50.713471	True
66	2022-03-08	6.510861	50.517012	True
67	2022-03-09	48.224867	48.224867	False
68	2022-03-10	27.549201	27.549201	False
69	2022-03-11	16.177437	16.177437	False
70	2022-03-12	2.087524	50.445379	True
71	2022-03-13	81.039423	81.039423	False
72	2022-03-14	60.843868	60.843868	False
73	2022-03-15	64.328461	64.328461	False
74	2022-03-16	98.581993	51.468240	True
75	2022-03-17	0.229440	50.027854	True
76	2022-03-18	42.466619	42.466619	False
77	2022-03-19	71.991242	71.991242	False
78	2022-03-20	95.856792	49.093140	True
79	2022-03-21	2.046130	50.027032	True
80	2022-03-22	42.088354	42.088354	False
81	2022-03-23	40.274422	40.274422	False
82	2022-03-24	80.872092	80.872092	False
83	2022-03-25	70.789452	70.789452	False
84	2022-03-26	15.641700	15.641700	False
85	2022-03-27	61.421355	61.421355	False
86	2022-03-28	67.323260	67.323260	False
87	2022-03-29	24.033506	24.033506	False
88	2022-03-30	36.818106	36.818106	False
89	2022-03-31	56.231732	56.231732	False
90	2022-04-01	81.227223	81.227223	False
91	2022-04-02	25.365495	25.365495	False
92	2022-04-03	72.344287	72.344287	False
93	2022-04-04	40.526373	40.526373	False
94	2022-04-05	48.293645	48.293645	False
95	2022-04-06	66.780186	66.780186	False
96	2022-04-07	79.509808	79.509808	False
97	2022-04-08	90.226501	90.226501	False
98	2022-04-09	69.436928	69.436928	False
99	2022-04-10	74.245766	74.245766	False
100	2022-04-11	5.624580	48.192223	True

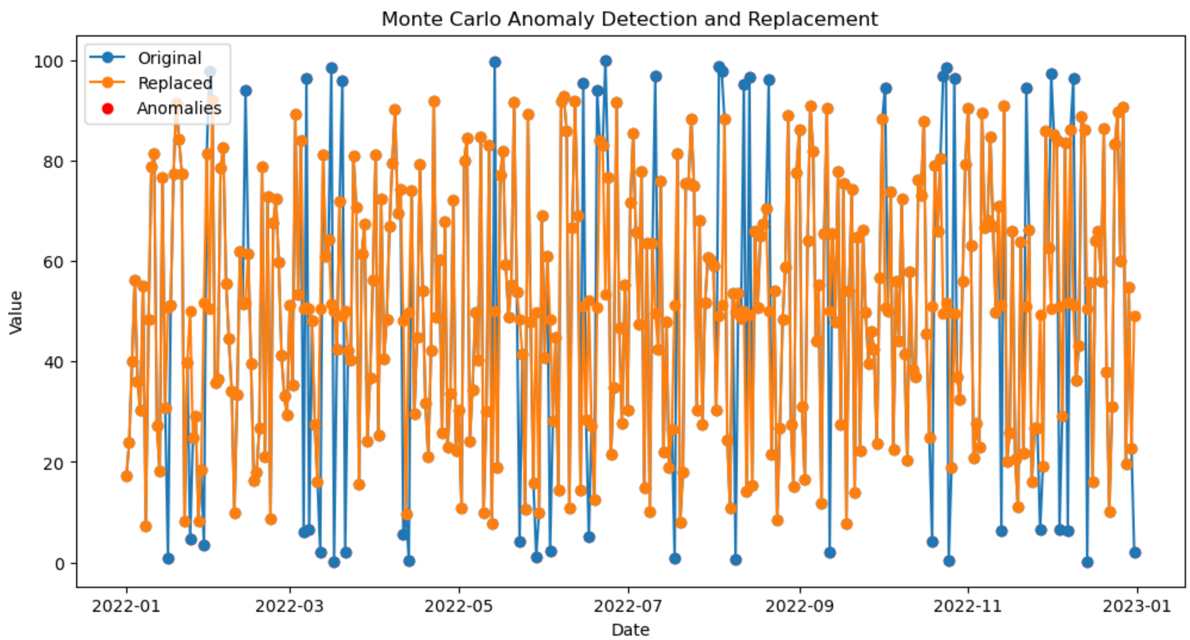
```
101 2022-04-12   9.581872   9.581872    False
102 2022-04-13   0.480818  49.687194     True
103 2022-04-14  74.075054  74.075054    False
104 2022-04-15  29.532013  29.532013    False
105 2022-04-16  44.919033  44.919033    False
106 2022-04-17  79.318710  79.318710    False
107 2022-04-18  53.955188  53.955188    False
108 2022-04-19  31.693897  31.693897    False
109 2022-04-20  21.070292  21.070292    False
110 2022-04-21  42.114909  42.114909    False
111 2022-04-22  91.878629  91.878629    False
112 2022-04-23  48.940667  48.940667    False
113 2022-04-24  60.198965  60.198965    False
114 2022-04-25  25.817230  25.817230    False
115 2022-04-26  67.865390  67.865390    False
116 2022-04-27  22.904959  22.904959    False
117 2022-04-28  33.719957  33.719957    False
118 2022-04-29  72.004106  72.004106    False
119 2022-04-30  22.161655  22.161655    False
120 2022-05-01  30.435711  30.435711    False
121 2022-05-02  10.780253  10.780253    False
122 2022-05-03  79.922188  79.922188    False
123 2022-05-04  84.526871  84.526871    False
124 2022-05-05  24.151776  24.151776    False
125 2022-05-06  34.466848  34.466848    False
126 2022-05-07  49.865918  49.865918    False
127 2022-05-08  40.194334  40.194334    False
128 2022-05-09  84.583993  84.583993    False
129 2022-05-10   9.901063   9.901063    False
130 2022-05-11  29.988062  29.988062    False
131 2022-05-12  83.091482  83.091482    False
132 2022-05-13   7.749104   7.749104    False
133 2022-05-14  99.619453  50.034891     True
134 2022-05-15  18.921870  18.921870    False
135 2022-05-16  76.994205  76.994205    False
136 2022-05-17  81.927838  81.927838    False
137 2022-05-18  59.360297  59.360297    False
138 2022-05-19  48.937226  48.937226    False
139 2022-05-20  55.295679  55.295679    False
140 2022-05-21  91.559731  91.559731    False
141 2022-05-22  53.731649  53.731649    False
142 2022-05-23   4.133606  48.366782     True
143 2022-05-24  41.550117  41.550117    False
144 2022-05-25  10.635445  10.635445    False
145 2022-05-26  89.291472  89.291472    False
146 2022-05-27  47.844574  47.844574    False
147 2022-05-28  15.793790  15.793790    False
148 2022-05-29   1.174022  49.756633     True
149 2022-05-30   9.978596   9.978596    False
150 2022-05-31  69.128886  69.128886    False
151 2022-06-01  40.883477  40.883477    False
152 2022-06-02  60.864794  60.864794    False
153 2022-06-03   2.242390  48.249878     True
154 2022-06-04  28.208840  28.208840    False
155 2022-06-05  44.777027  44.777027    False
156 2022-06-06  14.395065  14.395065    False
157 2022-06-07  91.820033  91.820033    False
158 2022-06-08  92.690335  92.690335    False
159 2022-06-09  85.810992  85.810992    False
```

```
160 2022-06-10  10.866967  10.866967    False
161 2022-06-11  66.623362  66.623362    False
162 2022-06-12  91.739415  91.739415    False
163 2022-06-13  69.079676  69.079676    False
164 2022-06-14  14.473059  14.473059    False
165 2022-06-15  95.460439  51.052473     True
166 2022-06-16  28.527694  28.527694    False
167 2022-06-17   5.068678  52.065619     True
168 2022-06-18  27.289601  27.289601    False
169 2022-06-19  12.604815  12.604815    False
170 2022-06-20  93.850376  50.829920     True
171 2022-06-21  83.929756  83.929756    False
172 2022-06-22  82.972703  82.972703    False
173 2022-06-23  99.965052  53.407343     True
174 2022-06-24  76.588754  76.588754    False
175 2022-06-25  21.451446  21.451446    False
176 2022-06-26  34.945709  34.945709    False
177 2022-06-27  91.701902  91.701902    False
178 2022-06-28  46.681014  46.681014    False
179 2022-06-29  27.613734  27.613734    False
180 2022-06-30  55.322381  55.322381    False
181 2022-07-01  30.284241  30.284241    False
182 2022-07-02  71.565274  71.565274    False
183 2022-07-03  85.340038  85.340038    False
184 2022-07-04  65.666911  65.666911    False
185 2022-07-05  47.355871  47.355871    False
186 2022-07-06  77.902354  77.902354    False
187 2022-07-07  14.926307  14.926307    False
188 2022-07-08  63.613576  63.613576    False
189 2022-07-09  10.105969  10.105969    False
190 2022-07-10  63.447903  63.447903    False
191 2022-07-11  96.893859  49.651238     True
192 2022-07-12  42.535514  42.535514    False
193 2022-07-13  75.950996  75.950996    False
194 2022-07-14  21.891973  21.891973    False
195 2022-07-15  47.784744  47.784744    False
196 2022-07-16  18.836006  18.836006    False
197 2022-07-17  26.556727  26.556727    False
198 2022-07-18   0.781499  51.330803     True
199 2022-07-19  81.340398  81.340398    False
200 2022-07-20   8.014817   8.014817    False
201 2022-07-21  18.067338  18.067338    False
202 2022-07-22  75.510877  75.510877    False
203 2022-07-23  75.386231  75.386231    False
204 2022-07-24  88.317835  88.317835    False
205 2022-07-25  74.849072  74.849072    False
206 2022-07-26  30.258284  30.258284    False
207 2022-07-27  68.031887  68.031887    False
208 2022-07-28  27.429936  27.429936    False
209 2022-07-29  51.755426  51.755426    False
210 2022-07-30  60.802663  60.802663    False
211 2022-07-31  60.228408  60.228408    False
212 2022-08-01  59.156038  59.156038    False
213 2022-08-02  30.250718  30.250718    False
214 2022-08-03  98.742603  49.183153     True
215 2022-08-04  97.644779  51.243598     True
216 2022-08-05  88.170805  88.170805    False
217 2022-08-06  24.497657  24.497657    False
218 2022-08-07  10.795125  10.795125    False
```

219	2022-08-08	53.584428	53.584428	False
220	2022-08-09	0.571233	49.720724	True
221	2022-08-10	53.644083	53.644083	False
222	2022-08-11	48.594708	48.594708	False
223	2022-08-12	95.137497	49.923548	True
224	2022-08-13	14.131521	14.131521	False
225	2022-08-14	96.516342	49.426762	True
226	2022-08-15	15.281610	15.281610	False
227	2022-08-16	66.004160	66.004160	False
228	2022-08-17	50.742419	50.742419	False
229	2022-08-18	64.908754	64.908754	False
230	2022-08-19	67.391997	67.391997	False
231	2022-08-20	70.367496	70.367496	False
232	2022-08-21	96.055725	49.987800	True
233	2022-08-22	21.486616	21.486616	False
234	2022-08-23	54.133550	54.133550	False
235	2022-08-24	8.434662	8.434662	False
236	2022-08-25	26.874015	26.874015	False
237	2022-08-26	48.312715	48.312715	False
238	2022-08-27	58.905964	58.905964	False
239	2022-08-28	89.012051	89.012051	False
240	2022-08-29	27.521546	27.521546	False
241	2022-08-30	15.027190	15.027190	False
242	2022-08-31	77.639220	77.639220	False
243	2022-09-01	86.010542	86.010542	False
244	2022-09-02	31.036460	31.036460	False
245	2022-09-03	16.460595	16.460595	False
246	2022-09-04	64.024827	64.024827	False
247	2022-09-05	90.839785	90.839785	False
248	2022-09-06	81.765810	81.765810	False
249	2022-09-07	44.089524	44.089524	False
250	2022-09-08	55.242978	55.242978	False
251	2022-09-09	11.900205	11.900205	False
252	2022-09-10	65.464448	65.464448	False
253	2022-09-11	90.506046	90.506046	False
254	2022-09-12	2.139853	50.131353	True
255	2022-09-13	65.550311	65.550311	False
256	2022-09-14	47.870391	47.870391	False
257	2022-09-15	77.915260	77.915260	False
258	2022-09-16	27.407097	27.407097	False
259	2022-09-17	75.435544	75.435544	False
260	2022-09-18	7.821601	7.821601	False
261	2022-09-19	54.038429	54.038429	False
262	2022-09-20	74.139121	74.139121	False
263	2022-09-21	13.844190	13.844190	False
264	2022-09-22	64.854796	64.854796	False
265	2022-09-23	22.149049	22.149049	False
266	2022-09-24	66.080132	66.080132	False
267	2022-09-25	49.741572	49.741572	False
268	2022-09-26	39.604586	39.604586	False
269	2022-09-27	45.916318	45.916318	False
270	2022-09-28	42.398910	42.398910	False
271	2022-09-29	23.592608	23.592608	False
272	2022-09-30	56.600043	56.600043	False
273	2022-10-01	88.262217	88.262217	False
274	2022-10-02	94.436558	50.849335	True
275	2022-10-03	50.058298	50.058298	False
276	2022-10-04	73.790069	73.790069	False
277	2022-10-05	22.525172	22.525172	False

```
278 2022-10-06  56.028784  56.028784    False
279 2022-10-07  44.068814  44.068814    False
280 2022-10-08  72.284279  72.284279    False
281 2022-10-09  41.478363  41.478363    False
282 2022-10-10  20.368036  20.368036    False
283 2022-10-11  57.763634  57.763634    False
284 2022-10-12  38.345674  38.345674    False
285 2022-10-13  37.026764  37.026764    False
286 2022-10-14  76.104135  76.104135    False
287 2022-10-15  72.979451  72.979451    False
288 2022-10-16  87.855785  87.855785    False
289 2022-10-17  45.507229  45.507229    False
290 2022-10-18  24.833716  24.833716    False
291 2022-10-19   4.127023  50.997555     True
292 2022-10-20  79.069469  79.069469    False
293 2022-10-21  66.040390  66.040390    False
294 2022-10-22  80.508589  80.508589    False
295 2022-10-23  96.829535  49.454301     True
296 2022-10-24  98.488782  51.607555     True
297 2022-10-25   0.381460  49.783209     True
298 2022-10-26  18.870677  18.870677    False
299 2022-10-27  96.403525  49.603682     True
300 2022-10-28  37.006334  37.006334    False
301 2022-10-29  32.533978  32.533978    False
302 2022-10-30  56.002566  56.002566    False
303 2022-10-31  79.137181  79.137181    False
304 2022-11-01  90.439062  90.439062    False
305 2022-11-02  63.026301  63.026301    False
306 2022-11-03  20.742761  20.742761    False
307 2022-11-04  27.622422  27.622422    False
308 2022-11-05  23.049377  23.049377    False
309 2022-11-06  89.504207  89.504207    False
310 2022-11-07  66.679998  66.679998    False
311 2022-11-08  68.195168  68.195168    False
312 2022-11-09  84.760229  84.760229    False
313 2022-11-10  66.824169  66.824169    False
314 2022-11-11  49.828728  49.828728    False
315 2022-11-12  70.999185  70.999185    False
316 2022-11-13   6.217892  51.302795     True
317 2022-11-14  90.980418  90.980418    False
318 2022-11-15  20.130901  20.130901    False
319 2022-11-16  25.812599  25.812599    False
320 2022-11-17  65.999811  65.999811    False
321 2022-11-18  20.489471  20.489471    False
322 2022-11-19  11.105462  11.105462    False
323 2022-11-20  63.688081  63.688081    False
324 2022-11-21  21.762358  21.762358    False
325 2022-11-22  94.495143  50.964296     True
326 2022-11-23  66.127296  66.127296    False
327 2022-11-24  16.065629  16.065629    False
328 2022-11-25  26.647437  26.647437    False
329 2022-11-26  26.873446  26.873446    False
330 2022-11-27   6.545430  49.231210     True
331 2022-11-28  19.229813  19.229813    False
332 2022-11-29  85.872789  85.872789    False
333 2022-11-30  62.554307  62.554307    False
334 2022-12-01  97.255045  50.600802     True
335 2022-12-02  85.064236  85.064236    False
336 2022-12-03  83.956446  83.956446    False
```

```
337 2022-12-04   6.568331  51.001133     True
338 2022-12-05  29.197188  29.197188    False
339 2022-12-06  83.488814  83.488814    False
340 2022-12-07   6.406983  51.626648     True
341 2022-12-08  86.199375  86.199375    False
342 2022-12-09  96.236849  51.237434     True
343 2022-12-10  36.186077  36.186077    False
344 2022-12-11  43.253270  43.253270    False
345 2022-12-12  88.804735  88.804735    False
346 2022-12-13  86.139464  86.139464    False
347 2022-12-14   0.226143  50.579834     True
348 2022-12-15  55.736198  55.736198    False
349 2022-12-16  16.043266  16.043266    False
350 2022-12-17  63.929723  63.929723    False
351 2022-12-18  66.022759  66.022759    False
352 2022-12-19  56.049987  56.049987    False
353 2022-12-20  86.246871  86.246871    False
354 2022-12-21  37.843312  37.843312    False
355 2022-12-22  10.194567  10.194567    False
356 2022-12-23  30.996891  30.996891    False
357 2022-12-24  83.168826  83.168826    False
358 2022-12-25  89.649694  89.649694    False
359 2022-12-26  60.096225  60.096225    False
360 2022-12-27  90.744706  90.744706    False
361 2022-12-28  19.675608  19.675608    False
362 2022-12-29  54.734194  54.734194    False
363 2022-12-30  22.710878  22.710878    False
364 2022-12-31   1.970247  49.124743     True
```

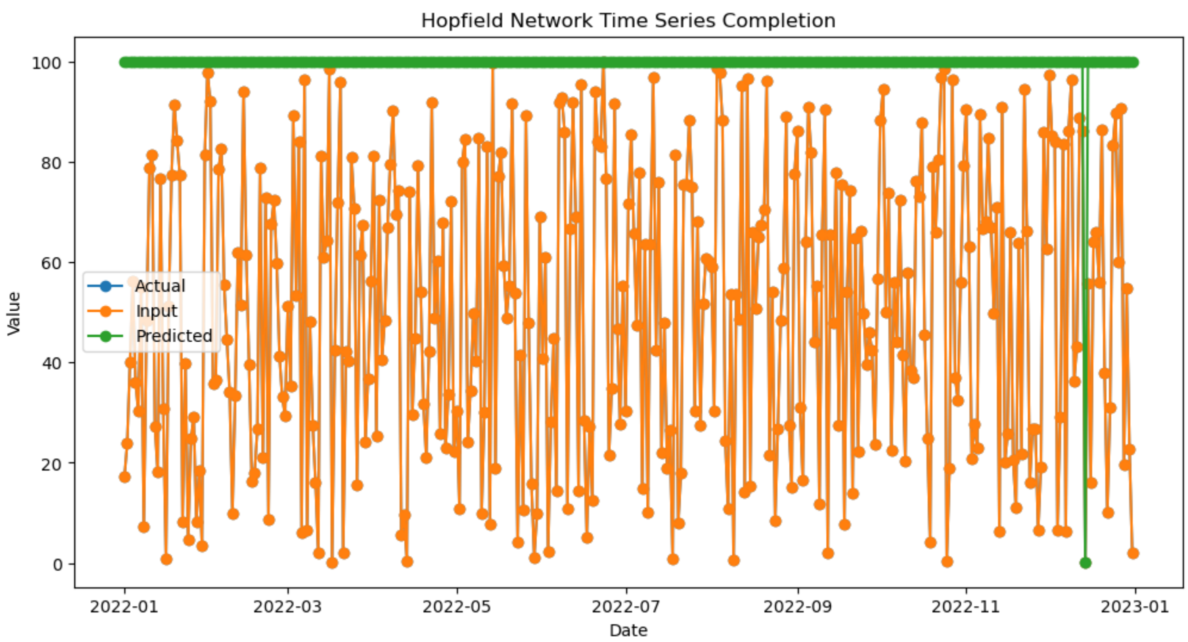


In [5]:
```
# Hopefield Network for Time-series Completion
#
# Igor Mol <igor.mol@makes.ai>
#
# Abstract:
# The approach implemented in the following program demonstrates the applica
# of the Hopfield Networks for time-series completion, offering a systematic
# framework for modeling and predicting missing values in sequential data.
#     A class named HopfieldNetwork is utilized to model and predict missing
# values in a time series. The network is initialized with a specified size,
# the weights matrix, representing the connections between neurons, is set t
# zeros initially. The training method updates these weights based on the ou
# product of input patterns, excluding self-connections to prevent distortio
#     The prediction process involves iteratively updating the output patter
# using the dot product with the weights matrix. This updated pattern is det
# mined by applying a sign function to the dot product result. The number of
# iterations is user-defined, influencing the convergence of the predicted
# pattern.
#
# "The memory has already entered your consciousness, but you must find it.
# will appear in dreams, in your waking hours, when you turn the page of a b
# or a corner. Do not be impatient, do not invent memories. Chance might fav
# or delay you, in its own mysterious way. As I begin to forget, you will be
# to remember. I promise nothing more".
# - Jorge Luis Borges

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Class `init':
# - Initializes the Hopfield Network with a specified size.
# - Sets the size of the network and initializes the weights matrix with zer

class HopfieldNetwork:

    def __init__(self, size):
        self.size = size
        self.weights = np.zeros((size, size))

# train:
# - Trains the Hopfield Network with a set of input patterns.
# - Updates the weights matrix based on the outer product of each pattern wi
# - Sets diagonal elements of the weights matrix to zero to prevent self-con

    def train(self, patterns):
        for pattern in patterns:
            self.weights += np.outer(pattern, pattern)
            np.fill_diagonal(self.weights, 0)

# predict:
# - Predicts a pattern using the trained Hopfield Network.
# - Iteratively updates the output pattern based on the dot product with the
# - Applies a sign function to the dot product result.
# - Returns the predicted pattern after a specified number of iterations.

    def predict(self, input_pattern, max_iterations=100):
        output_pattern = np.copy(input_pattern)
```

```
        for _ in range(max_iterations):
            output_pattern = np.sign(np.dot(self.weights, output_pattern))
        return output_pattern

# Two utility functions, normalize_data and denormalize_pattern, assist in p
# processing the time series. normalize_data scales the data to the [0, 1] r
# ge, while denormalize_pattern reverts a normalized pattern to its original
# scale based on the original minimum and maximum values.

def normalize_data(data):
    min_value, max_value = np.min(data), np.max(data)
    normalized_data = (data - min_value) / (max_value - min_value)
    return normalized_data, min_value, max_value

def denormalize_pattern(pattern, min_value, max_value):
    return pattern * (max_value - min_value) + min_value

# In the main function, time series data is loaded from a CSV file and subse
# quently normalized using the utility functions. A Hopfield Network instanc
# created, and the network is trained with the normalized time series values
# subset of the time series, such as the first half, is chosen as the input
# pattern.
#     The Hopfield Network is then employed to predict the remaining values
# the time series. The predicted and input patterns are denormalized to thei
# original scale, and the results are presented in a tabular format. Additic
# lly, a graphical representation is provided through a time series plot tha
# showcases the actual, input, and predicted values.

def main():
    # Load the time-series data
    file_path = "/Users/igormol/Desktop/time_series_data.csv"
    df = pd.read_csv(file_path)
    df['Date'] = pd.to_datetime(df['Date'])
    df = df.sort_values(by='Date')

    # Normalize the 'Value' column
    values = df['Value'].values
    normalized_values, min_value, max_value = normalize_data(values)

    # Create a Hopfield Network
    input_size = len(normalized_values)
    hopfield_net = HopfieldNetwork(size=input_size)

    # Train the network with the normalized values
    hopfield_net.train([normalized_values])

    # Choose a subset of the time series as input (e.g., the first half)
    input_pattern = normalized_values[:input_size]

    # Predict the remaining values
    predicted_pattern = hopfield_net.predict(input_pattern)

    # Denormalize the patterns
    input_pattern_denormalized = denormalize_pattern(input_pattern, min_valu
    predicted_pattern_denormalized = denormalize_pattern(predicted_pattern,

    # Create a DataFrame for the results
    results = pd.DataFrame({'Date': df['Date'], 'Actual': values, 'Input': i

    # Print the results in a formatted table
```

```
    # Print the results in a formatted table
    print(results)

    # Plot the actual, input, and predicted time series
    plt.figure(figsize=(12, 6))
    plt.plot(df['Date'], values, label='Actual', marker='o')
    plt.plot(df['Date'], input_pattern_denormalized, label='Input', marker='
    plt.plot(df['Date'], predicted_pattern_denormalized, label='Predicted',
    plt.title('Hopfield Network Time Series Completion')
    plt.xlabel('Date')
    plt.ylabel('Value')
    plt.legend()
    plt.show()

if __name__ == "__main__":
    main()
```

```
           Date     Actual      Input  Predicted
0    2022-01-01  17.205673  17.205673  99.965052
1    2022-01-02  23.882583  23.882583  99.965052
2    2022-01-03  39.984362  39.984362  99.965052
3    2022-01-04  56.131855  56.131855  99.965052
4    2022-01-05  36.124378  36.124378  99.965052
..          ...        ...        ...        ...
360  2022-12-27  90.744706  90.744706  99.965052
361  2022-12-28  19.675608  19.675608  99.965052
362  2022-12-29  54.734194  54.734194  99.965052
363  2022-12-30  22.710878  22.710878  99.965052
364  2022-12-31   1.970247   1.970247  99.965052

[365 rows x 4 columns]
```



In [6]:
```
# Generative Adversarial Network for Time Series Analysis
#
# Igor Mol <igor.mol@makes.ai>
#
# This program uses a Generative Adversarial Network (GAN) for time series
# analysis. The GAN consists of a generator and a discriminator. The generat
# creates synthetic time series data, and the discriminator distinguishes
# between real and generated data. The goal is to train the generator to pro
# realistic time series data.
#     The program trains the GAN by generating synthetic time series and
# updating the discriminator with real and generated data. It calculates and
# prints the discriminator and generator losses during training. The Mean
# Squared Error (MSE) between actual and generated time series is also compu
# and plotted to visualize the overall trend.
#     Finally, the program generates synthetic time series for the entire
# dataset, creates a DataFrame with actual and generated data, prints the
# results, and plots the actual and generated time series for comparison. Th
# GAN learns to generate time series data that closely resembles the real da
# demonstrating its ability to capture underlying patterns in the time serie

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, LeakyReLU, BatchNormalization, Re
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import Input
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

class TimeSeriesGAN:

# Initialization:
# The class TimeSeriesGAN is initialized with a file path (file_path) pointi
# to a CSV file with time series data and a latent dimension (latent_dim) fo
# the generator. Attributes include df (DataFrame for time series data), val
# (normalized time series values), scaler (MinMaxScaler for normalization),
# models for the generator, discriminator, and GAN.

    def __init__(self, file_path, latent_dim=100):
        self.file_path = file_path
        self.latent_dim = latent_dim
        self.df = None
        self.values = None
        self.scaler = MinMaxScaler()
        self.generator = None
        self.discriminator = None
        self.gan = None

# load_data:
# - Reads CSV data into a DataFrame.
# - Converts the 'Date' column to datetime format, sorts the DataFrame by da
# - Normalizes time series values using MinMaxScaler.

    def load_data(self):
        self.df = pd.read_csv(self.file_path)
        self.df['Date'] = pd.to_datetime(self.df['Date'])
```

```
        self.df = self.df.sort_values(by='Date')
        self.values = self.scaler.fit_transform(self.df['Value'].values.resh

# build_generator:
# - Constructs the generator model with dense layers, LeakyReLU activation,
# batch normalization.
# - The generator output has a single node with a sigmoid activation.
# - Compiles the model with binary crossentropy loss and the Adam optimizer.

    def build_generator(self):
        generator = Sequential()
        generator.add(Dense(128, input_dim=self.latent_dim))
        generator.add(LeakyReLU(alpha=0.2))
        generator.add(BatchNormalization(momentum=0.8))
        generator.add(Dense(1, activation='sigmoid'))
        generator.compile(loss='binary_crossentropy', optimizer=Adam(0.0002,
        self.generator = generator

# build_discriminator:
# - Constructs the discriminator model with a similar architecture to the ge
# - Compiles the model with binary crossentropy loss, the Adam optimizer, ar
# accuracy as a metric.

    def build_discriminator(self):
        discriminator = Sequential()
        discriminator.add(Dense(128, input_dim=1))
        discriminator.add(LeakyReLU(alpha=0.2))
        discriminator.add(Dense(1, activation='sigmoid'))
        discriminator.compile(loss='binary_crossentropy', optimizer=Adam(0.0
        self.discriminator = discriminator

# build_gan:
# - Builds the GAN model by combining the generator and discriminator.
# - Freezes discriminator weights during GAN training.
# - Compiles the GAN model with binary crossentropy loss and the Adam optimi

    def build_gan(self):
        self.discriminator.trainable = False
        gan_input = Input(shape=(self.latent_dim,))
        x = self.generator(gan_input)
        gan_output = self.discriminator(x)
        gan = Model(gan_input, gan_output)
        gan.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))
        self.gan = gan

# train_gan:
# - Trains the GAN for a specified number of epochs.
# - Generates synthetic time series, updates the discriminator with real anc
# - Calculates and prints discriminator and generator losses at intervals.
# - Computes Mean Squared Error (MSE) between actual and generated time seri

    def train_gan(self, epochs=50, batch_size=64, sample_interval=1000):
        mse_history = []

        for epoch in range(epochs):
            noise = np.random.normal(0, 1, size=(batch_size, self.latent_dim
            generated_series = self.generator.predict(noise)

            idx = np.random.randint(0, self.values.shape[0], batch_size)
            real_series = self.values[idx]
```

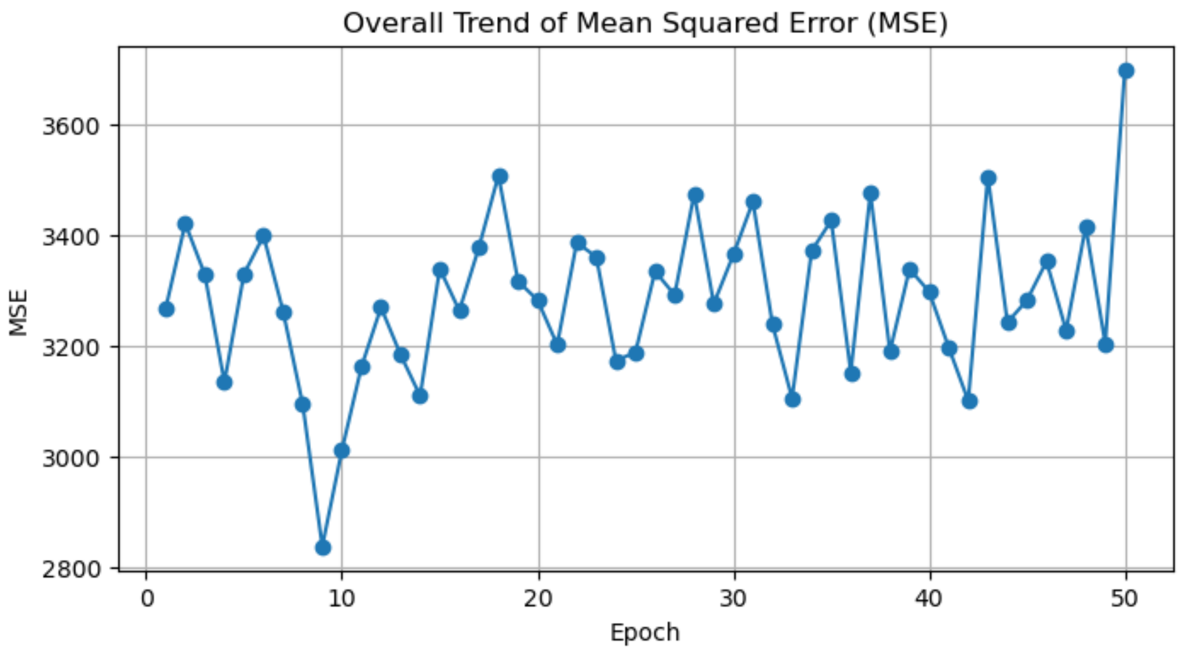
```
            real_series = self.values[idx]

            labels_real = np.ones((batch_size, 1))
            labels_fake = np.zeros((batch_size, 1))

            d_loss_real = self.discriminator.train_on_batch(real_series, lab
            d_loss_fake = self.discriminator.train_on_batch(generated_series
            d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

            noise = np.random.normal(0, 1, size=(batch_size, self.latent_dim
            labels_gan = np.ones((batch_size, 1))

            g_loss = self.gan.train_on_batch(noise, labels_gan)

            if epoch % sample_interval == 0:
                print(f"Epoch {epoch}, [D loss: {d_loss[0]} | D accuracy: {1

            # Calculate MSE and store for plotting
            generated_series = self.generate_synthetic_series(num_samples=se
            mse = mean_squared_error(self.values, generated_series)
            mse_history.append(mse)

        # Plot the overall trend of MSE
        self.plot_mse_trend(mse_history)

# generate_synthetic_series:
# Generates synthetic time series using the trained generator.

    def generate_synthetic_series(self, num_samples):
        noise = np.random.normal(0, 1, size=(num_samples, self.latent_dim))
        generated_series = self.generator.predict(noise)
        return self.scaler.inverse_transform(generated_series)

# plot_mse_trend method:
# Plots the overall trend of Mean Squared Error (MSE) during GAN training.

    def plot_mse_trend(self, mse_history):
        plt.figure(figsize=(8, 4))
        plt.plot(range(1, len(mse_history) + 1), mse_history, marker='o')
        plt.title('Overall Trend of Mean Squared Error (MSE)')
        plt.xlabel('Epoch')
        plt.ylabel('MSE')
        plt.grid(True)
        plt.show()

# visualize_results:
# - Generates synthetic time series for the entire dataset.
# - Creates a DataFrame with actual and generated time series.
# - Prints results in a formatted table.
# - Plots the actual and generated time series.

    def visualize_results(self):
        # Generate synthetic time series
        generated_series = self.generate_synthetic_series(num_samples=self.v

        # Create a DataFrame for the results
        results = pd.DataFrame({'Date': self.df['Date'], 'Actual': self.df['

        # Print the results in a formatted table
        pd.set_option('display.max_rows', None)
```

```
        pd.set_option('display.max_rows', None)
        pd.set_option('display.max_columns', None)
        print(results)
        pd.reset_option('display.max_rows')
        pd.reset_option('display.max_columns')

        # Plot the actual and generated time series
        plt.figure(figsize=(12, 6))
        plt.plot(self.df['Date'], self.df['Value'], label='Actual', marker='
        plt.plot(self.df['Date'], generated_series, label='Generated', marke
        plt.title('Generative Adversarial Network Time Series Generation')
        plt.xlabel('Date')
        plt.ylabel('Value')
        plt.legend()
        plt.show()

# Main function:
# - Creates an instance of TimeSeriesGAN.
# - Loads data, builds generator, discriminator, and GAN models.
# - Trains the GAN and visualizes the results.

def main():
    gan_model = TimeSeriesGAN(file_path="/Users/igormol/Desktop/time_series_
    gan_model.load_data()
    gan_model.build_generator()
    gan_model.build_discriminator()
    gan_model.build_gan()
    gan_model.train_gan()
    gan_model.visualize_results()

if __name__ == "__main__":
    main()
```

```
2/2 [==============================] - 0s 6ms/step
Epoch 0, [D loss: 0.6939854621887207 | D accuracy: 49.21875] [G loss: 0.659
6560478210449]
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 5ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 10ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 10ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 8ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 5ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 5ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 4ms/step
2/2 [==============================] - 0s 12ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
12/12 [==============================] - 0s 4ms/step
2/2 [==============================] - 0s 8ms/step
```

```
12/12 [==============================] - 0s 4ms/step
2/2 [==============================] - 0s 5ms/step
12/12 [==============================] - 0s 7ms/step
2/2 [==============================] - 0s 10ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 5ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 5ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 5ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 14ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 5ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 5ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 5ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 5ms/step
12/12 [==============================] - 0s 2ms/step
```

```
2/2 [==============================] - 0s 3ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
12/12 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 5ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 5ms/step
12/12 [==============================] - 0s 2ms/step
```



```
12/12 [==============================] - 0s 2ms/step
          Date     Actual  Generated
0   2022-01-01  17.205673  49.800240
1   2022-01-02  23.882583  16.537466
2   2022-01-03  39.984362  48.451469
3   2022-01-04  56.131855  88.634865
4   2022-01-05  36.124378  36.638332
5   2022-01-06  30.250211  85.227165
6   2022-01-07  55.043937  57.186935
7   2022-01-08   7.400941  50.365200
8   2022-01-09  48.476184  75.390923
9   2022-01-10  78.648685  31.032705
10  2022-01-11  81.300903  62.528423
11  2022-01-12  27.328824  51.555546
12  2022-01-13  18.239756  63.598690
13  2022-01-14  76.547443  33.877220
14  2022-01-15  30.689008  16.579840
15  2022-01-16   0.958607  11.541868
16  2022-01-17  51.138039  35.771233
17  2022-01-18  77.246803  82.725624
18  2022-01-19  91.421612  82.669182
19  2022-01-20  84.154631  22.520220
20  2022-01-21  77.347057  23.291582
21  2022-01-22   8.236078  51.780659
22  2022-01-23  39.757374  45.650879
23  2022-01-24   4.565118  22.606630
24  2022-01-25  24.750312  62.208752
25  2022-01-26  29.149634  60.931515
26  2022-01-27   8.272093  22.139700
27  2022-01-28  18.452323  33.495975
28  2022-01-29   3.472170  67.064156
29  2022-01-30  81.266011  87.413353
30  2022-01-31  97.880334  74.390205
31  2022-02-01  92.026172  30.531731
32  2022-02-02  35.802006  27.809072
33  2022-02-03  36.395385  67.125443
34  2022-02-04  78.625231  38.122334
35  2022-02-05  82.484448  96.989784
36  2022-02-06  55.560241  28.647539
37  2022-02-07  44.502025  61.563831
38  2022-02-08  34.140065  95.024567
39  2022-02-09   9.914127  79.460060
40  2022-02-10  33.471826  20.611679
41  2022-02-11  61.985206  92.517250
42  2022-02-12  51.508424  52.499283
43  2022-02-13  93.994475  62.318027
44  2022-02-14  61.489319  38.653378
45  2022-02-15  39.644319  30.363766
46  2022-02-16  16.253762  96.104340
47  2022-02-17  17.990712  26.175289
48  2022-02-18  26.656962  51.835094
49  2022-02-19  78.792661  83.903267
50  2022-02-20  20.987891  52.886089
51  2022-02-21  72.791894   3.033281
52  2022-02-22   8.743812  25.124331
53  2022-02-23  67.491099   3.579738
54  2022-02-24  72.393459  55.775436
55  2022-02-25  59.699182  84.896713
56  2022-02-26  41.311351  76.120247
```

57	2022-02-27	33.093305	59.795235
58	2022-02-28	29.380650	60.702053
59	2022-03-01	51.249703	40.397522
60	2022-03-02	35.362935	9.136670
61	2022-03-03	89.118747	72.061089
62	2022-03-04	53.392011	51.711994
63	2022-03-05	84.081062	9.558528
64	2022-03-06	6.073966	52.826019
65	2022-03-07	96.241722	64.059975
66	2022-03-08	6.510861	61.354145
67	2022-03-09	48.224867	83.589920
68	2022-03-10	27.549201	55.100117
69	2022-03-11	16.177437	65.174835
70	2022-03-12	2.087524	61.820213
71	2022-03-13	81.039423	81.271797
72	2022-03-14	60.843868	41.497524
73	2022-03-15	64.328461	91.682686
74	2022-03-16	98.581993	9.510833
75	2022-03-17	0.229440	85.166740
76	2022-03-18	42.466619	87.667526
77	2022-03-19	71.991242	59.004185
78	2022-03-20	95.856792	47.659664
79	2022-03-21	2.046130	43.391361
80	2022-03-22	42.088354	90.191414
81	2022-03-23	40.274422	68.269310
82	2022-03-24	80.872092	56.235546
83	2022-03-25	70.789452	77.351402
84	2022-03-26	15.641700	67.425240
85	2022-03-27	61.421355	49.459251
86	2022-03-28	67.323260	30.003927
87	2022-03-29	24.033506	86.146301
88	2022-03-30	36.818106	47.130772
89	2022-03-31	56.231732	64.975044
90	2022-04-01	81.227223	11.274142
91	2022-04-02	25.365495	61.774990
92	2022-04-03	72.344287	79.617920
93	2022-04-04	40.526373	44.411476
94	2022-04-05	48.293645	61.316341
95	2022-04-06	66.780186	75.650116
96	2022-04-07	79.509808	91.488060
97	2022-04-08	90.226501	40.500984
98	2022-04-09	69.436928	92.935844
99	2022-04-10	74.245766	55.068535
100	2022-04-11	5.624580	14.703361
101	2022-04-12	9.581872	93.304588
102	2022-04-13	0.480818	58.147022
103	2022-04-14	74.075054	49.252026
104	2022-04-15	29.532013	30.152332
105	2022-04-16	44.919033	18.706335
106	2022-04-17	79.318710	24.179855
107	2022-04-18	53.955188	95.286980
108	2022-04-19	31.693897	63.747437
109	2022-04-20	21.070292	75.362305
110	2022-04-21	42.114909	96.938843
111	2022-04-22	91.878629	19.530836
112	2022-04-23	48.940667	67.060715
113	2022-04-24	60.198965	94.254723
114	2022-04-25	25.817230	85.612312
115	2022-04-26	67.865390	25.469641

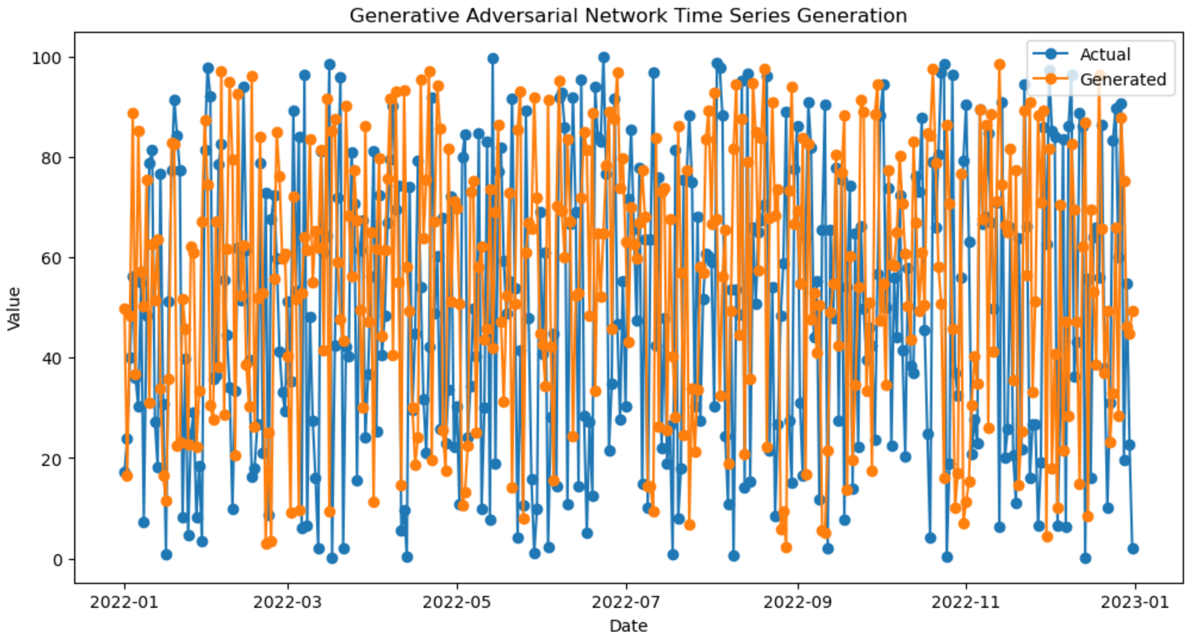
```
116 2022-04-27  22.904959  17.611818
117 2022-04-28  33.719957  81.667336
118 2022-04-29  72.004106  51.274967
119 2022-04-30  22.161655  71.265724
120 2022-05-01  30.435711  69.691444
121 2022-05-02  10.780253  50.823177
122 2022-05-03  79.922188  10.607021
123 2022-05-04  84.526871  13.200974
124 2022-05-05  24.151776  22.551432
125 2022-05-06  34.466848  73.008247
126 2022-05-07  49.865918  75.284203
127 2022-05-08  40.194334  25.010769
128 2022-05-09  84.583993  58.197372
129 2022-05-10   9.901063  62.118740
130 2022-05-11  29.988062  43.669418
131 2022-05-12  83.091482  45.727844
132 2022-05-13   7.749104  73.440491
133 2022-05-14  99.619453  41.996181
134 2022-05-15  18.921870  69.021065
135 2022-05-16  76.994205  86.423111
136 2022-05-17  81.927838  47.244160
137 2022-05-18  59.360297  31.167463
138 2022-05-19  48.937226  52.379337
139 2022-05-20  55.295679  72.870796
140 2022-05-21  91.559731  14.219826
141 2022-05-22  53.731649  50.819347
142 2022-05-23   4.133606  85.429977
143 2022-05-24  41.550117  93.119377
144 2022-05-25  10.635445   8.018489
145 2022-05-26  89.291472  61.013157
146 2022-05-27  47.844574  66.915466
147 2022-05-28  15.793790  65.649719
148 2022-05-29   1.174022  91.934235
149 2022-05-30   9.978596  71.861031
150 2022-05-31  69.128886  44.798916
151 2022-06-01  40.883477  42.748714
152 2022-06-02  60.864794  34.254864
153 2022-06-03   2.242390  91.435532
154 2022-06-04  28.208840  42.195198
155 2022-06-05  44.777027  15.614402
156 2022-06-06  14.395065  70.283333
157 2022-06-07  91.820033  95.057961
158 2022-06-08  92.690335  69.259529
159 2022-06-09  85.810992  59.896698
160 2022-06-10  10.866967  83.536354
161 2022-06-11  66.623362  67.214890
162 2022-06-12  91.739415  24.374531
163 2022-06-13  69.079676  52.495247
164 2022-06-14  14.473059  52.927731
165 2022-06-15  95.460439  71.888603
166 2022-06-16  28.527694  84.966782
167 2022-06-17   5.068678  81.453682
168 2022-06-18  27.289601  48.261009
169 2022-06-19  12.604815  88.743324
170 2022-06-20  93.850376  33.322460
171 2022-06-21  83.929756  64.754547
172 2022-06-22  82.972703  52.241695
173 2022-06-23  99.965052  64.810310
174 2022-06-24  76.588754  78.283920
```

175	2022-06-25	21.451446	88.913651
176	2022-06-26	34.945709	45.691322
177	2022-06-27	91.701902	87.578247
178	2022-06-28	46.681014	96.841064
179	2022-06-29	27.613734	73.777840
180	2022-06-30	55.322381	79.766251
181	2022-07-01	30.284241	63.056709
182	2022-07-02	71.565274	43.256210
183	2022-07-03	85.340038	69.920204
184	2022-07-04	65.666911	62.564259
185	2022-07-05	47.355871	59.782726
186	2022-07-06	77.902354	66.899849
187	2022-07-07	14.926307	77.348343
188	2022-07-08	63.613576	68.059952
189	2022-07-09	10.105969	14.475846
190	2022-07-10	63.447903	14.403792
191	2022-07-11	96.893859	9.472464
192	2022-07-12	42.535514	83.844780
193	2022-07-13	75.950996	26.371437
194	2022-07-14	21.891973	72.979759
195	2022-07-15	47.784744	73.802177
196	2022-07-16	18.836006	25.687510
197	2022-07-17	26.556727	67.584648
198	2022-07-18	0.781499	40.368828
199	2022-07-19	81.340398	28.294786
200	2022-07-20	8.014817	86.054657
201	2022-07-21	18.067338	56.924633
202	2022-07-22	75.510877	24.659950
203	2022-07-23	75.386231	77.252495
204	2022-07-24	88.317835	6.797021
205	2022-07-25	74.849072	33.802498
206	2022-07-26	30.258284	21.309393
207	2022-07-27	68.031887	33.568920
208	2022-07-28	27.429936	58.081551
209	2022-07-29	51.755426	56.882317
210	2022-07-30	60.802663	83.542992
211	2022-07-31	60.228408	89.156433
212	2022-08-01	59.156038	66.590401
213	2022-08-02	30.250718	92.751884
214	2022-08-03	98.742603	67.571426
215	2022-08-04	97.644779	32.359211
216	2022-08-05	88.170805	56.209759
217	2022-08-06	24.497657	65.537437
218	2022-08-07	10.795125	18.804476
219	2022-08-08	53.584428	49.306515
220	2022-08-09	0.571233	81.635391
221	2022-08-10	53.644083	94.521950
222	2022-08-11	48.594708	44.474583
223	2022-08-12	95.137497	87.594795
224	2022-08-13	14.131521	20.777719
225	2022-08-14	96.516342	79.099281
226	2022-08-15	15.281610	35.899281
227	2022-08-16	66.004160	94.621475
228	2022-08-17	50.742419	84.877968
229	2022-08-18	64.908754	57.325573
230	2022-08-19	67.391997	83.758377
231	2022-08-20	70.367496	97.505974
232	2022-08-21	96.055725	22.355087
233	2022-08-22	21.486616	67.766182

```
234 2022-08-23  54.133550  90.800613
235 2022-08-24   8.434662  68.254440
236 2022-08-25  26.874015  73.560760
237 2022-08-26  48.312715   5.831961
238 2022-08-27  58.905964   9.348370
239 2022-08-28  89.012051   2.232037
240 2022-08-29  27.521546  73.292076
241 2022-08-30  15.027190  93.975746
242 2022-08-31  77.639220  66.754517
243 2022-09-01  86.010542  69.304863
244 2022-09-02  31.036460  54.736973
245 2022-09-03  16.460595  83.658722
246 2022-09-04  64.024827  16.805887
247 2022-09-05  90.839785  82.514153
248 2022-09-06  81.765810  47.565849
249 2022-09-07  44.089524  52.940697
250 2022-09-08  55.242978  40.925217
251 2022-09-09  11.900205  50.472630
252 2022-09-10  65.464448   5.654585
253 2022-09-11  90.506046   5.151480
254 2022-09-12   2.139853  21.539721
255 2022-09-13  65.550311  49.182034
256 2022-09-14  47.870391  54.671959
257 2022-09-15  77.915260  80.320282
258 2022-09-16  27.407097  42.451965
259 2022-09-17  75.435544  76.979874
260 2022-09-18   7.821601  88.154022
261 2022-09-19  54.038429  13.746394
262 2022-09-20  74.139121  60.223740
263 2022-09-21  13.844190  19.578091
264 2022-09-22  64.854796  34.696388
265 2022-09-23  22.149049  54.121655
266 2022-09-24  66.080132  91.367462
267 2022-09-25  49.741572  88.875481
268 2022-09-26  39.604586  33.316277
269 2022-09-27  45.916318  50.915619
270 2022-09-28  42.398910  17.383091
271 2022-09-29  23.592608  88.512291
272 2022-09-30  56.600043  94.471298
273 2022-10-01  88.262217  47.391930
274 2022-10-02  94.436558  54.784157
275 2022-10-03  50.058298  34.596947
276 2022-10-04  73.790069  77.423073
277 2022-10-05  22.525172  58.498375
278 2022-10-06  56.028784  58.247196
279 2022-10-07  44.068814  65.013290
280 2022-10-08  72.284279  80.264244
281 2022-10-09  41.478363  70.673195
282 2022-10-10  20.368036  60.637878
283 2022-10-11  57.763634  50.292755
284 2022-10-12  38.345674  43.594810
285 2022-10-13  37.026764  82.927109
286 2022-10-14  76.104135  66.940628
287 2022-10-15  72.979451  49.230331
288 2022-10-16  87.855785  60.986221
289 2022-10-17  45.507229  50.573925
290 2022-10-18  24.833716  84.787933
291 2022-10-19   4.127023  84.198769
292 2022-10-20  79.069469  97.481133
```

293	2022-10-21	66.040390	78.838089
294	2022-10-22	80.508589	58.127708
295	2022-10-23	96.829535	50.815372
296	2022-10-24	98.488782	16.115910
297	2022-10-25	0.381460	86.443779
298	2022-10-26	18.870677	70.731956
299	2022-10-27	96.403525	45.786827
300	2022-10-28	37.006334	10.196501
301	2022-10-29	32.533978	17.044531
302	2022-10-30	56.002566	76.639343
303	2022-10-31	79.137181	7.135691
304	2022-11-01	90.439062	11.264929
305	2022-11-02	63.026301	15.408127
306	2022-11-03	20.742761	30.590427
307	2022-11-04	27.622422	40.190701
308	2022-11-05	23.049377	34.812077
309	2022-11-06	89.504207	89.539116
310	2022-11-07	66.679998	67.689934
311	2022-11-08	68.195168	86.302956
312	2022-11-09	84.760229	25.957832
313	2022-11-10	66.824169	88.617622
314	2022-11-11	49.828728	41.253410
315	2022-11-12	70.999185	71.181099
316	2022-11-13	6.217892	98.551468
317	2022-11-14	90.980418	74.456245
318	2022-11-15	20.130901	66.449074
319	2022-11-16	25.812599	65.024742
320	2022-11-17	65.999811	81.685677
321	2022-11-18	20.489471	35.509075
322	2022-11-19	11.105462	77.429916
323	2022-11-20	63.688081	14.702118
324	2022-11-21	21.762358	25.255146
325	2022-11-22	94.495143	89.214272
326	2022-11-23	66.127296	56.498379
327	2022-11-24	16.065629	90.861809
328	2022-11-25	26.647437	33.189606
329	2022-11-26	26.873446	51.192642
330	2022-11-27	6.545430	88.261078
331	2022-11-28	19.229813	71.030731
332	2022-11-29	85.872789	89.326828
333	2022-11-30	62.554307	4.497389
334	2022-12-01	97.255045	81.701591
335	2022-12-02	85.064236	17.933731
336	2022-12-03	83.956446	40.796341
337	2022-12-04	6.568331	10.126137
338	2022-12-05	29.197188	70.436874
339	2022-12-06	83.488814	21.592373
340	2022-12-07	6.406983	47.178204
341	2022-12-08	86.199375	28.414846
342	2022-12-09	96.236849	82.511429
343	2022-12-10	36.186077	69.477013
344	2022-12-11	43.253270	47.229946
345	2022-12-12	88.804735	14.840010
346	2022-12-13	86.139464	62.232296
347	2022-12-14	0.226143	86.815468
348	2022-12-15	55.736198	8.415585
349	2022-12-16	16.043266	69.589890
350	2022-12-17	63.929723	53.212433
351	2022-12-18	66.022759	38.685059

```
352 2022-12-19  56.049987  96.388092
353 2022-12-20  86.246871  65.715240
354 2022-12-21  37.843312  36.999062
355 2022-12-22  10.194567  49.346527
356 2022-12-23  30.996891  23.302330
357 2022-12-24  83.168826  32.867004
358 2022-12-25  89.649694  65.941399
359 2022-12-26  60.096225  28.513498
360 2022-12-27  90.744706  87.786400
361 2022-12-28  19.675608  75.239929
362 2022-12-29  54.734194  46.172192
363 2022-12-30  22.710878  44.772099
364 2022-12-31   1.970247  49.343609
```



In [8]:
```
# Deep Belief Network of Type CNN/LSTM for Time Series Analysis
#
# Igor Mol <igor.mol@makes.ai>
#
# The following program implements a Deep Belief Network (DBN) for time seri
# prediction using a combination of Convolutional Neural Network (CNN) and L
# Short-Term Memory (LSTM) layers. The DBNModel class is defined to encapsul
# the architecture and functionality of the model. The constructor initializ
# parameters such as sequence length, CNN filters, CNN kernel size, and LSTM
# The build_model method assembles the DBN architecture using a Sequential m
# from the Keras library. It includes a 1D Convolutional layer with ReLU act
# tion, a MaxPooling layer for down-sampling, an LSTM layer with ReLU activa
# and a Dense output layer. The model is compiled with the Adam optimizer an
# squared error loss.
#     The main function, designated by the if name == "main": block, orchest
# the overall process. It loads time series data from a CSV file, preprocess
# by scaling the values using Min-Max normalization, and structures the data
# input sequences and target values. The DBN model is instantiated, trained
# training set, and then used to predict the target values on the test set.
# predictions are denormalized to their original scale using the Min-Max sca
# Two functions, print_table and plot_actual_vs_predicted, are defined to vi
# assess the model's performance. print_table generates a tabulated display
# versus predicted values, while plot_actual_vs_predicted produces a time se
# comparing the actual and predicted values.
# In summary, the code demonstrates the construction and utilization of a DE
# for time series prediction, specifically tailored to sequences with a give
# The integration of CNN and LSTM layers allows the model to capture both lc
# patterns through convolutional operations and long-term dependencies throu
# recurrent connections. The script culminates in the presentation of the mc
# predictions, enabling an evaluation of its effectiveness in capturing the
# patterns within the time series data.

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tabulate import tabulate
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM, Dense

# Define a class for the Deep Belief Network (DBN) model
class DBNModel:
    def __init__(self, sequence_length, cnn_filters=64, cnn_kernel_size=3, l
        # Initialize model parameters
        self.sequence_length = sequence_length
        self.cnn_filters = cnn_filters
        self.cnn_kernel_size = cnn_kernel_size
        self.lstm_units = lstm_units
        # Build the DBN model upon instantiation
        self.model = self.build_model()

    # Define a method to build the DBN model
    def build_model(self):
        model = Sequential()
        # Add a 1D Convolutional layer with ReLU activation
        model.add(Conv1D(filters=self.cnn_filters, kernel_size=self.cnn_kern
                         input_shape=(self.sequence_length, 1)))
        # Add a MaxPooling layer for down-sampling
```

```
        model.add(MaxPooling1D(pool_size=2))
        # Add an LSTM layer with ReLU activation
        model.add(LSTM(self.lstm_units, activation='relu'))
        # Add a Dense output layer
        model.add(Dense(1))
        # Compile the model using Adam optimizer and mean squared error loss
        model.compile(optimizer='adam', loss='mean_squared_error')
        return model

    # Define a method to train the DBN model
    def train(self, X_train, y_train, epochs=50, batch_size=32):
        self.model.fit(X_train, y_train, epochs=epochs, batch_size=batch_siz

    # Define a method to make predictions using the trained model
    def predict(self, X_test):
        return self.model.predict(X_test)

# Define a function to print a table of actual versus predicted values
def print_table(actual, predicted):
    table = pd.DataFrame({
        'Actual': actual.flatten(),
        'Predicted': predicted.flatten()
    })
    print(tabulate(table, headers='keys', tablefmt='fancy_grid'))

# Define a function to plot the actual versus predicted time series
def plot_actual_vs_predicted(actual, predicted):
    plt.figure(figsize=(10, 6))
    plt.plot(actual, label='Actual')
    plt.plot(predicted, label='Predicted')
    plt.legend()
    plt.title('Actual vs. Predicted Time Series')
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.show()

# Define the main function to execute the DBN model on time series data
def main():
    # Load and preprocess the time series data
    file_path = "/Users/igormol/Desktop/time_series_data.csv"  # Update with
    time_series_df = pd.read_csv(file_path)
    time_series_df['Date'] = pd.to_datetime(time_series_df['Date'])
    values = time_series_df['Value'].values.reshape(-1, 1)
    scaler = MinMaxScaler()
    values_scaled = scaler.fit_transform(values)

    # Create input sequences and target values
    sequence_length = 10
    X, y = [], []
    for i in range(len(values_scaled) - sequence_length):
        X.append(values_scaled[i:i + sequence_length])
        y.append(values_scaled[i + sequence_length])

    X = np.array(X)
    y = np.array(y)

    # Split the data into training and testing sets
    train_size = int(len(values_scaled) * 0.8)
    X_train, X_test, y_train, y_test = X[:train_size], X[train_size:], y[:tr
```

```
    # Create and train the DBN model
    dbn_model = DBNModel(sequence_length)
    dbn_model.train(X_train, y_train)

    # Make predictions on the test set
    y_pred = dbn_model.predict(X_test)

    # Denormalize the predictions and actual values
    y_pred_denormalized = scaler.inverse_transform(y_pred)
    y_test_denormalized = scaler.inverse_transform(y_test)

    # Print the table of predicted versus actual values
    print_table(y_test_denormalized, y_pred_denormalized)

    # Plot the actual versus predicted time series
    plot_actual_vs_predicted(y_test_denormalized, y_pred_denormalized)

# Execute the main function if the script is run directly
if __name__ == "__main__":
    main()
```

```
Epoch 1/50
10/10 - 3s - loss: 0.2233 - 3s/epoch - 277ms/step
Epoch 2/50
10/10 - 0s - loss: 0.1140 - 101ms/epoch - 10ms/step
Epoch 3/50
10/10 - 0s - loss: 0.0982 - 101ms/epoch - 10ms/step
Epoch 4/50
10/10 - 0s - loss: 0.0927 - 70ms/epoch - 7ms/step
Epoch 5/50
10/10 - 0s - loss: 0.0947 - 70ms/epoch - 7ms/step
Epoch 6/50
10/10 - 0s - loss: 0.0931 - 79ms/epoch - 8ms/step
Epoch 7/50
10/10 - 0s - loss: 0.0930 - 96ms/epoch - 10ms/step
Epoch 8/50
10/10 - 0s - loss: 0.0923 - 88ms/epoch - 9ms/step
Epoch 9/50
10/10 - 0s - loss: 0.0906 - 76ms/epoch - 8ms/step
Epoch 10/50
10/10 - 0s - loss: 0.0899 - 67ms/epoch - 7ms/step
Epoch 11/50
10/10 - 0s - loss: 0.0896 - 66ms/epoch - 7ms/step
Epoch 12/50
10/10 - 0s - loss: 0.0894 - 65ms/epoch - 7ms/step
Epoch 13/50
10/10 - 0s - loss: 0.0886 - 65ms/epoch - 7ms/step
Epoch 14/50
10/10 - 0s - loss: 0.0883 - 69ms/epoch - 7ms/step
Epoch 15/50
10/10 - 0s - loss: 0.0880 - 76ms/epoch - 8ms/step
Epoch 16/50
10/10 - 0s - loss: 0.0873 - 73ms/epoch - 7ms/step
Epoch 17/50
10/10 - 0s - loss: 0.0868 - 77ms/epoch - 8ms/step
Epoch 18/50
10/10 - 0s - loss: 0.0865 - 71ms/epoch - 7ms/step
Epoch 19/50
10/10 - 0s - loss: 0.0868 - 67ms/epoch - 7ms/step
```

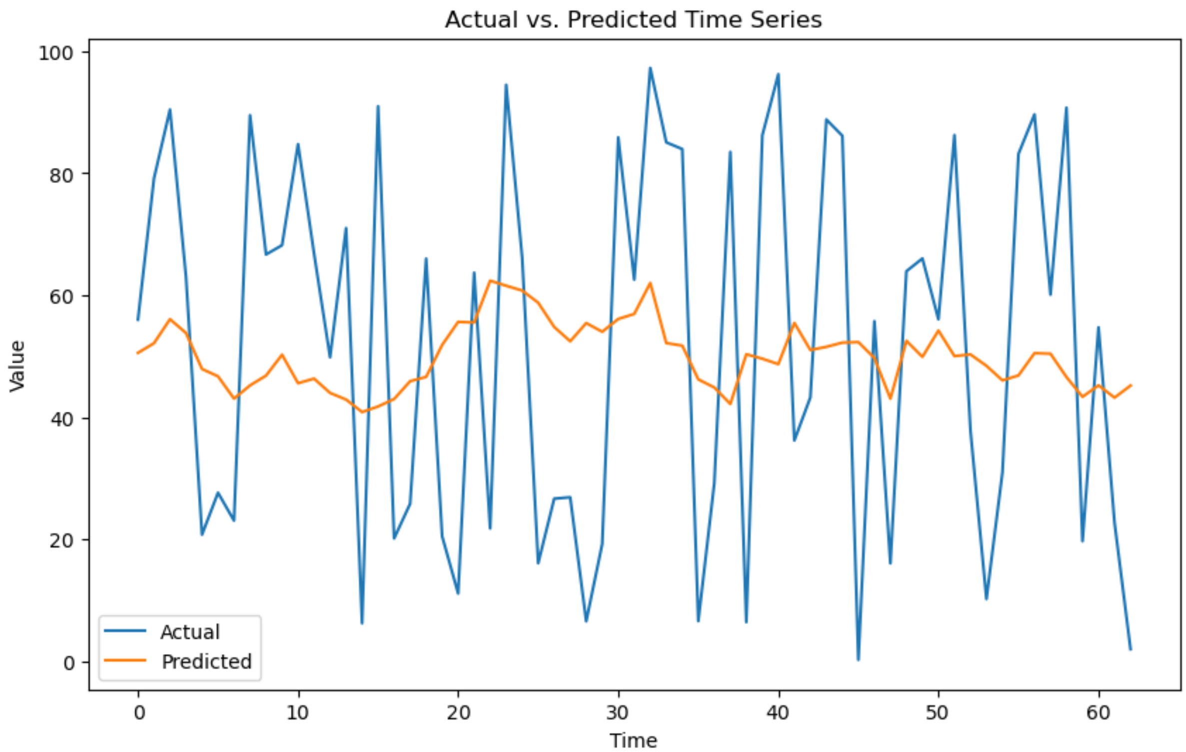
```
Epoch 20/50
10/10 - 0s - loss: 0.0857 - 73ms/epoch - 7ms/step
Epoch 21/50
10/10 - 0s - loss: 0.0859 - 73ms/epoch - 7ms/step
Epoch 22/50
10/10 - 0s - loss: 0.0848 - 68ms/epoch - 7ms/step
Epoch 23/50
10/10 - 0s - loss: 0.0855 - 65ms/epoch - 7ms/step
Epoch 24/50
10/10 - 0s - loss: 0.0867 - 67ms/epoch - 7ms/step
Epoch 25/50
10/10 - 0s - loss: 0.0859 - 70ms/epoch - 7ms/step
Epoch 26/50
10/10 - 0s - loss: 0.0846 - 69ms/epoch - 7ms/step
Epoch 27/50
10/10 - 0s - loss: 0.0838 - 68ms/epoch - 7ms/step
Epoch 28/50
10/10 - 0s - loss: 0.0843 - 68ms/epoch - 7ms/step
Epoch 29/50
10/10 - 0s - loss: 0.0829 - 59ms/epoch - 6ms/step
Epoch 30/50
10/10 - 0s - loss: 0.0826 - 65ms/epoch - 6ms/step
Epoch 31/50
10/10 - 0s - loss: 0.0828 - 72ms/epoch - 7ms/step
Epoch 32/50
10/10 - 0s - loss: 0.0824 - 68ms/epoch - 7ms/step
Epoch 33/50
10/10 - 0s - loss: 0.0829 - 65ms/epoch - 6ms/step
Epoch 34/50
10/10 - 0s - loss: 0.0828 - 90ms/epoch - 9ms/step
Epoch 35/50
10/10 - 0s - loss: 0.0820 - 75ms/epoch - 8ms/step
Epoch 36/50
10/10 - 0s - loss: 0.0821 - 72ms/epoch - 7ms/step
Epoch 37/50
10/10 - 0s - loss: 0.0830 - 88ms/epoch - 9ms/step
Epoch 38/50
10/10 - 0s - loss: 0.0834 - 81ms/epoch - 8ms/step
Epoch 39/50
10/10 - 0s - loss: 0.0835 - 67ms/epoch - 7ms/step
Epoch 40/50
10/10 - 0s - loss: 0.0811 - 66ms/epoch - 7ms/step
Epoch 41/50
10/10 - 0s - loss: 0.0852 - 66ms/epoch - 7ms/step
Epoch 42/50
10/10 - 0s - loss: 0.0841 - 59ms/epoch - 6ms/step
Epoch 43/50
10/10 - 0s - loss: 0.0807 - 66ms/epoch - 7ms/step
Epoch 44/50
10/10 - 0s - loss: 0.0808 - 67ms/epoch - 7ms/step
Epoch 45/50
10/10 - 0s - loss: 0.0803 - 62ms/epoch - 6ms/step
Epoch 46/50
10/10 - 0s - loss: 0.0804 - 67ms/epoch - 7ms/step
Epoch 47/50
10/10 - 0s - loss: 0.0809 - 62ms/epoch - 6ms/step
Epoch 48/50
10/10 - 0s - loss: 0.0804 - 68ms/epoch - 7ms/step
Epoch 49/50
```

```
10/10 - 0s - loss: 0.0805 - 65ms/epoch - 7ms/step
Epoch 50/50
10/10 - 0s - loss: 0.0800 - 64ms/epoch - 6ms/step
2/2 [==============================] - 0s 7ms/step
```

	Actual	Predicted
0	56.0026	50.5516
1	79.1372	52.1289
2	90.4391	56.0663
3	63.0263	53.8065
4	20.7428	47.9072
5	27.6224	46.7001
6	23.0494	43.0468
7	89.5042	45.2131
8	66.68	46.8209
9	68.1952	50.2453
10	84.7602	45.5733
11	66.8242	46.3247
12	49.8287	43.9784
13	70.9992	42.8929
14	6.21789	40.8426
15	90.9804	41.7607
16	20.1309	42.9813
17	25.8126	45.903
18	65.9998	46.6081
19	20.4895	51.8442
20	11.1055	55.6317
21	63.6881	55.5246
22	21.7624	62.3814
23	94.4951	61.552
24	66.1273	60.7661
25	16.0656	58.769

26	26.6474	54.7973
27	26.8734	52.4453
28	6.54543	55.4303
29	19.2298	54.0097
30	85.8728	56.1069
31	62.5543	56.9306
32	97.255	62.0095
33	85.0642	52.1556
34	83.9564	51.7329
35	6.56833	46.1971
36	29.1972	44.8752
37	83.4888	42.1726
38	6.40698	50.3183
39	86.1994	49.5917
40	96.2368	48.6987
41	36.1861	55.4553
42	43.2533	51.0158
43	88.8047	51.5362
44	86.1395	52.2289
45	0.226143	52.3123
46	55.7362	49.7131
47	16.0433	43.0405
48	63.9297	52.5211
49	66.0228	49.8927
50	56.05	54.2108
51	86.2469	50.0188
52	37.8433	50.2929
53	10.1946	48.4212
54	30.9969	46.0398
55	83.1688	46.8483

56	89.6497	50.4986
57	60.0962	50.4095
58	90.7447	46.5747
59	19.6756	43.3481
60	54.7342	45.2224
61	22.7109	43.205
62	1.97025	45.1904



In [9]:
```
# Restricted Boltzmann Machine for Time Series Analysis
#
# Igor Mol <igor.mol@makes.ai>
#
# In this implementation, a Restricted Boltzmann Machine (RBM) is employed f
# time series data generation and reconstruction. The code begins by definin
# class, TimeSeriesData, that handles the preprocessing of time series data.
# data is loaded from a CSV file and is then normalized using Min-Max scalin
# Subsequently, sequences are created from the time series data, with each s
# quence having a length of 10, suitable for training. This processed data i
# utilized for training and testing an RBM model. The RBM class, represented
# the RBM class, is initialized with the visible and hidden layer sizes, and
# includes methods for sampling hidden and visible layer states, as well as
# training the RBM using contrastive divergence.
#     The RBM model is trained on the preprocessed time series data using th
# contrastive divergence algorithm. The training involves both positive and
# gative phases, where hidden layer probabilities and states are sampled in
# positive phase and visible layer probabilities and states are sampled in t
# negative phase. The model's weights and biases are then updated based on t
# associations computed during these phases. This process is iterated for a
# cified number of epochs, refining the RBM's ability to learn and represent
# patterns in the time series data. After training, the RBM is employed to g
# rate new samples by initializing visible layer states and sampling subsequ
# hidden and visible layer states iteratively.
#     Finally, the generated samples are denormalized, transforming them bac
# the original scale. The create_table function is used to present a tabulat
# view of the generated and actual time series sequences, facilitating a qua
# tative assessment of the RBM's performance. The code concludes by executin
# main function, which orchestrates the entire process, including data prepr
# cessing, RBM training, generation of new samples, denormalization, and res
# visualization. The RBM's ability to capture temporal dependencies in time
# series data is demonstrated through its generative capabilities, providing
# valuable tool for various applications, such as synthetic data generation
# anomaly detection.


import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tabulate import tabulate

class TimeSeriesData:
    def __init__(self, file_path):
        # Initialize the class with the file path to the time series data
        self.time_series_df = pd.read_csv(file_path)
        self.sequence_length = 10
        self.scaler = MinMaxScaler()
        self.X_train = None
        self.X_test = None

    def preprocess_data(self):
        # Preprocess the time series data by converting the 'Date' column to
        # Scale the 'Value' column using Min-Max scaling
        # Convert the time series data into sequences suitable for training
        self.time_series_df['Date'] = pd.to_datetime(self.time_series_df['Da
        values = self.time_series_df['Value'].values.reshape(-1, 1)
        values_scaled = self.scaler.fit_transform(values)

        X = []
```

```
        X = []
        for i in range(len(values_scaled) - self.sequence_length):
            X.append(values_scaled[i:i + self.sequence_length].flatten())

        X = np.array(X)
        train_size = int(len(values_scaled) * 0.8)
        self.X_train, self.X_test = X[:train_size], X[train_size:]

class RBM:
    def __init__(self, visible_size, hidden_size):
        # Initialize the Restricted Boltzmann Machine (RBM) with visible and
        self.visible_size = visible_size
        self.hidden_size = hidden_size
        self.weights = np.random.randn(visible_size, hidden_size)
        self.visible_bias = np.zeros((1, visible_size))
        self.hidden_bias = np.zeros((1, hidden_size))

    def sigmoid(self, x):
        # Sigmoid activation function
        return 1 / (1 + np.exp(-x))

    def sample_hidden(self, visible_probs):
        # Sample hidden layer states based on visible layer probabilities
        hidden_probs = self.sigmoid(np.dot(visible_probs, self.weights) + se
        hidden_states = np.random.binomial(1, hidden_probs)
        return hidden_probs, hidden_states

    def sample_visible(self, hidden_probs):
        # Sample visible layer states based on hidden layer probabilities
        visible_probs = self.sigmoid(np.dot(hidden_probs, self.weights.T) +
        visible_states = np.random.binomial(1, visible_probs)
        return visible_probs, visible_states

    def train(self, data, learning_rate=0.01, epochs=50, batch_size=32):
        # Train the RBM using contrastive divergence
        num_samples = data.shape[0]

        for epoch in range(epochs):
            np.random.shuffle(data)

            for i in range(0, num_samples, batch_size):
                batch_data = data[i:i + batch_size]

                # Positive phase
                positive_hidden_probs, positive_hidden_states = self.sample_
                positive_associations = np.dot(batch_data.T, positive_hidden

                # Negative phase
                negative_visible_probs, negative_visible_states = self.sampl
                negative_hidden_probs, negative_hidden_states = self.sample_
                negative_associations = np.dot(negative_visible_states.T, ne

                # Update weights and biases
                self.weights += learning_rate * (positive_associations - neg
                self.visible_bias += learning_rate * np.mean(batch_data - ne
                self.hidden_bias += learning_rate * np.mean(positive_hidden_

    def generate_samples(self, num_samples):
        # Generate new samples from the RBM
        samples = np.random.rand(num_samples, self.visible_size)
```

```
        samples = np.random.rand(num_samples, self.visible_size)
        hidden_probs, _ = self.sample_hidden(samples)
        visible_probs, _ = self.sample_visible(hidden_probs)
        return visible_probs

def create_table(actual, predicted):
    # Create a table comparing actual and predicted values
    table_data = {'Actual': actual.flatten(), 'Predicted': predicted.flatter
    table = pd.DataFrame(table_data)
    return tabulate(table, headers='keys', tablefmt='pretty', showindex=Fals

def main():
    # Main function
    file_path = "/Users/igormol/Desktop/time_series_data.csv"  # Update with
    time_series_data = TimeSeriesData(file_path)
    time_series_data.preprocess_data()

    visible_size = time_series_data.X_train.shape[1]
    hidden_size = 50

    rbm = RBM(visible_size, hidden_size)
    rbm.train(time_series_data.X_train, epochs=50, batch_size=32)

    num_samples = time_series_data.X_test.shape[0]
    generated_samples = rbm.generate_samples(num_samples)

    generated_samples_denormalized = time_series_data.scaler.inverse_transfo
    table = create_table(time_series_data.X_test, generated_samples_denormal
    print(table)

if __name__ == "__main__":
    main()
```

```
+-----------------------+--------------------+
|        Actual         |     Predicted      |
+-----------------------+--------------------+
|  0.7904971722460135   | 83.27917314849186  |
|  0.6598653114510935   | 97.25304878229116  |
|  0.8049260492587612   | 38.718602581795025 |
|   0.968562748740495   | 12.373471959718872 |
|  0.9851986507547381   | 81.84959124297026  |
| 0.0015572327657798462 | 10.124015091682196 |
|  0.18693340210190945  | 48.12689487806005  |
|  0.9642914914648604   | 6.289509371713122  |
|  0.36876471834394303  | 21.51107867229589  |
|  0.32392408225433034  | 82.63457973388114  |
|  0.6598653114510935   |  4.15494062061679  |
|  0.8049260492587612   |  25.6633989863642  |
|   0.968562748740495   | 23.37870690231466  |
|  0.9851986507547381   | 50.95219616553286  |
| 0.0015572327657798462 | 95.20097628165057  |
|  0.18693340210190945  | 74.66660556456715  |
|  0.9642914914648604   |  85.4294891169457  |
|  0.36876471834394303  | 5.421106958873809  |
|  0.32392408225433034  | 5.880693260966947  |
|  0.5592243055017212   | 97.56946453850558  |
|  0.8049260492587612   | 2.1844220573572533 |
|   0.968562748740495   | 4.215210640617864  |
|  0.9851986507547381   | 60.14158906263921  |
| 0.0015572327657798462 | 48.69538839256778  |
```

0.18693340210190945	96.63056498857165
0.9642914914648604	91.36794385880103
0.36876471834394303	75.03546149782171
0.32392408225433034	90.39947083178674
0.5592243055017212	79.91390097719677
0.7911760619608509	13.413554881055216
0.968562748740495	33.46242236745981
0.9851986507547381	90.5669147122067
0.0015572327657798462	93.99990212612164
0.18693340210190945	64.78717071414825
0.9642914914648604	92.8008338727511
0.36876471834394303	19.169022073787065
0.32392408225433034	90.27918925479605
0.5592243055017212	51.28656065877618
0.7911760619608509	8.541347389288072
0.9044907330423345	3.3151607908265555
0.9851986507547381	50.9385041242783
0.0015572327657798462	77.44971855300368
0.18693340210190945	82.06363200174216
0.9642914914648604	7.76596329730351
0.36876471834394303	6.1100337264483775
0.32392408225433034	57.031275510675954
0.5592243055017212	26.134286276223442
0.7911760619608509	92.49282381427331
0.9044907330423345	91.79058505441036
0.62964552288955	81.13623185027413
0.0015572327657798462	3.021858269153817
0.18693340210190945	60.41345285655069
0.9642914914648604	84.6283812945742
0.36876471834394303	73.9454459428688
0.32392408225433034	87.45682830045406
0.5592243055017212	61.675725786916864
0.7911760619608509	74.34493091047209
0.9044907330423345	39.37468269679871
0.62964552288955	75.43734106710312
0.20570324453176486	72.71682210643122
0.18693340210190945	20.181836012787464
0.9642914914648604	91.26770829879905
0.36876471834394303	48.248772522946176
0.32392408225433034	11.591853768264285
0.5592243055017212	8.194665512991872
0.7911760619608509	22.928700580994853
0.9044907330423345	37.586728400155735
0.62964552288955	93.72408150499214
0.20570324453176486	99.11777572750337
0.2746799547437327	15.727982996342188
0.9642914914648604	76.30798320265964
0.36876471834394303	47.9448563638427
0.32392408225433034	95.70464404064619
0.5592243055017212	96.73918414984284
0.7911760619608509	46.911491338611725
0.9044907330423345	15.628198025062172
0.62964552288955	22.197981919483855
0.20570324453176486	83.25676658840682
0.2746799547437327	1.9355866543018971
0.22882979406431883	11.808720114743979
0.36876471834394303	21.948309594874864
0.32392408225433034	79.76095260520388
0.5592243055017212	11.961004258926152

0.7911760619608509	67.52159551713417
0.9044907330423345	47.49768358119638
0.62964552288955	95.83552297594315
0.20570324453176486	86.3285210466453
0.2746799547437327	28.60382433318455
0.22882979406431883	52.40898482823019
0.8951177076816685	9.169454068210149
0.32392408225433034	33.96516433694729
0.5592243055017212	91.68448588417117
0.7911760619608509	74.58704332997608
0.9044907330423345	91.60994071635305
0.62964552288955	30.564395625528267
0.20570324453176486	83.1953051275691
0.2746799547437327	68.61921868146894
0.22882979406431883	58.17537415428404
0.8951177076816685	9.418812805602068
0.6662781393594458	24.526621056668052
0.5592243055017212	81.29496973482807
0.7911760619608509	30.474632123449144
0.9044907330423345	88.35425790372612
0.62964552288955	61.82102858990603
0.20570324453176486	3.939416078978043
0.2746799547437327	64.1356598383237
0.22882979406431883	89.65275669639881
0.8951177076816685	27.244147213129025
0.6662781393594458	94.38543082647138
0.6814695041800238	66.2521740123401
0.7911760619608509	98.3679593761779
0.9044907330423345	5.928356285621836
0.62964552288955	24.14134608905926
0.20570324453176486	13.154370479246646
0.2746799547437327	36.16709755915701
0.22882979406431883	27.937532473583698
0.8951177076816685	69.38602050701562
0.6662781393594458	69.0726511943543
0.6814695041800238	28.510980527795006
0.8475537419017511	85.00825785036855
0.9044907330423345	89.47444115010327
0.62964552288955	87.23044628468028
0.20570324453176486	52.74694213331529
0.2746799547437327	55.33310902896337
0.22882979406431883	60.56935547762382
0.8951177076816685	85.86110066602795
0.6662781393594458	87.91329600727448
0.6814695041800238	17.600428022651005
0.8475537419017511	16.692263603821715
0.6677236188073693	90.41201649724212
0.62964552288955	94.20262380239562
0.20570324453176486	10.473105290713058
0.2746799547437327	95.1587116707903
0.22882979406431883	94.02017368779289
0.8951177076816685	83.91023505731748
0.6662781393594458	2.7075820104362323
0.6814695041800238	53.39834764935822
0.8475537419017511	7.538326603635392
0.6677236188073693	3.1347522476855256
0.497324312501742	4.377307787449108
0.20570324453176486	5.383625463220568
0.2746799547437327	3.9536997285512125

0.22882979406431883	95.75884485377753
0.8951177076816685	95.8251943040998
0.6662781393594458	86.57858486980452
0.6814695041800238	88.85987729032439
0.8475537419017511	79.48770033180568
0.6677236188073693	18.3407678355208
0.497324312501742	2.368871721142358
0.7095830762586778	16.829969537865022
0.2746799547437327	45.260637559053606
0.22882979406431883	18.0398272043558
0.8951177076816685	87.74098272442545
0.6662781393594458	4.534471520467544
0.6814695041800238	81.6563166166005
0.8475537419017511	31.038841702659294
0.6677236188073693	88.74060462970394
0.497324312501742	64.53947151786993
0.7095830762586778	56.46120982719126
0.06007433598747219	93.25029649598032
0.22882979406431883	24.01031634649891
0.8951177076816685	94.53813946244243
0.6662781393594458	17.823858212165806
0.6814695041800238	9.409699833903757
0.8475537419017511	72.2049999439253
0.6677236188073693	80.2854974565095
0.497324312501742	47.24211010891359
0.7095830762586778	74.68534239674977
0.06007433598747219	94.73079919584634
0.9099184602480401	43.01068702320875
0.8951177076816685	95.36352267526028
0.6662781393594458	90.04002789311069
0.6814695041800238	30.224431166900292
0.8475537419017511	21.843493610288196
0.6677236188073693	61.80464482727103
0.497324312501742	57.47715178235971
0.7095830762586778	53.72902602366808
0.06007433598747219	95.27470280202321
0.9099184602480401	54.54542262660178
0.19956863426895669	8.073259726386016
0.6662781393594458	15.578158213766368
0.6814695041800238	40.84446616759708
0.8475537419017511	32.279053841793484
0.6677236188073693	44.12897664107589
0.497324312501742	72.25579993669145
0.7095830762586778	10.965164199527306
0.06007433598747219	61.968650061435206
0.9099184602480401	61.26044714695871
0.19956863426895669	96.45821889266875
0.25653434037160106	8.53853529574554
0.6814695041800238	87.95465570158133
0.8475537419017511	98.38868938121138
0.6677236188073693	2.9768049717126837
0.497324312501742	19.017333180652
0.7095830762586778	64.40890388510066
0.06007433598747219	77.0644330333189
0.9099184602480401	3.582385664381384
0.19956863426895669	82.74848993948555
0.25653434037160106	61.11417233727652
0.6594584631902157	78.55913903000717
0.8475537419017511	7.213385682246578

0.6677236188073693	70.0580864055937
0.497324312501742	30.547948745006465
0.7095830762586778	91.79810866789936
0.06007433598747219	69.52049358077075
0.9099184602480401	80.73725017591593
0.19956863426895669	17.909032757268843
0.25653434037160106	19.73551116631061
0.6594584631902157	90.87501885828468
0.20316371642745362	87.76898486746806
0.6677236188073693	57.715791090552464
0.497324312501742	81.432032118268
0.7095830762586778	40.057248414113346
0.06007433598747219	66.32352167278547
0.9099184602480401	89.62542560429279
0.19956863426895669	77.46615306564443
0.25653434037160106	19.515623692472474
0.6594584631902157	83.80653006441393
0.20316371642745362	55.28542644209491
0.10907798298364516	90.94100516703766
0.497324312501742	2.644436099643882
0.7095830762586778	11.227377404381436
0.06007433598747219	31.864655157130002
0.9099184602480401	7.078293141322682
0.19956863426895669	64.21736237490366
0.25653434037160106	53.07270085695373
0.6594584631902157	78.67929277241836
0.20316371642745362	72.2314999896664
0.10907798298364516	98.69393150940893
0.6362806510722646	8.263203972513898
0.7095830762586778	99.44292354739507
0.06007433598747219	91.49826542484895
0.9099184602480401	18.50493681429586
0.19956863426895669	42.01341143283257
0.25653434037160106	90.29967355114115
0.6594584631902157	3.709306035553103
0.20316371642745362	27.60011135667619
0.10907798298364516	16.376803031524098
0.6362806510722646	37.719062024987835
0.21592590850142365	17.207091993852252
0.06007433598747219	97.65673199950544
0.9099184602480401	95.22982613679073
0.19956863426895669	45.387796231428084
0.25653434037160106	3.2163373554363957
0.6594584631902157	2.150365244063743
0.20316371642745362	23.993392558475435
0.10907798298364516	75.52017294773508
0.6362806510722646	28.721307323184714
0.21592590850142365	95.2828738020015
0.945157721942989	93.0615308627516
0.9099184602480401	72.347799744193
0.19956863426895669	96.703968793425
0.25653434037160106	25.7107392409393
0.6594584631902157	32.803679954038714
0.20316371642745362	8.697618739112116
0.10907798298364516	91.65172710753036
0.6362806510722646	48.592624055449534
0.21592590850142365	81.48816783186238
0.945157721942989	80.30850371612684
0.6607366537961368	86.37717383795157

0.19956863426895669	95.96955664256255
0.25653434037160106	80.26617256850896
0.6594584631902157	84.19211038988377
0.20316371642745362	70.51559766364156
0.10907798298364516	66.60996039890603
0.6362806510722646	12.424110374681053
0.21592590850142365	9.94593618500049
0.945157721942989	88.76194229571635
0.6607366537961368	7.949493265380911
0.15880948890829222	30.67167975562944
0.25653434037160106	14.213116714128056
0.6594584631902157	2.6944462566462577
0.20316371642745362	49.39703026629536
0.10907798298364516	95.8957449812113
0.6362806510722646	95.61256713946679
0.21592590850142365	13.49796551085388
0.945157721942989	37.941062438457166
0.6607366537961368	9.309752309998066
0.15880948890829222	1.7361143349932102
0.26490457885041674	92.2242413962192
0.6594584631902157	8.259973078932344
0.20316371642745362	94.01702210555163
0.10907798298364516	19.602047790308294
0.6362806510722646	84.03846598894665
0.21592590850142365	13.168616915409041
0.945157721942989	90.93827261477223
0.6607366537961368	10.710230657378538
0.15880948890829222	77.14163301975202
0.26490457885041674	96.63076276417341
0.26717057991638865	57.060947785861615
0.20316371642745362	93.62400706775762
0.10907798298364516	88.6126481675958
0.6362806510722646	81.4786878318243
0.21592590850142365	57.22065210009516
0.945157721942989	3.4278249015244344
0.6607366537961368	14.58555118139141
0.15880948890829222	75.59132478913686
0.26490457885041674	9.257728157394864
0.26717057991638865	70.66244944125803
0.06335828662112462	79.59402510377723
0.10907798298364516	88.06782333645651
0.6362806510722646	97.16692051724361
0.21592590850142365	87.10815568091107
0.945157721942989	10.002938674313256
0.6607366537961368	17.82891282589115
0.15880948890829222	15.669323543758889
0.26490457885041674	81.2928384771509
0.26717057991638865	6.010290417661602
0.06335828662112462	86.910658639528
0.19053416474478202	8.717979452550995
0.6362806510722646	93.66397278041633
0.21592590850142365	69.19576820331079
0.945157721942989	41.517153125449276
0.6607366537961368	61.863668505872546
0.15880948890829222	65.77711730499139
0.26490457885041674	87.76964253559956
0.26717057991638865	27.21453969752963
0.06335828662112462	43.68583918104038
0.19053416474478202	76.16601780575586

0.8587084621965111	49.96809243004279
0.21592590850142365	83.14253191089871
0.945157721942989	31.505402080582225
0.6607366537961368	5.386589019963812
0.15880948890829222	70.93230438337372
0.26490457885041674	81.00717475521974
0.26717057991638865	52.84654907504789
0.06335828662112462	14.95032814500878
0.19053416474478202	67.41760331450818
0.8587084621965111	40.40864531055447
0.6249132307609485	68.44649269124402
0.945157721942989	3.369958230791585
0.6607366537961368	84.71117865862263
0.15880948890829222	85.86887165759703
0.26490457885041674	92.41230415145581
0.26717057991638865	36.48060465895754
0.06335828662112462	88.61093558617726
0.19053416474478202	62.59697993754766
0.8587084621965111	82.98322101743248
0.6249132307609485	8.630000142710077
0.9728289883740522	22.914887465585466
0.6607366537961368	98.92105709146706
0.15880948890829222	86.2603808197425
0.26490457885041674	52.4951017283293
0.26717057991638865	42.63885962494015
0.06335828662112462	13.849457575498109
0.19053416474478202	41.91521445154792
0.8587084621965111	90.76680352979729
0.6249132307609485	5.672987456023683
0.9728289883740522	87.65094371136318
0.8506017693566247	40.50316518005544
0.15880948890829222	33.13896003986292
0.26490457885041674	91.11606072254723
0.26717057991638865	89.51449294324183
0.06335828662112462	8.232213402204634
0.19053416474478202	39.25807458649537
0.8587084621965111	7.288954174012925
0.6249132307609485	65.0081499296819
0.9728289883740522	20.960251295332128
0.8506017693566247	95.44185325529412
0.8394948740642394	6.13365678545794
0.26490457885041674	99.0730740671841
0.26717057991638865	2.665173336328759
0.06335828662112462	43.90400821806295
0.19053416474478202	96.7991821853423
0.8587084621965111	53.94590966722605
0.6249132307609485	8.357780533828752
0.9728289883740522	4.901397266170067
0.8506017693566247	73.10656859922027
0.8394948740642394	1.728558745332183
0.06358789717148432	3.9155177271800223
0.26717057991638865	98.20344455227202
0.06335828662112462	45.22742280413309
0.19053416474478202	47.03994574303331
0.8587084621965111	91.74965262318366
0.6249132307609485	68.63205784079649
0.9728289883740522	28.0612591386314
0.8506017693566247	19.59193037726484
0.8394948740642394	6.91193800037957

0.06358789717148432	63.5212079117425
0.2904688305420574	18.946900034677334
0.06335828662112462	8.985229017115595
0.19053416474478202	4.708849719535262
0.8587084621965111	31.25835225184172
0.6249132307609485	4.33954465234377
0.9728289883740522	64.96266026122076
0.8506017693566247	19.7854614494587
0.8394948740642394	79.6374210500259
0.06358789717148432	31.69638526381081
0.2904688305420574	74.10435827663062
0.8348063108093235	96.22392677352586
0.19053416474478202	3.1924722962112746
0.8587084621965111	46.3212662217851
0.6249132307609485	76.66877290066351
0.9728289883740522	2.654493331815187
0.8506017693566247	10.338182944988544
0.8394948740642394	47.400847748882384
0.06358789717148432	15.69104888676177
0.2904688305420574	89.69568527542964
0.8348063108093235	98.74330492576594
0.061970190749695044	92.09477332029086
0.8587084621965111	1.7102858198996258
0.6249132307609485	87.04085328345275
0.9728289883740522	66.4716484927086
0.8506017693566247	19.695838926911748
0.8394948740642394	15.82252003158569
0.06358789717148432	89.88544157745652
0.2904688305420574	25.877630310907687
0.8348063108093235	97.79825192537577
0.061970190749695044	49.04918517196093
0.861982880801505	48.33918222188583
0.6249132307609485	97.97366540301998
0.9728289883740522	3.3869869932677386
0.8506017693566247	87.04441587479074
0.8394948740642394	72.68143371310728
0.06358789717148432	6.489247766616362
0.2904688305420574	14.537889541223906
0.8348063108093235	35.257160643576164
0.061970190749695044	13.727631576965324
0.861982880801505	71.20345794356717
0.9626203763709649	84.39239296127585
0.9728289883740522	69.86914011803601
0.8506017693566247	85.18862194585111
0.8394948740642394	65.51092993892276
0.06358789717148432	52.09474237766704
0.2904688305420574	5.073328477500836
0.8348063108093235	33.6187740783853
0.061970190749695044	36.09511786797954
0.861982880801505	70.49971812744276
0.9626203763709649	95.33625122128903
0.3605406724877948	89.22265539712964
0.8506017693566247	80.41980834075761
0.8394948740642394	97.68036009027297
0.06358789717148432	19.41166094468634
0.2904688305420574	25.30158503214152
0.8348063108093235	87.80848901162524
0.061970190749695044	64.89796113415548
0.861982880801505	7.144891286115118

0.9626203763709649	93.70084369841537
0.3605406724877948	37.494360628715675
0.431397602293037	78.67525317448477
0.8394948740642394	4.906062849384632
0.06358789717148432	12.08454991233298
0.2904688305420574	83.28730949845416
0.8348063108093235	58.80890552520927
0.061970190749695044	17.827130208546485
0.861982880801505	85.70504074074586
0.9626203763709649	37.75953719748297
0.3605406724877948	94.45148714630643
0.431397602293037	31.500872866789436
0.8881046729508704	9.409500503265864
0.06358789717148432	97.39610140769625
0.2904688305420574	14.681662648659525
0.8348063108093235	62.619761863509744
0.061970190749695044	80.54834975786589
0.861982880801505	25.65500702412808
0.9626203763709649	36.558353613391965
0.3605406724877948	42.99967300240872
0.431397602293037	4.688475334078323
0.8881046729508704	79.9020297960093
0.8613821978061533	42.92680133809352
0.2904688305420574	98.28806233693489
0.8348063108093235	42.06632803935837
0.061970190749695044	90.75517338732492
0.861982880801505	41.093445021696034
0.9626203763709649	28.049981690508947
0.3605406724877948	27.048997313313002
0.431397602293037	94.0965194229653
0.8881046729508704	22.08953475679484
0.8613821978061533	24.893740638171163
0.0	46.36721123737864
0.8348063108093235	22.236204605674914
0.061970190749695044	8.308549112807189
0.861982880801505	91.20224802505113
0.9626203763709649	51.87719423961
0.3605406724877948	91.65095988330873
0.431397602293037	44.670036788042566
0.8881046729508704	69.91991880065778
0.8613821978061533	87.77236770514916
0.0	26.15267762895508
0.5565536599068617	88.69617570897954
0.061970190749695044	12.324857125144307
0.861982880801505	2.559723738987048
0.9626203763709649	36.976310814684524
0.3605406724877948	79.78852929283023
0.431397602293037	46.10098553586644
0.8881046729508704	13.182154696923677
0.8613821978061533	14.1048869627678
0.0	42.400190503137075
0.5565536599068617	73.6752596003291
0.15858527314156567	88.11966334526568
0.861982880801505	11.953514572456426
0.9626203763709649	2.206061312879562
0.3605406724877948	54.973394914787
0.431397602293037	96.55564813655161
0.8881046729508704	93.31006850726352
0.8613821978061533	52.41734373646695

0.0	8.102897891208201
0.5565536599068617	91.82362389274533
0.15858527314156567	1.5834945579994468
0.6387033965750694	29.82714152644709
0.9626203763709649	11.355424780184896
0.3605406724877948	96.15348397552216
0.431397602293037	5.483380095145241
0.8881046729508704	88.72063942798918
0.8613821978061533	74.19856156901893
0.0	78.16723374566746
0.5565536599068617	5.581095156131009
0.15858527314156567	98.54417750294313
0.6387033965750694	51.095121723196385
0.659688544301449	4.625601454868243
0.3605406724877948	11.376428102941265
0.431397602293037	3.5320029649650735
0.8881046729508704	7.33784835670122
0.8613821978061533	92.55822293397888
0.0	87.15339166485451
0.5565536599068617	74.19128794001163
0.15858527314156567	24.3618101666428
0.6387033965750694	93.47088915916504
0.659688544301449	88.65601406391019
0.5596997632997004	6.039863257488422
0.431397602293037	10.456787680922782
0.8881046729508704	2.992100260188335
0.8613821978061533	91.92636967417704
0.0	87.64645665870249
0.5565536599068617	87.01891025109589
0.15858527314156567	79.94591482284575
0.6387033965750694	78.96274948185157
0.659688544301449	8.919444514966816
0.5596997632997004	53.85970203520639
0.8624590835032977	94.08518856136729
0.8881046729508704	28.694323289713235
0.8613821978061533	3.11802342179346
0.0	17.532714677100216
0.5565536599068617	90.71514579715813
0.15858527314156567	97.70374958129966
0.6387033965750694	24.102720157588838
0.659688544301449	82.53852581115031
0.5596997632997004	12.70650577974657
0.8624590835032977	67.14444604999647
0.3771564036246608	10.550644213693085
0.8613821978061533	81.18964461559467
0.0	24.052862020248877
0.5565536599068617	86.47456510783405
0.15858527314156567	85.50233787816143
0.6387033965750694	9.590222202199662
0.659688544301449	70.41913187366669
0.5596997632997004	16.420568875741168
0.8624590835032977	87.74205893770944
0.3771564036246608	84.22978126155287
0.09994518027903934	16.79074272533231
0.0	18.413135658554527
0.5565536599068617	38.081614104218815
0.15858527314156567	94.59919909868566
0.6387033965750694	9.106091448225808
0.659688544301449	25.293671794258326

0.5596997632997004	17.43917006721535
0.8624590835032977	93.73424949360366
0.3771564036246608	69.42497831444464
0.09994518027903934	24.33067818257623
0.3085129734030654	21.016700754829486
0.5565536599068617	2.167634206392044
0.1585827314156567	9.448667011861305
0.6387033965750694	5.474866170321662
0.659688544301449	13.713794452796716
0.5596997632997004	93.12957829941998
0.8624590835032977	50.42157416865137
0.3771564036246608	64.4778566261867
0.09994518027903934	93.0979050478007
0.3085129734030654	12.32663600873598
0.8315980540556376	97.13135592060767
0.1585827314156567	23.510476134411547
0.6387033965750694	97.58188558119237
0.659688544301449	24.10734007300354
0.5596997632997004	14.436796107401776
0.8624590835032977	59.77928110296398
0.3771564036246608	87.52613389099484
0.09994518027903934	21.048401553914374
0.3085129734030654	30.612843350299443
0.8315980540556376	91.82993095858482
0.8965763850206687	93.46257797674146
0.6387033965750694	94.92461927357792
0.659688544301449	22.330129216545306
0.5596997632997004	38.756549447577946
0.8624590835032977	16.408775590874818
0.3771564036246608	23.488737509425107
0.09994518027903934	28.16272001479939
0.3085129734030654	88.16402119098892
0.8315980540556376	49.74760574713882
0.8965763850206687	90.51268837358268
0.6002680575694376	66.00877161836527
0.659688544301449	98.72177576245286
0.5596997632997004	3.4802921110361074
0.8624590835032977	71.69245166245437
0.3771564036246608	11.585003299411538
0.09994518027903934	26.330621540824673
0.3085129734030654	13.83886483615199
0.8315980540556376	45.62543347873813
0.8965763850206687	55.49162062438589
0.6002680575694376	52.430342231159436
0.9075551685431381	85.8004298949952
0.5596997632997004	99.68810047574323
0.8624590835032977	2.225589535398182
0.3771564036246608	18.510182490377744
0.09994518027903934	3.9057616854415693
0.3085129734030654	95.86619576292058
0.8315980540556376	1.631890092320884
0.8965763850206687	24.58110765479365
0.6002680575694376	6.751115258357882
0.9075551685431381	3.0609888773249785
0.1950037835883633	84.12989487467257
0.8624590835032977	91.54683628433267
0.3771564036246608	1.6338455306073363
0.09994518027903934	45.45106193523779
0.3085129734030654	74.51989653898869

```
|  0.8315980540556376   | 94.97253659147822  |
|  0.8965763850206687   | 24.732593969802114 |
|  0.6002680575694376   |  32.9551902292043  |
|  0.9075551685431381   | 15.409450131924537 |
|  0.1950037835883633   | 20.508998709113353 |
|  0.5465073870614098   | 92.65674948500815  |
|  0.3771564036246608   | 12.467924790155239 |
|  0.09994518027903934  | 31.684072709023962 |
|  0.3085129734030654   | 72.92255552665357  |
|  0.8315980540556376   | 65.63005968189448  |
|  0.8965763850206687   | 62.46655506617356  |
|  0.6002680575694376   | 24.59018207749384  |
|  0.9075551685431381   | 95.29971649085316  |
|  0.1950037835883633   | 9.507400903088707  |
|  0.5465073870614098   | 83.94958774609162  |
|   0.225435937328899   |  3.37380565944312  |
+-----------------------+--------------------+
```

In [ ]: