**Handling sequences with PyTorch**

We've learned to handle tabular and image data. Let's now discuss sequential data.

**Sequential data**

Sequential data is ordered in time or space, where the order of the data points is important and can contain temporal or spatial dependencies between them. Time series, data recorded over time like stock prices, weather, or daily sales is sequential. So is text, in which the order of words in a sentence determines its meaning. Another example is audio waves, where the order of data points is crucial to the sound reproduced when the audio file is played.

**Electricity consumption prediction**

In this chapter, we will tackle the problem of predicting electricity consumption based on past patterns. We will use a subset of the electricity consumption dataset from the UC Irvine Machine Learning Repository. It contains electricity consumption in kilowatts, or kW, for a certain user recorded every 15 minutes for four years.

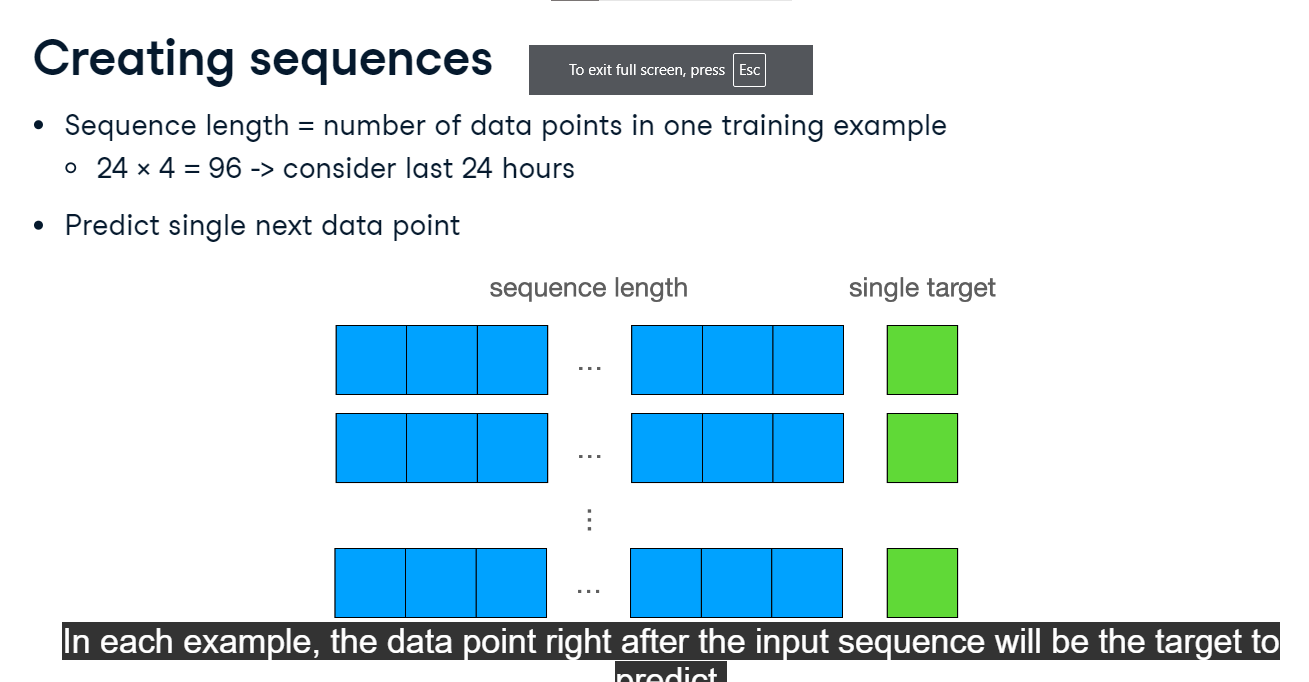
1. 1 Trindade,Artur. (2015). ElectricityLoadDiagrams20112014. UCI Machine Learning Repository. https://doi.org/10.24432/C58C86.

**Train-test split**

In many machine learning applications, one randomly splits the data into training and testing sets. However, with sequential data, there are better approaches. If we split the data randomly, we risk creating a look-ahead bias, where the model has information about the future when making forecasts. In practice, we won't have information about the future when making predictions, so our test set should reflect this reality. To avoid the look-ahead bias, we should split the data by time. We will train on the first three years of data, and test on the fourth year.

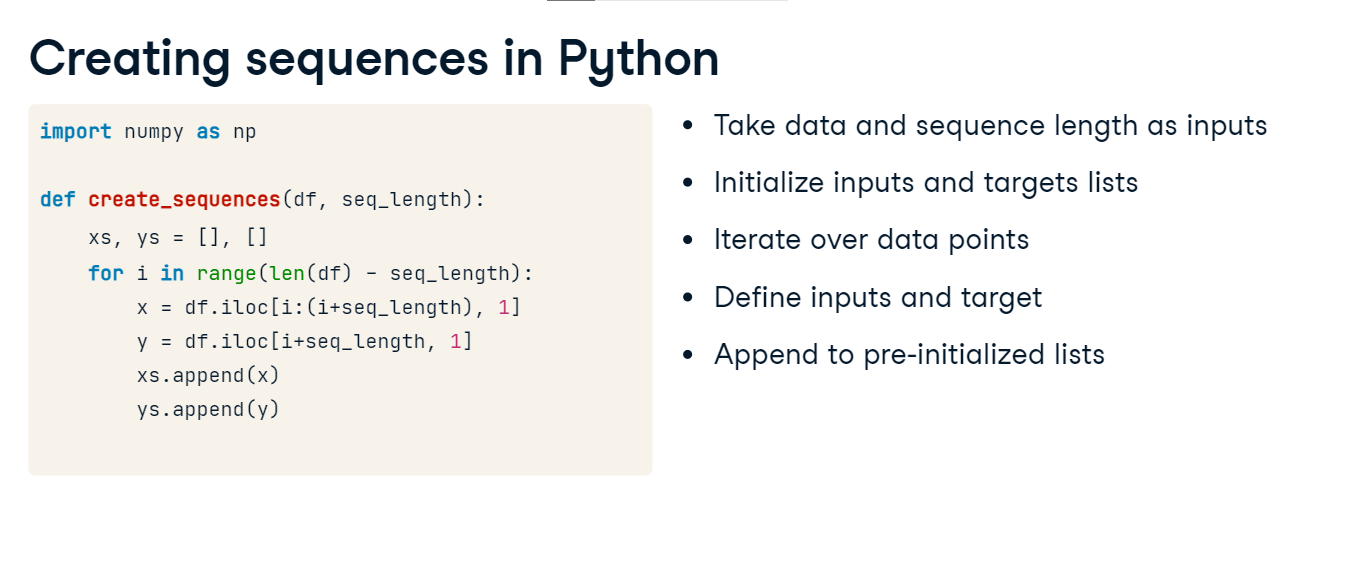
**Creating sequences**

To feed the training data to the model, we need to chunk it first to create sequences that the model can use as training examples. First, we need to select the sequence length, which is the number of data points in one training example. Let's make each forecast based on the previous 24 hours. Because data is at 15 minute intervals, we need to use 24 times 4 which is 96 data points. In each example, the data point right after the input sequence will be the target to predict.



**Creating sequences in Python**

Let's implement a Python function to create sequences. It takes the DataFrame and the sequence length as inputs. We start with initializing two empty lists, xs for inputs and ys for targets. Next, we iterate over the DataFrame. The loop only goes up to "len(df) - seq\_length", ensuring that for every iteration, there are always seq\_length data points available in the DataFrame for creating the sequence and a subsequent data point to serve as the target. For each considered data point, we define inputs x as the electricity consumption values starting from this point plus the next sequence length points, and the target y as the subsequent electricity consumption value. The 1 passed to the iloc method stands for the second DataFrame column, which stores the electricity consumption data. Finally, we append the inputs and the target to pre-initialized lists, and after the loop, return them as NumPy arrays.



**TensorDataset**

Let's use our function to create sequences from the training data. This gives us almost 35 thousand training examples. To convert them to a torch Dataset, we will use the TensorDataset function. We pass it two arguments, the inputs and the targets. Each argument is the NumPy array converted to a tensor with torch.from\_numpy and parsed to float. The TensorDataset behaves just like all other torch Datasets and it can be passed to a DataLoader in the same way.



**Applicability to other sequential data**

Everything we have learned here can also be applied to other sequential data. For example, Large Language Models are trained to predict the next word in a sentence, a problem similar to predicting the next amount of electricity used. For speech recognition, which means transcribing an audio recording of someone speaking to text, one would typically use the same sequence-processing model architectures we will learn about soon.