**Understanding Time-Series Functioning**

Understanding the working of Lag Features which are being added to get the forecast:

**Purpose of Shifting the Data**

**Creating Lag Features:**

* Shifting the data by i time steps create lag features. Lag features are previous observations of the time series that can be used as input features for predicting future values.
* For example, if you want to predict the pollution level at time t, you might use the pollution levels at times t-1, t-2, etc., as input features.

**Transforming Time Series Data into Supervised Learning Format:**

* Time series data is inherently sequential and needs to be transformed into a format suitable for supervised learning.
* By shifting the data, we create input-output pairs where the inputs are past observations (lags) and the outputs are future values.
* This transformation enables the use of machine learning models to learn the temporal dependencies in the data.

**Overall Purpose of the Process:**

The overall process of shifting the data and preparing it for an LSTM model involves several steps:

1) Load and Preprocess the Dataset:

2) Load the time series data from a CSV file.

3) Encode any categorical features and ensure all data is in a numeric format.

4) Normalize the features to ensure they are on a similar scale.

5) Frame as Supervised Learning Problem

6) Use the series\_to\_supervised function to transform the time series data into a supervised learning format by creating lag features and future value columns.

7) This involves shifting the data to create lagged versions of the features and corresponding output columns.

8) Drop Unnecessary Columns:

9) After creating the lagged features and future values, drop any columns that are not needed for the prediction.

10) Prepare Data for LSTM Model:

The final DataFrame now contains rows where each row has past observations (inputs) and the target value (output) for training the LSTM model.

Understanding the **series\_to\_supervised** function in detail:

1. *if type(data) is list:*

*n\_vars = 1*

*else:*

*n\_vars = data.shape[1]*

It determines the number of variables (or features) in the dataset. If the input data is a list, it assumes there is only one variable. If data is a DataFrame or a NumPy array, it uses the number of columns in data to determine the number of variables.

:::::::::::::::::::::::::::::::::::::::::::::::::::::::::

2) *for i in range(n\_in, 0, -1):*

*# Shift the DataFrame by i time steps*

*shifted\_df = df.shift(i)*

*# Append the shifted DataFrame to the list of columns*

*cols.append(shifted\_df)*

*# Create column names for the shifted DataFrame*

*for j in range(n\_vars):*

*col\_name = 'var%d(t-%d)' % (j+1, i)*

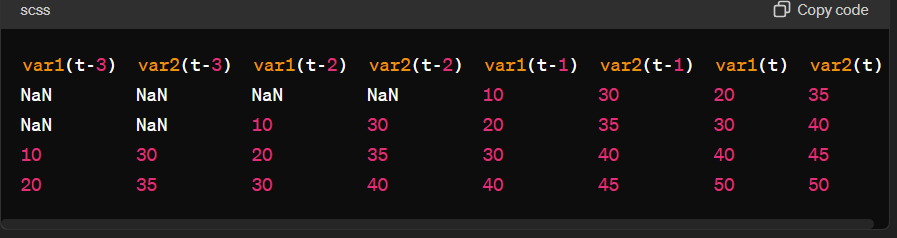
*names.append(col\_name)*

:::::::::::::::::::::::::::::::::::::::::::::::::::::::::

**range(n\_in, 0, -1)**: This creates a sequence from n\_in down to 1. For instance, if n\_in is 3, the sequence will be [3, 2, 1].

**df.shift(i)**: This shifts the DataFrame by i time steps, creating lagged versions of the data.

**Column Naming**: The inner list comprehension creates names for each shifted column, where j iterates over the number of variables, and i indicates the lag.

Ref: 

Keeping the above for visualization reference.

*for i in range(0, n\_out):*

*cols.append(df.shift(-i))*

*if i == 0:*

*names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]*

*else:*

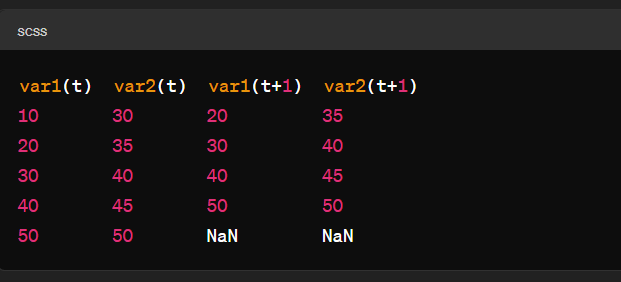
*names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]*

**Shifting DataFrame for Future Values:**

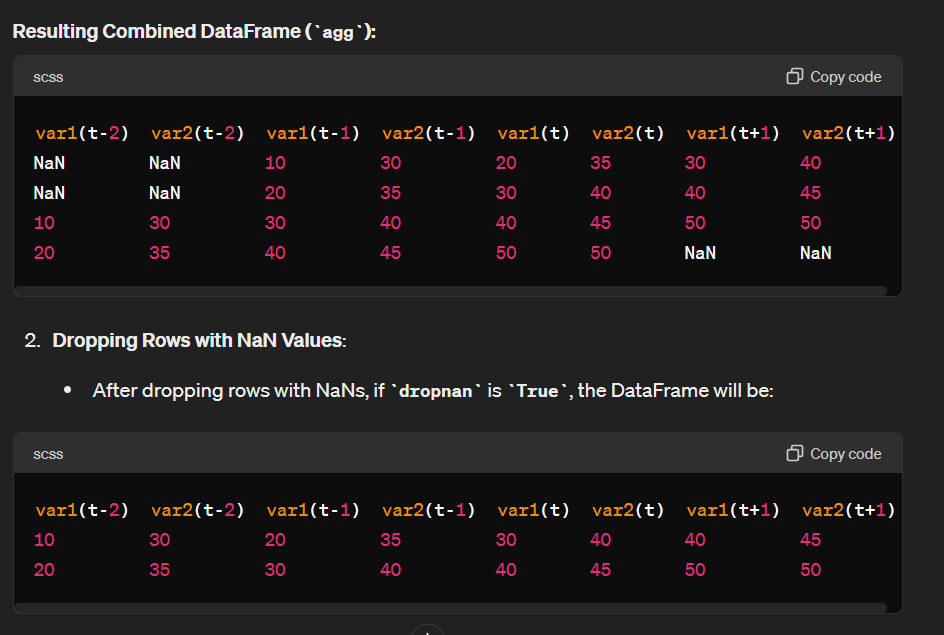
df.shift(-i): This shifts the DataFrame by -i time steps to the left, effectively aligning future values with the current time step.

When i = 0, df.shift(0) doesn't shift the data.

When i = 1, df.shift(-1) shifts the data by one step to the left, so the value at t+1 becomes the value at t, and so on.



*Returning the agg and removing null values:*



*encoder = LabelEncoder()*

*values[:,4] = encoder.fit\_transform(values[:,4])*

* The column wnd\_dir contains categorical values therefore:
  + LabelEncoder transforms these categorical values into numerical labels.
* The fit\_transform method is used to both fit the encoder to the data and transform the data in a single step.
* During the fit step, the encoder learns the mapping between categories and numerical labels.
* During the transform step, the encoder applies this mapping to transform the categorical values into numerical labels.

*scaler = MinMaxScaler(feature\_range=(0, 1))*

*scaled = scaler.fit\_transform(values)*

* This initializes a MinMaxScaler object with the specified feature range. Here, the feature\_range is set to (0, 1), which means the data will be scaled to a range between 0 and 1.

*reframed.drop(reframed.columns[[9,10,11,12,13,14,15]], axis=1, inplace=True)*



**FAQ**

**Q1) Why the use of i for shifting data?**

**Ans)** Using a variable like i in a loop rather than a concrete number allows for flexibility and scalability when creating lag features. Here’s a breakdown:

Flexibility: The use of i allows the function to handle any number of past time steps (n\_in) dynamically. Instead of writing separate code for each possible lag, a loop with i handles any specified number of lags.

Scalability: If you need to change the number of past observations to consider (e.g., from the last 3 days to the last 5 days), you only need to change the value of n\_in rather than rewriting the code.

**Q2) More reference about creating a lag feature via pandas?**

**Ans)** <https://www.statology.org/pandas-lag/>