

Generative AI Appendix

Had to ask how MinMaxScaler was called. I was just using transform and was getting errors. This was the response I used.

You can use `sklearn`'s `MinMaxScaler` to achieve this easily.

Code Example:

```
python Copy code  
  
from sklearn.preprocessing import MinMaxScaler  
  
# Initialize the MinMaxScaler  
scaler = MinMaxScaler(feature_range=(0, 1))  
  
# TODO: Create a normalized version of the 'cycle' column (e.g., 'cycle_norm') using the o  
train_df['cycle_norm'] = scaler.fit_transform(train_df[['time_in_cycles']])  
  
# Check the normalized column  
print(train_df[['time_in_cycles', 'cycle_norm']].head())
```

Asked for an example of what generated sequences for an LSTM would look like. This was the response I used for syntax reference.

```
def generate_sequences(data, sequence_length, step=1):  
    """  
    Generate sequences from data for training an LSTM model.  
  
    Parameters:  
    - data: A numpy array or list containing the input data.  
    - sequence_length: The length of each sequence to be generated.  
    - step: The step size to move forward to generate the next sequence.  
  
    Returns:  
    - sequences: A numpy array of sequences where each sequence is of length `sequence_length`  
    """  
    sequences = []  
    for i in range(0, len(data) - sequence_length, step):  
        sequences.append(data[i:i + sequence_length])  
  
    return np.array(sequences)
```

Asked for what are good general starting parameters for LSTM, this is what I used and adjusted.

Example LSTM Model Setup in Keras:

```
python Copy code  
  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import LSTM, Dense, Dropout  
  
model = Sequential()  
model.add(LSTM(100, input_shape=(sequence_length, num_features), return_sequences=True))  
model.add(Dropout(0.2))  
model.add(LSTM(50, return_sequences=False))  
model.add(Dropout(0.2))  
model.add(Dense(1)) # For regression; adjust for classification  
model.compile(optimizer='adam', loss='mean_squared_error') # Adjust loss for classification
```

Asked for it to give me a general plot layout for training and validation plot

```
# Plot training & validation loss values  
plt.figure(figsize=(10, 6))  
plt.plot(history.history['loss'], label='Training Loss')  
plt.plot(history.history['val_loss'], label='Validation Loss')  
plt.title('Model Loss')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.legend()  
plt.grid(True)  
plt.show()
```