Remaining Useful Life

Remaining Useful Life

The Remaining Useful Life is a key concept in predictive maintenance

The RUL refers to the time until a component becomes unusable

- If we can estimate the RUL of a component
- ...We can schedule maintenance operations only when they are needed

Current best practices are based on preventive maintenance

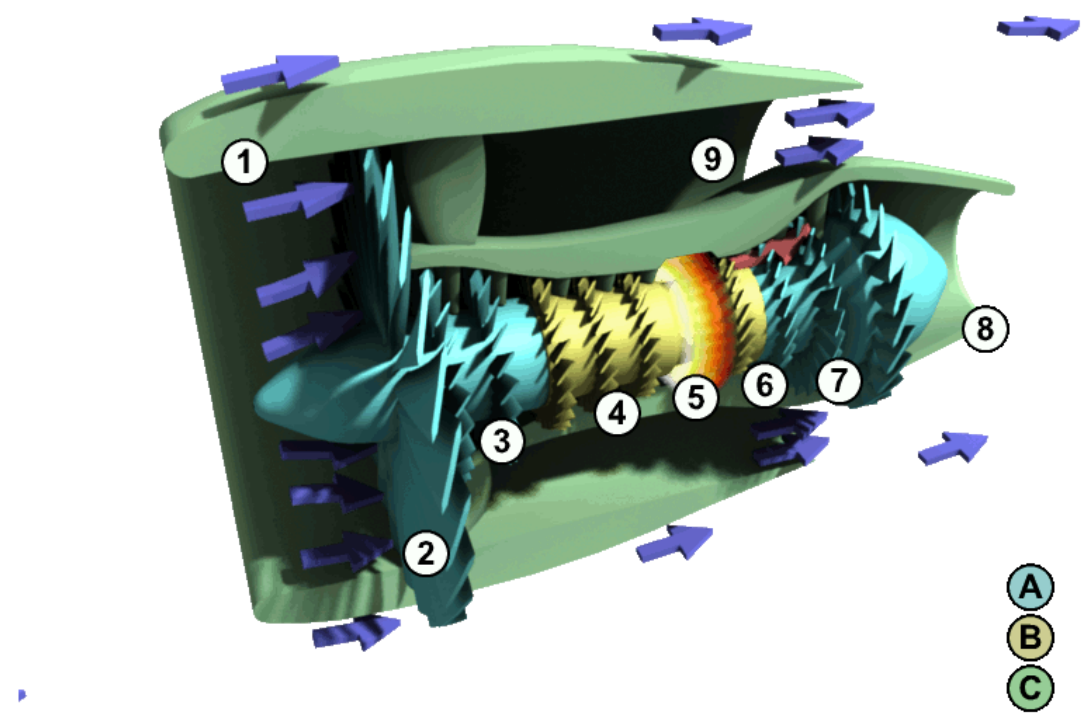
I.e. on having a fixed maintenance schedule for each component family

- RUL prediction can lead to significant savings
- ...By delaying maintenance operations w.r.t. the schedule
- ...But only as long as we are still able to prevent critical failures

The Dataset

We will consider the NASA C-MAPSS dataset

- The Modular Aero-Propulsion System Simulation (MAPSS)
- ...Is a NASA-developed simulator for turbofan engines



- It comes with both a Military (MAPSS) and commercial versionn (C-MAPSS)
- They different in the attributes of the considered engines

The Dataset

The C-MAPSS system can simulate a number of faults and defects

...And it was used to build a high-quality dataset for the PHM08 conference

- The dataset consists of 4 "training set" files and 4 "test set" files
- The training set files contain multiple run-to-failure experiments
- The test set files contain truncated experiments

PHM-08 hosted a competition based on this dataset

The goal was to predict the RUL at the end of each truncated experiment

- This is fine as long as the focus is on pure prediction
- ...But we want to tackle the whole predictive maintenance problem

As a consequence, we will focus only on the "training" data

The Dataset

Each training file refes to different faults and operating conditions

Dataset	Operating conditions	Fault modes
FD001	1 (sea level)	HPC
FD002	6	HPC
FD003	1 (sea level)	HPC, fan
FD004	6	HPC, fan

Fault modes refer to degration of either:

- The High Pressure Compressor
- The fan at the "mouth" of the engine

Inspecting the Data

Let's have a look at the row data

In [2]:
```
data_folder = os.path.join('..', 'data')
data = util.load_data(data_folder)
data.head()
```

Out[2]:

	src	machine	cycle	p1	p2	p3	s1	s2	s3	s4	...	s13	s14	s15	s16	s1
0	train_FD001	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	...	2388.02	8138.62	8.4195	0.03	39.
1	train_FD001	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	...	2388.07	8131.49	8.4318	0.03	39.
2	train_FD001	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	...	2388.03	8133.23	8.4178	0.03	39
3	train_FD001	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	...	2388.08	8133.83	8.3682	0.03	39.
4	train_FD001	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	...	2388.04	8133.80	8.4294	0.03	39.

5 rows × 28 columns

- Columns "p1, p2, p3" refer to controlled parameters
- Columns "s1" to "s21" refer to sensor reading
- Binning has already been applied in the original dataset

Statistics

Let's check some statistics

In [3]:
```
dt_in = list(data.columns[3:-1]) # Exclude metadata
data[dt_in].describe()
```

Out[3]:

	p1	p2	p3	s1	s2	s3	s4	
count	160359.000000	160359.000000	160359.000000	160359.000000	160359.000000	160359.000000	160359.000000	160359.00
mean	17.211973	0.410004	95.724344	485.840890	597.361022	1467.035653	1260.956434	9.894999
std	16.527988	0.367938	12.359044	30.420388	42.478516	118.175261	136.300073	4.265554
min	-0.008700	-0.000600	60.000000	445.000000	535.480000	1242.670000	1023.770000	3.910000
25%	0.001300	0.000200	100.000000	449.440000	549.960000	1357.360000	1126.830000	5.480000
50%	19.998100	0.620000	100.000000	489.050000	605.930000	1492.810000	1271.740000	9.350000
75%	35.001500	0.840000	100.000000	518.670000	642.340000	1586.590000	1402.200000	14.62000
max	42.008000	0.842000	100.000000	518.670000	645.110000	1616.910000	1441.490000	14.62000

8 rows × 24 columns

There are no missing values:

In [4]:
```
data[dt_in].isnull().any().any()
```

Out[4]: `False`

Heatmaps

Let's prepare for displaying all time series

First, we standardize each column:

In [5]:
```
data_sv = data.copy()
data_sv[dt_in] = (data_sv[dt_in] - data_sv[dt_in].mean()) / data_sv[dt_in].std()
```

Then, we split our data based on the source file:

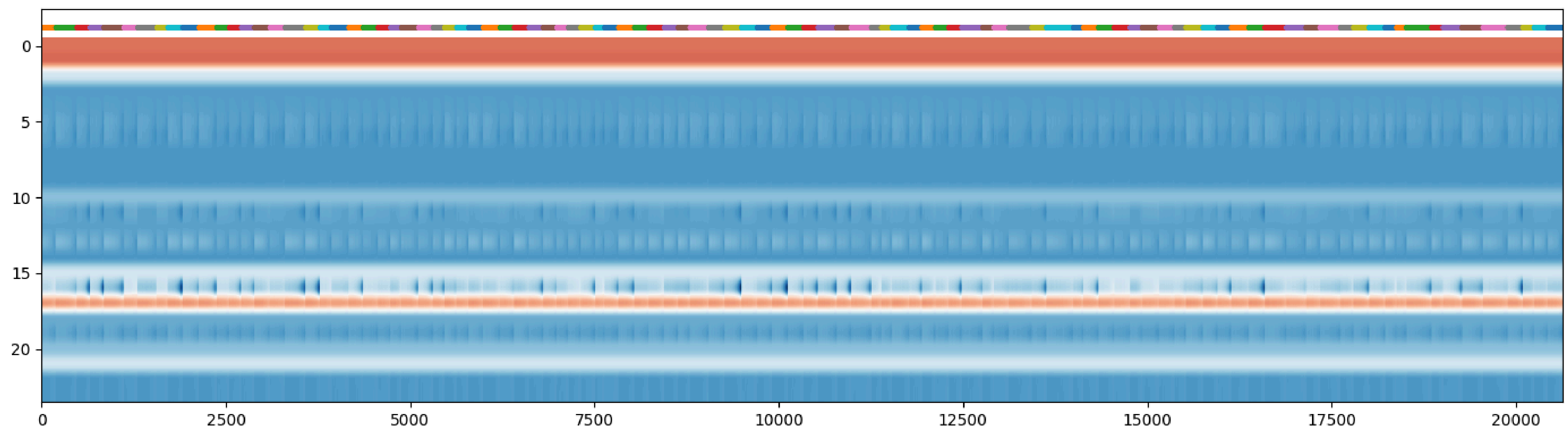
In [6]:
```
data_sv_dict = util.split_by_field(data_sv, field='src')
print('{{{}}}'.format(', '.join(f'{k}: ...' for k in data_sv_dict.keys())))
```

```
{train_FD001: ..., train_FD002: ..., train_FD003: ..., train_FD004: ...}
```

Heatmaps

Now, let's plot all parameters and sensors for FD001

In [7]:
```
tmp = data_sv_dict['train_FD001']
util.plot_dataframe(tmp[dt_in], labels=tmp['machine'], figsize=figsize)
```

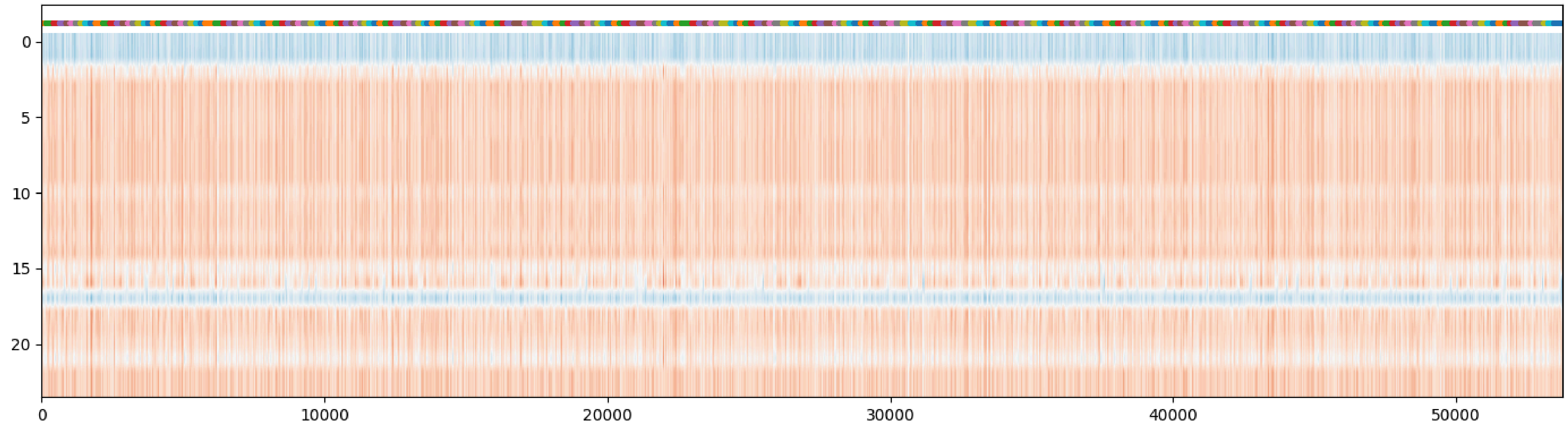


- The data contains series for multiple machines
- These are highlighted at the top with different colors

Heatmaps

Now, let's plot all parameters and sensors for FD002

In [8]:
```
tmp = data_sv_dict['train_FD002']
util.plot_dataframe(tmp[dt_in], labels=tmp['machine'], figsize=figsize)
```

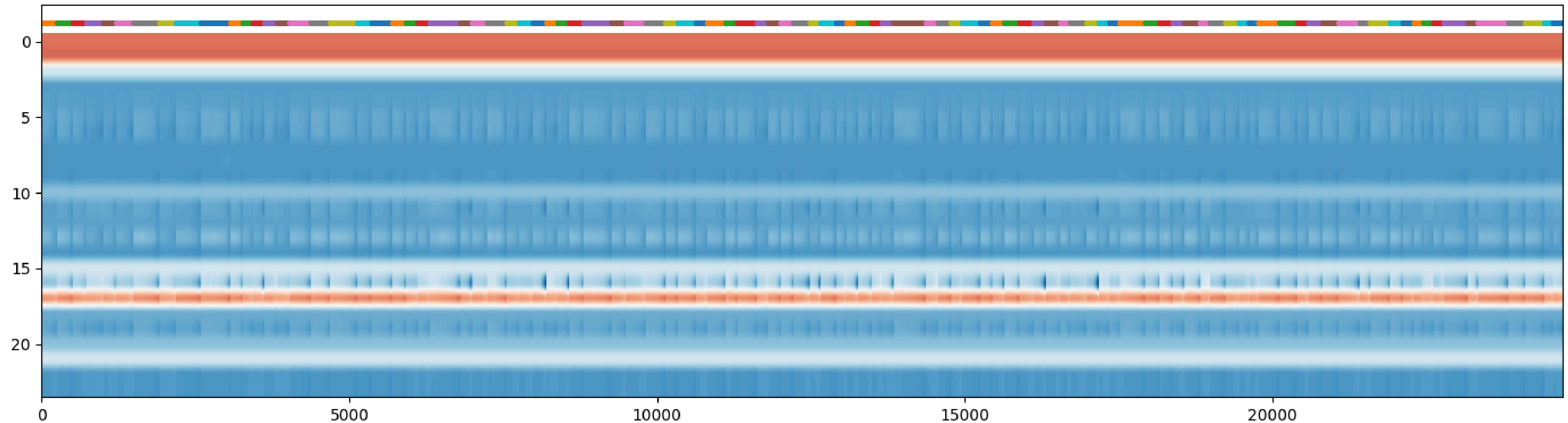


- The series is much more variable in this case
- This is due to the multiple operating conditions

Heatmaps

Now, let's plot all parameters and sensors for FD003

In [9]:
```
tmp = data_sv_dict['train_FD003']
util.plot_dataframe(tmp[dt_in], labels=tmp['machine'], figsize=figsize)
```

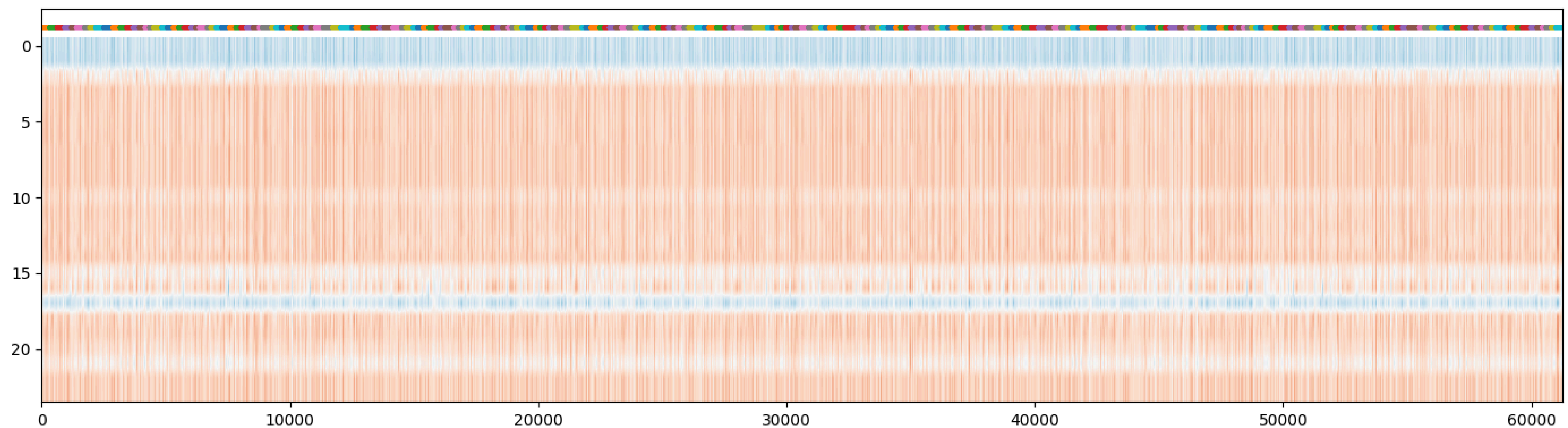


- Only one operating condition in this case (but two fault modes)
- The series is similar to FD001

Heatmaps

Finally, let's plot all parameters and sensors for FD004

In [10]:
```
tmp = data_sv_dict['train_FD004']
util.plot_dataframe(tmp[dt_in], labels=tmp['machine'], figsize=figsize)
```

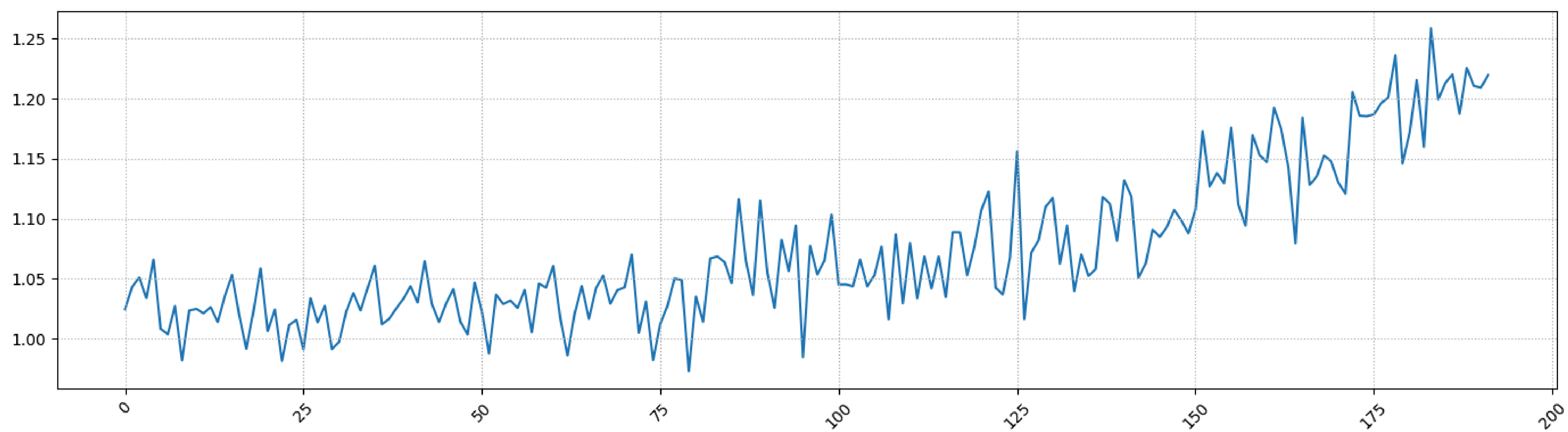


- Again six operating conditions
- ...And the series is similar to FD004

Selected Column

Let's plot one column in deeper detail for a single machine in FD001

In [11]:
```
tmp = data_sv_dict['train_FD001']
tmp = tmp[tmp['machine'] == tmp['machine'].iloc[0]]
util.plot_series(tmp['s4'], figsize=figsize)
```

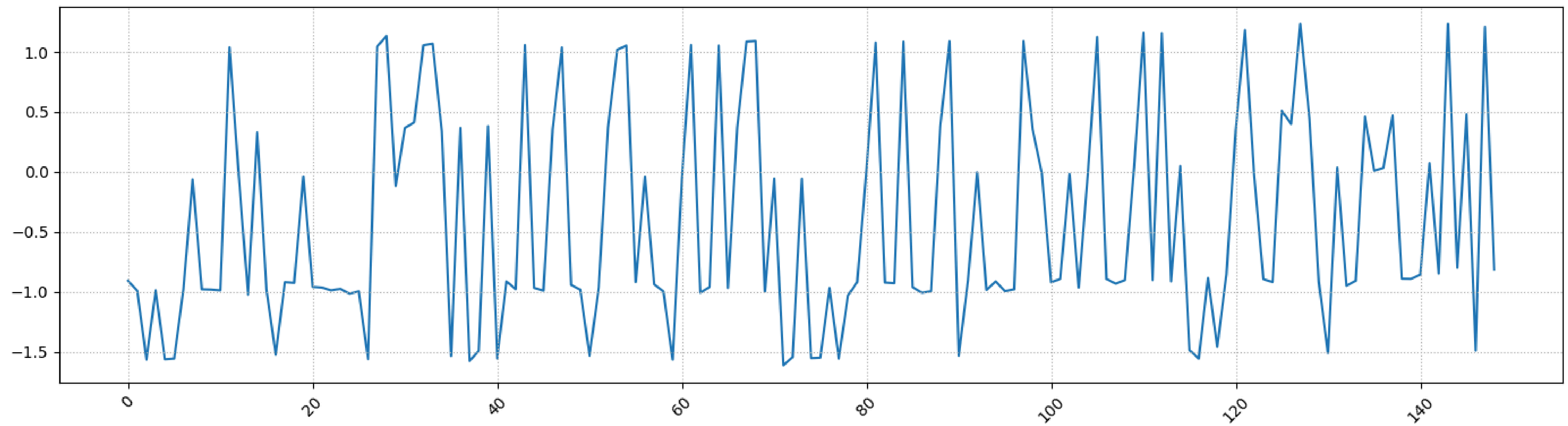


- A clear trend, possibly correlated to component wear

Selected Column

Let's see the same column for FD002

```
In [12]: tmp = data_sv_dict['train_FD002']
         tmp = tmp[tmp['machine'] == tmp['machine'].iloc[0]]
         util.plot_series(tmp['s4'], figsize=figsize)
```

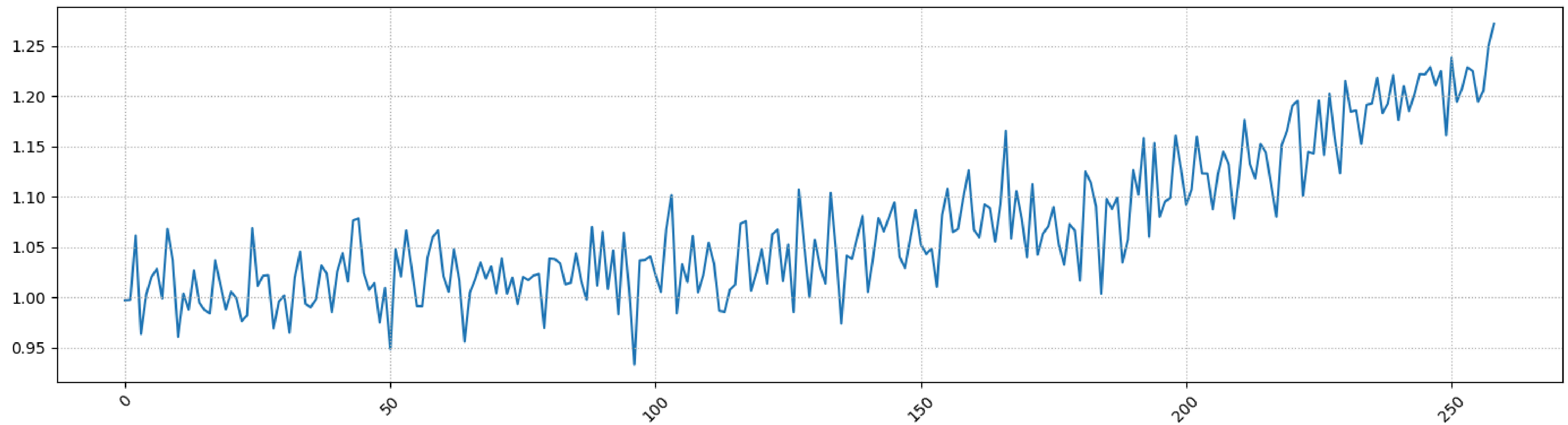


- The trend is still present, but weaker and hidden by wide oscillations

Selected Column

...And then the same column for FD003

In [13]:
```
tmp = data_sv_dict['train_FD003']
tmp = tmp[tmp['machine'] == tmp['machine'].iloc[0]]
util.plot_series(tmp['s4'], figsize=figsize)
```

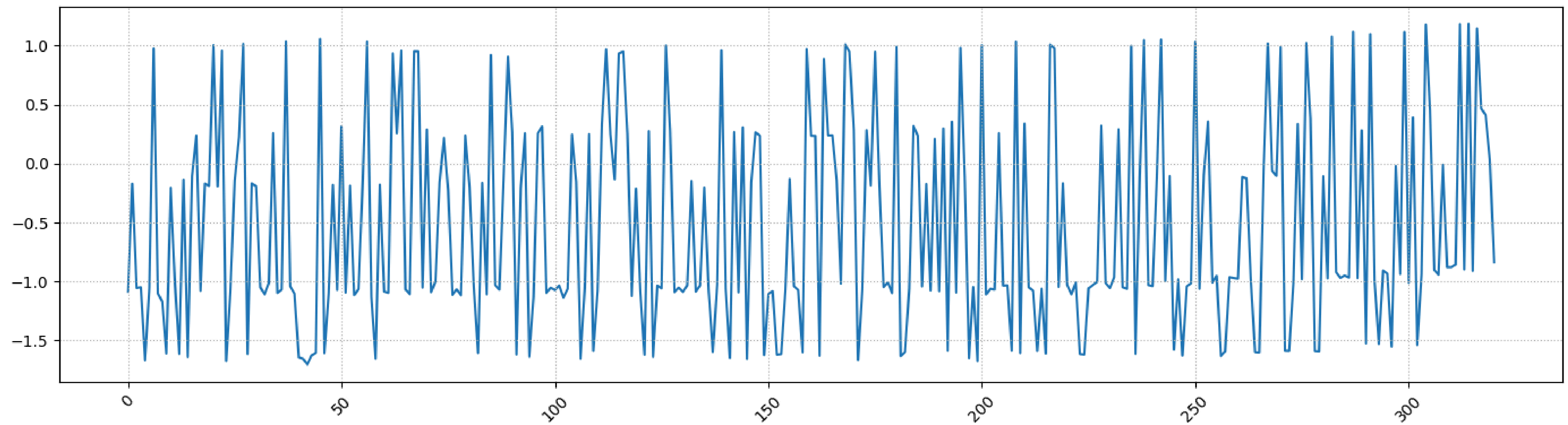


- Clear trend, with small oscillations that are more frequent than FD001

Selected Column

Let's see the same column for FD004

```
In [14]: tmp = data_sv_dict['train_FD004']
         tmp = tmp[tmp['machine'] == tmp['machine'].iloc[0]]
         util.plot_series(tmp['s4'], figsize=figsize)
```



- Very weak trend, wide and frequent oscillations