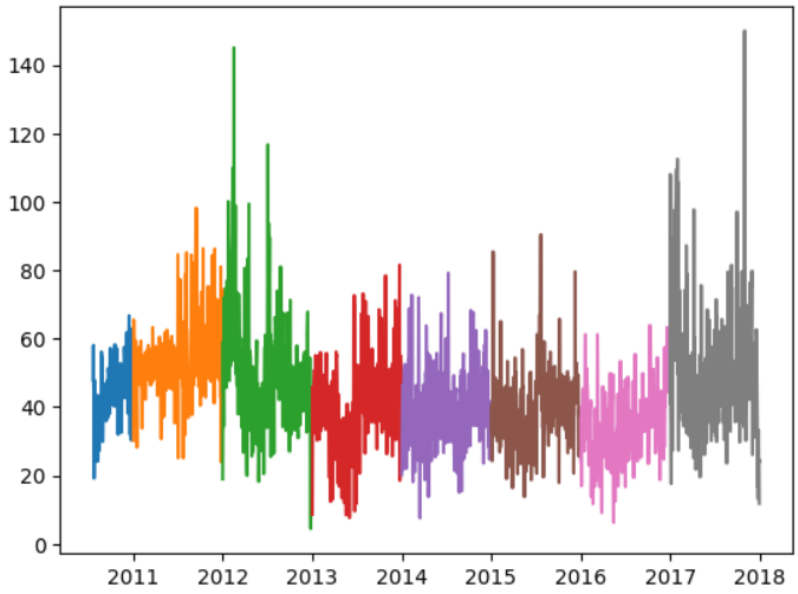
SustAInify

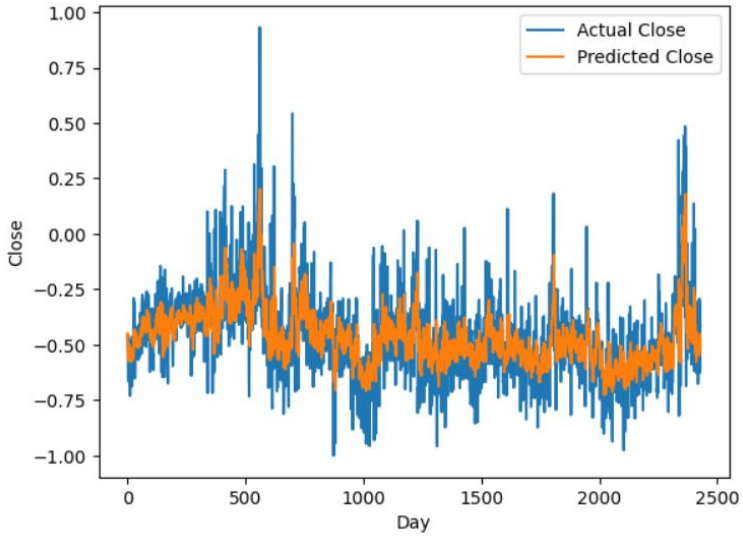
- Initially the data had to be done a lot of the preprocessing as it had a lot of the outliers and the uneven values. For the pre processing first we replaced the outliers with the average of its neighbours. And then grouped the data of the each of the year according to the mean and indexes.



- Then we created a deep copy of the DataFrame, sets the "Date" column as the index, and then adds new columns representing the price at different time lags (from 1 to n). The original DataFrame is not modified, and rows with missing values resulting from the shift operation are removed. The function returns the modified DataFrame. The code then applies this function to a DataFrame named df with n set to 7, resulting in a new DataFrame called shifted_df.
- Using the minmaxscaler the data was then fit and transformed.
- After splitting the dataset into train and the test set, they were reshaped and the torch.float() was applied.
- Then we defined a PyTorch dataset class, TimeSeriesDataset, tailored for time series data. It takes input sequences X and y during initialization and implements required methods (__len__ and __getitem__). Instances of this class, train_dataset and test_dataset, are created using training and testing data. These datasets can be employed with PyTorch data loaders for efficient handling of time series data during model training and evaluation.
- Finally we defined a Long Short-Term Memory (LSTM) neural network model using PyTorch. The LSTM class is a subclass of nn.Module and takes parameters such as input_size, hidden_size, num_stacked_layers. The model architecture includes an LSTM layer with specified parameters, followed by a dropout layer and a fully connected (linear) layer. The forward method implements the forward pass of the model. It initializes the hidden and cell states, passes the input through the LSTM layer, applies dropout, and then passes the final sequence element through the linear layer. An instance of this model is created (model = LSTM(1, 4, 1)) and is moved to the CPU (model.to('cpu')). The resulting model can be used for tasks such as time series prediction or sequence modeling.

```
class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_stacked_layers):
        super().__init__()
        self.hidden_size = hidden_size
        self.num_stacked_layers = num_stacked_layers

        self.lstm = nn.LSTM(input_size, hidden_size, num_stacked_layers,
                            batch_first=True)

        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        batch_size = x.size(0)
        h0 = torch.zeros(self.num_stacked_layers, batch_size, self.hidden_size).to(device)
        c0 = torch.zeros(self.num_stacked_layers, batch_size, self.hidden_size).to(device)

        out, _ = self.lstm(x, (h0, c0))
        out = self.fc(out[:, -1, :])
        return out

model = LSTM(1, 4, 1)
model.to('cpu')
model
```

- Then we trained each of the epoch and validated them hands on.

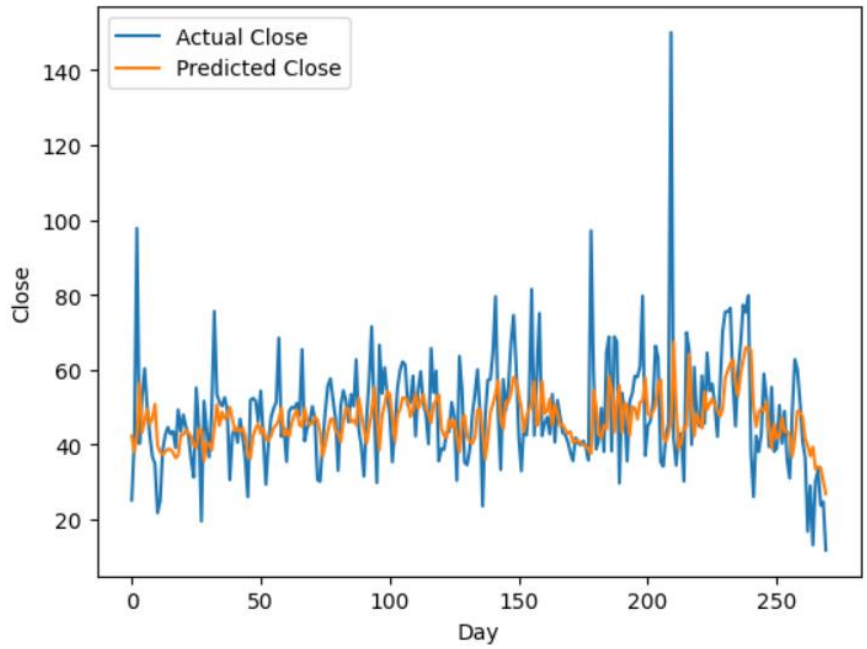



```
z=new_y_train-train_predictions
z=np.square(z)
z=np.mean(z)
```

```
z=np.sqrt(z)
print('Root Mean Square Error is ', z)
```

```
Root Mean Square Error is  10.20529868411758
```

```
test_predictions = model(X_test.to(device)).detach().cpu().numpy().flatten()

dummies = np.zeros((X_test.shape[0], lookback+1))
dummies[:, 0] = test_predictions
dummies = scaler.inverse_transform(dummies)

test_predictions = dc(dummies[:, 0])
test_predictions
```



- Optimization condition for the renewable energy

- X = Q_grid
- Thus (X*0.15) + [(1050-X)*0.05] + 150*1 >= 0.2*1200
- Therefore, Q_grid = 375MWH and Q_exchange = 675MWH, Q_solar = 150MWH on Jan 1st 2018
- This optimizes the both of the price as well as renewable enrgy.

- This ensures that the renewable energy percentage is always greater than 20 per cent for any given day, while also maintaining the optimum price paid
- And the total price of the energy is 38564.3369294945 in Euros.
- MSE = 10.1
- R^2 = 0.76
- The outputs of the prediction of the Jan 2018 are as :

```
price_1_jan=df_saved['Predicted Price(EUR/MWh)'][0]
price_1_jan
```

```
25.12123989554743
```

Total Renewable electricity is 20%.

```
Total_price= Q_Solar * 0 + Q_Grid*57.62 + Q_Exchange*price_1_jan
```

```
print('Total Price of Energy in EUR' , Total_price)
```

```
Total Price of Energy in EUR 38564.336929494515
```