

Base Characteristics

In [1]:

```
1  # Graph: Distribution of engine lifespans (histogram)
2
3  def plot_engine_lifespan_hist(df, dataset_name="FD001", bins=20, savepath=None, ax=None):
4      """
5      Plot a histogram of engine lifespans (max cycles per unit) using the
6      C-MAPSS column names: 'unit_number' and 'time_in_cycles'.
7
8      Args:
9          df (pd.DataFrame): Raw or preprocessed dataset with required columns.
10         dataset_name (str): Label used in the plot title.
11         bins (int): Number of histogram bins.
12         savepath (str | Path | None): If provided, saves the figure.
13         ax (matplotlib.axes.Axes | None): Optional axes to draw on.
14
15     Returns:
16         pd.Series: Lifespan (max cycles) per engine, indexed by unit_number.
17     """
18     required_cols = {"unit_number", "time_in_cycles"}
19     missing = required_cols - set(df.columns)
20     if missing:
21         raise KeyError(f"Missing columns {missing}. Expected {required_cols}.")
22
23     # Compute lifespan per engine (max cycles before failure)
24     lifespans = df.groupby("unit_number")["time_in_cycles"].max().sort_values()
25
26     # Prepare axes
27     created_fig = False
28     if ax is None:
29         fig, ax = plt.subplots(figsize=(10, 6))
30         created_fig = True
31
32     # Plot histogram
33     ax.hist(lifespans, bins=bins, edgecolor="black")
34     ax.set_title(f"Distribution of Engine Lifespans ({dataset_name})")
35     ax.set_xlabel("Max Cycles Before Failure")
36     ax.set_ylabel("Number of Engines")
37     ax.grid(True, linestyle="--", alpha=0.5)
38
39     # Annotate mean ± std for quick reference
40     mean_cycles = lifespans.mean()
41     std_cycles = lifespans.std()
```

```

42     ax.axvline(mean_cycles, linestyle="--", linewidth=1)
43     ax.text(mean_cycles, ax.get_ylim()[1] * 0.95,
44             f"mean={mean_cycles:.1f}\no={std_cycles:.1f}",
45             ha="center", va="top")
46
47     if savepath:
48         plt.savefig(savepath, dpi=300, bbox_inches="tight")
49
50     if created_fig:
51         plt.tight_layout()
52         plt.show()
53
54     return lifespans
55
56     #=====
57
58     # Graph: example rul trajectories
59     from typing import Iterable, Dict, List, Optional, Tuple
60
61     def plot_example_rul_trajectories(df, unit_ids, max_rul=130,
62                                     dataset_name="FD001", savepath=None):
63         """Line plots of RUL vs cycles for selected engines."""
64         import matplotlib.pyplot as plt
65         import pre_processing as pp
66
67         df = df.copy()
68         df = pp.calculate_rul(df, max_rul=max_rul)
69         plt.figure(figsize=(10, 6))
70
71         for uid in unit_ids:
72             g = df[df.unit_number == uid].sort_values('time_in_cycles')
73             plt.plot(g['time_in_cycles'], g['RUL'],
74                     marker='.', linewidth=1, label=f"Engine {uid}")
75
76         plt.title(f"Example RUL Trajectories ({dataset_name})")
77         plt.xlabel("Cycle")
78         plt.ylabel("RUL")
79         plt.legend(ncol=2, fontsize=9)
80         plt.grid(True, alpha=0.3)
81         plt.tight_layout()
82         plt.show()
83

```

```

84
85 #=====
86
87 # Graph: Sensor correlation heatmap
88 def plot_sensor_correlation_heatmap(df, dataset_name="FD001", method="pearson", savepath=None):
89     """
90     Plot a correlation heatmap for C-MAPSS sensor columns (sensor_measurement_*).
91
92     Args:
93         df (pd.DataFrame): DataFrame with sensor columns.
94         dataset_name (str): Label for the title.
95         method (str): Correlation method ('pearson', 'spearman', 'kendall').
96         savepath (str|Path|None): If provided, save the figure.
97
98     Returns:
99         pd.DataFrame: Correlation matrix used for the plot.
100     """
101     import matplotlib.pyplot as plt
102
103     sensor_cols = [c for c in df.columns if c.startswith("sensor_measurement")]
104     if not sensor_cols:
105         raise ValueError("No sensor_measurement_* columns found.")
106
107     corr = df[sensor_cols].corr(method=method)
108
109     # Prefer seaborn if available, else matplotlib fallback
110     try:
111         import seaborn as sns
112         plt.figure(figsize=(12, 10))
113         sns.heatmap(corr, cmap="coolwarm", center=0, square=True,
114                     linewidths=0.5, cbar_kws={"shrink": 0.8})
115     except Exception:
116         plt.figure(figsize=(12, 10))
117         im = plt.imshow(corr, cmap="coolwarm", vmin=-1, vmax=1)
118         plt.colorbar(im, fraction=0.046, pad=0.04)
119         labels = [s.replace("sensor_measurement_", "S") for s in sensor_cols]
120         plt.xticks(range(len(sensor_cols)), labels, rotation=90)
121         plt.yticks(range(len(sensor_cols)), labels)
122
123     plt.title(f"Sensor Correlation Heatmap ({dataset_name})")
124     plt.tight_layout()
125     if savepath:

```

```

126     plt.savefig(savepath, dpi=300, bbox_inches="tight")
127     plt.show()
128
129     return corr
130
131     #=====
132
133     import numpy as np
134     import pandas as pd
135
136     def summarize_sensor_correlations(df, method="pearson", top_k=5):
137         """
138         Summarize inter-sensor correlations and list top +/- pairs.
139
140         Returns:
141             stats (dict): mean/median of corr and |corr| over unique pairs
142             top_pos (pd.DataFrame): top-k most positively correlated pairs
143             top_neg (pd.DataFrame): top-k most negatively correlated pairs
144             corr (pd.DataFrame): full correlation matrix for reuse
145         """
146         sensor_cols = [c for c in df.columns if c.startswith("sensor_measurement")]
147         if len(sensor_cols) < 2:
148             raise ValueError("Need at least two sensor_measurement_* columns.")
149
150         corr = df[sensor_cols].corr(method=method)
151
152         # take upper triangle (unique pairs, exclude diagonal)
153         iu = np.triu_indices_from(corr, k=1)
154         pair_list = []
155         for i, j in zip(iu[0], iu[1]):
156             pair_list.append({
157                 "sensor_a": sensor_cols[i],
158                 "sensor_b": sensor_cols[j],
159                 "corr": corr.iloc[i, j],
160                 "abs_corr": abs(corr.iloc[i, j]),
161             })
162         pairs = pd.DataFrame(pair_list)
163
164         # summary stats
165         stats = {
166             "mean_corr": float(pairs["corr"].mean()),
167             "median_corr": float(pairs["corr"].median()),

```

```
168         "mean_abs_corr": float(pairs["abs_corr"].mean()),
169         "median_abs_corr": float(pairs["abs_corr"].median()),
170         "n_pairs": int(len(pairs)),
171     }
172
173     # top-k lists
174     top_pos = pairs.sort_values("corr", ascending=False).head(top_k).reset_index(drop=True)
175     top_neg = pairs.sort_values("corr", ascending=True).head(top_k).reset_index(drop=True)
176
177     return stats, top_pos, top_neg, corr
178
```


In [2]:

```
1  # Load Dataset
2
3
4  # Standard Libs
5  from pathlib import Path
6  import numpy as np, pandas as pd
7  import matplotlib.pyplot as plt
8  import joblib
9
10
11 # Project modules
12 import data_loader as dl
13 import pre_processing as pp
14 import evaluator as ev
15 import base_model as base
16 import lstm_model as lstm
17 import cnn_model as cnn
18 import cnn_lstm_model as cnnlstm
19
20 # ---- Paths ----
21 ROOT = Path.cwd()
22 CMAPS = ROOT / "CMAPS" # keep correct folder case
23 # ==== Minimal config you tweak next time ====
24 DATASET = "FD001" # <- change this to FD002/FD003/FD004 Later
25 SEQ_LEN = 30 # sliding window
26 MAX_RUL = 130 # RUL clipping
27 VAL_SPLIT = 0.30 # val split by unit
28
29 # Files derived from DATASET (so you edit one line only)
30 TRAIN_PATH = CMAPS / f"train_{DATASET}.txt"
31 TEST_PATH = CMAPS / f"test_{DATASET}.txt"
32 RUL_PATH = CMAPS / f"RUL_{DATASET}.txt"
33
34 # Artifacts folder for this dataset
35 ART_DIR = ROOT / f"{DATASET} data & artefacts"
36 ART_DIR.mkdir(exist_ok=True)
37
38 print(f"backend: torch | dataset: {DATASET}")
39 print("Train:", TRAIN_PATH.name, "| Test:", TEST_PATH.name, "| RUL:", RUL_PATH.name)
40
41
```

```

42 # --- Load FD001 ---
43 train_df = dl.load_raw_data(CMAPS / f"train_{DATASET}.txt")
44 test_df, rul_df = dl.load_test_data(
45     CMAPS / f"test_{DATASET}.txt",
46     CMAPS / f"RUL_{DATASET}.txt"
47 )
48
49 print("Loaded.")
50 print("  train_df:", train_df.shape, "  test_df:", test_df.shape, "  rul_df:", rul_df.shape)
51 assert train_df.shape[1] == 26 and test_df.shape[1] == 26

```

WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.
 WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.
 WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.
 WARNING:root:Limited tf.summary API due to missing TensorBoard installation.
 WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.
 WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.
 WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.

backend: torch | dataset: FD001

Train: train_FD001.txt | Test: test_FD001.txt | RUL: RUL_FD001.txt

Loaded.

train_df: (20631, 26) test_df: (13096, 26) rul_df: (100, 1)

Preamble

```
In [3]: 1 dl.inspect_data(train_df)
```

```
Shape: (20631, 26)
```

```
Unique engines: 100
```

```
Missing values:  
0
```

```
Max cycles per engine:
```

```
count    100.000000  
mean      206.310000  
std        46.342749  
min       128.000000  
25%       177.000000  
50%       199.000000  
75%       229.250000  
max       362.000000
```

```
Name: time_in_cycles, dtype: float64
```

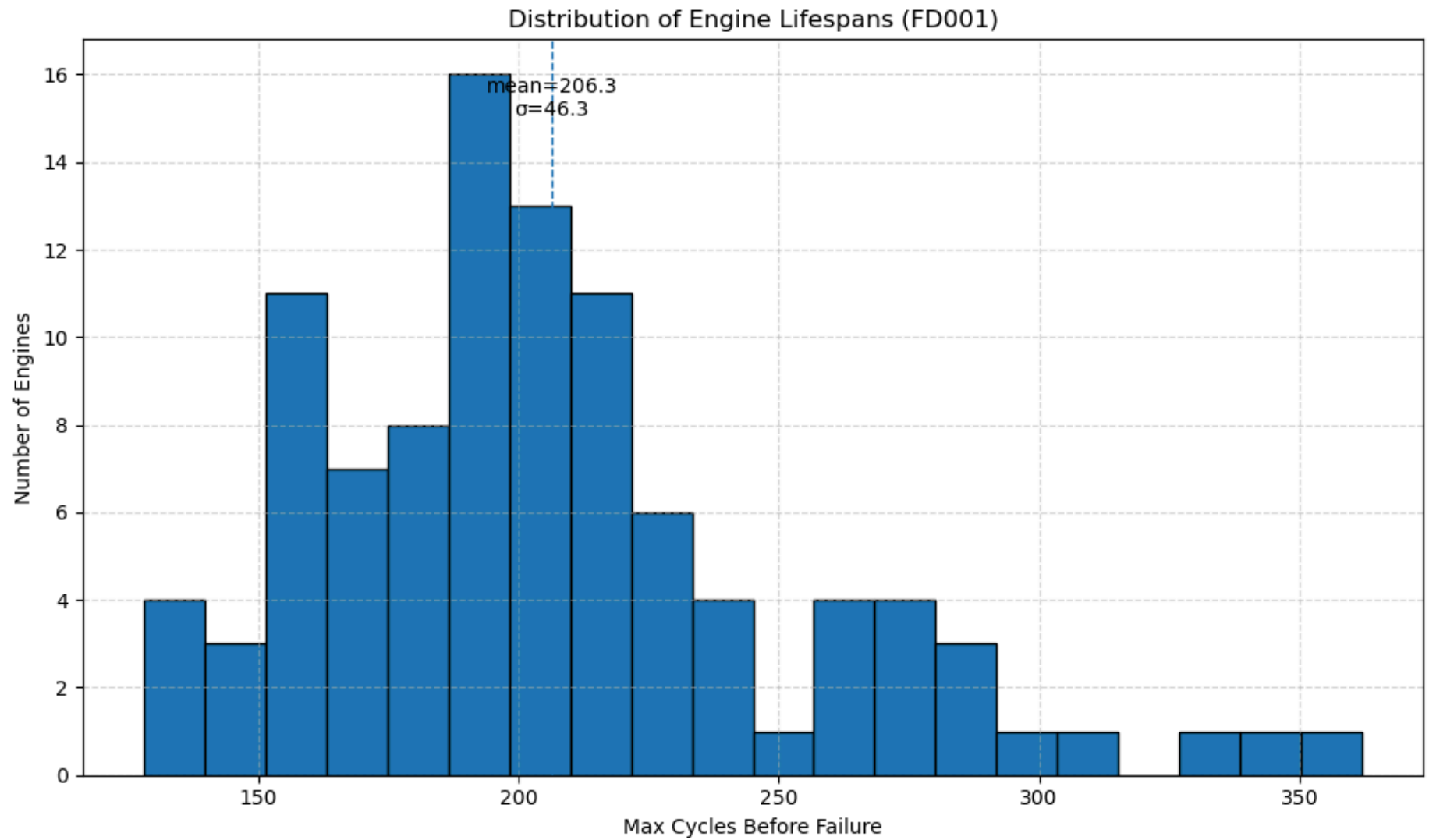
```
First 5 rows:
```

	unit_number	time_in_cycles	op_setting_1	op_setting_2	op_setting_3	sensor_measurement_1	sensor_measurement_2	sensor_measurement_3	se
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	

```
5 rows × 26 columns
```



```
In [4]: 1 plot_engine_lifespan_hist(train_df, dataset_name=DATASET)
```



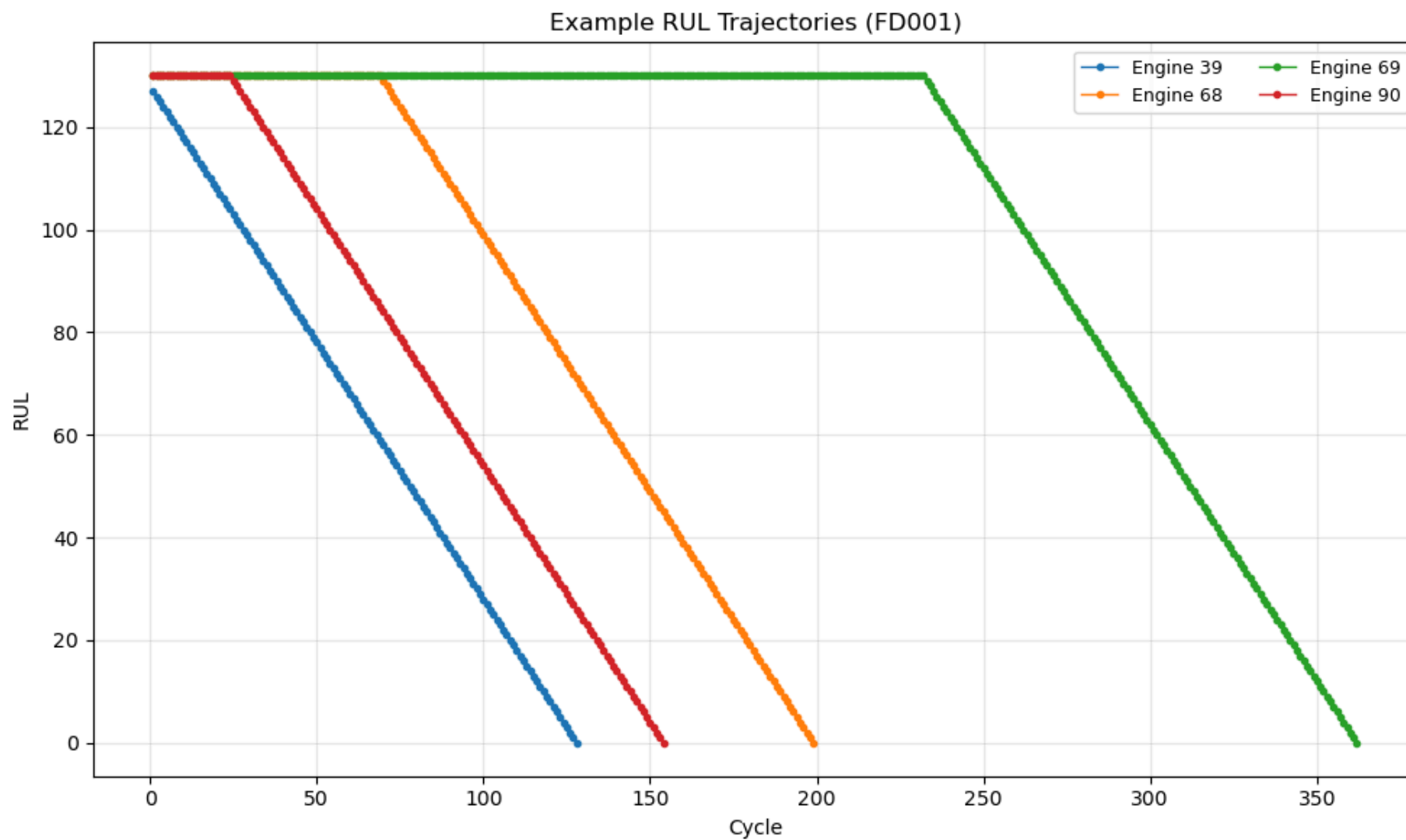
```
Out[4]: unit_number
      39      128
      91      135
      70      137
      57      137
      58      147
      ...
      83      293
      67      313
      96      336
      92      341
      69      362
Name: time_in_cycles, Length: 100, dtype: int64
```

In [5]:

```
1 import numpy as np
2
3 def select_representative_units(df, random_state=42):
4     """
5     Select representative engine units:
6     - Shortest-lived
7     - Median-lived
8     - Longest-lived
9     - One random unit (reproducible with random_state)
10
11     Args:
12         df (pd.DataFrame): Engine dataset with 'unit_number' and 'time_in_cycles'.
13         random_state (int): Seed for reproducibility.
14
15     Returns:
16         list[int]: List of selected engine IDs.
17     """
18     # Compute max cycles per engine
19     lifespans = df.groupby("unit_number")["time_in_cycles"].max().sort_values()
20
21     # Shortest-Lived
22     shortest = lifespans.index[0]
23     # Median-Lived
24     median = lifespans.index[len(lifespans) // 2]
25     # Longest-Lived
26     longest = lifespans.index[-1]
27     # One random
28     rng = np.random.default_rng(random_state)
29     random_unit = rng.choice(lifespans.index)
30
31     return [shortest, median, longest, int(random_unit)]
32
```

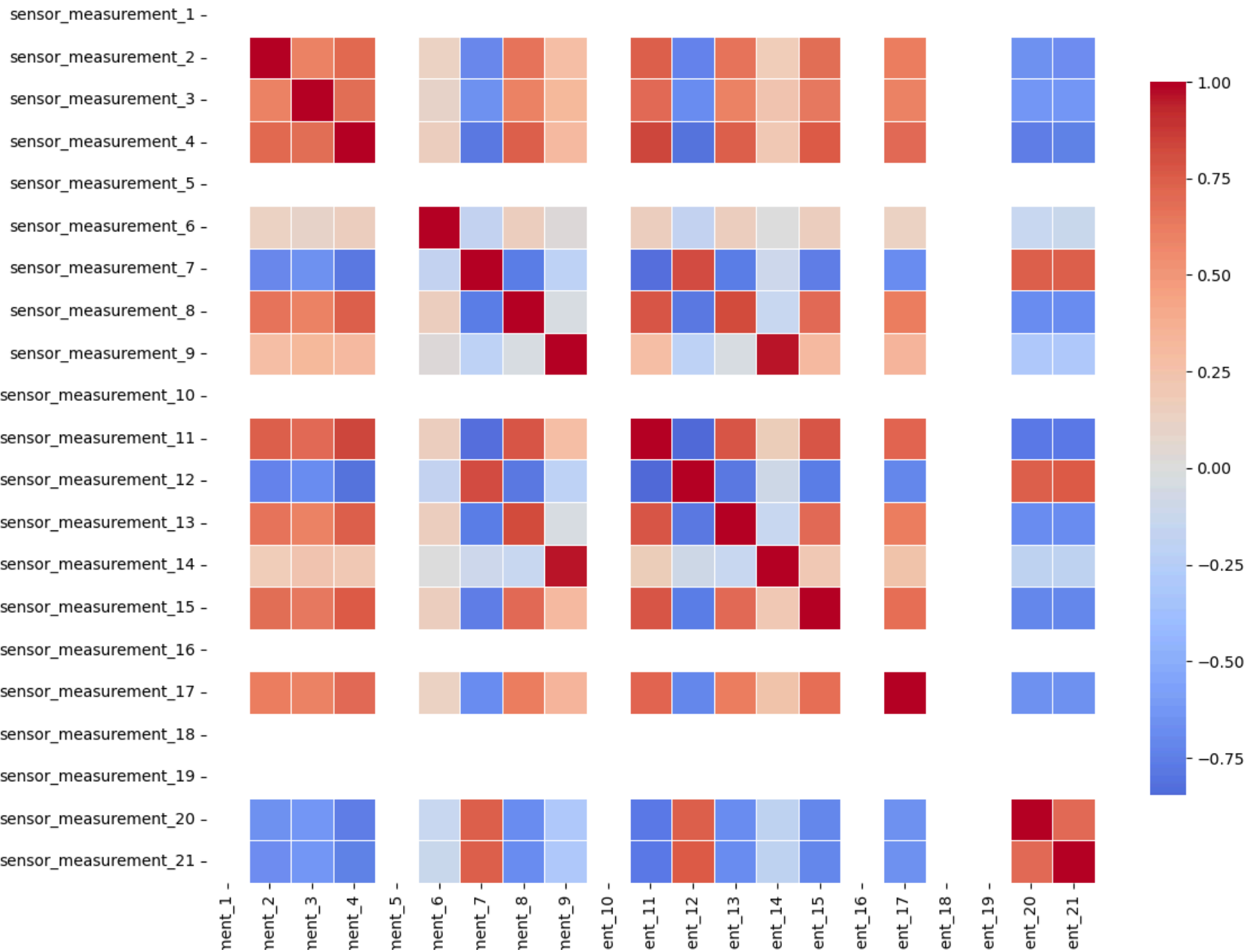
```
In [6]: 1 example_units = select_representative_units(train_df, random_state=42)
2 print("Selected engines:", example_units)
3
4 plot_example_rul_trajectories(train_df, unit_ids=example_units,
5                               max_rul=MAX_RUL, dataset_name=DATASET)
6
```

Selected engines: [39, 68, 69, 90]



```
In [7]: 1 plot_sensor_correlation_heatmap(train_df, dataset_name=DATASET)
```


Sensor Correlation Heatmap (FD001)



Out[7]:

	sensor_measurement_1	sensor_measurement_2	sensor_measurement_3	sensor_measurement_4	sensor_measurement_5	ser
sensor_measurement_1	NaN	NaN	NaN	NaN	NaN	
sensor_measurement_2	NaN	1.000000	0.602610	0.714949	NaN	
sensor_measurement_3	NaN	0.602610	1.000000	0.678413	NaN	
sensor_measurement_4	NaN	0.714949	0.678413	1.000000	NaN	
sensor_measurement_5	NaN	NaN	NaN	NaN	NaN	
sensor_measurement_6	NaN	0.132242	0.116039	0.150480	NaN	
sensor_measurement_7	NaN	-0.702136	-0.664595	-0.793130	NaN	
sensor_measurement_8	NaN	0.662325	0.602481	0.746852	NaN	
sensor_measurement_9	NaN	0.273764	0.322964	0.297429	NaN	
sensor_measurement_10	NaN	NaN	NaN	NaN	NaN	
sensor_measurement_11	NaN	0.740020	0.695900	0.830136	NaN	
sensor_measurement_12	NaN	-0.724578	-0.680307	-0.815591	NaN	
sensor_measurement_13	NaN	0.661792	0.600963	0.745158	NaN	
sensor_measurement_14	NaN	0.179752	0.237137	0.190748	NaN	
sensor_measurement_15	NaN	0.675975	0.639921	0.758459	NaN	
sensor_measurement_16	NaN	NaN	NaN	NaN	NaN	
sensor_measurement_17	NaN	0.629886	0.600017	0.703499	NaN	
sensor_measurement_18	NaN	NaN	NaN	NaN	NaN	
sensor_measurement_19	NaN	NaN	NaN	NaN	NaN	
sensor_measurement_20	NaN	-0.661841	-0.625941	-0.748067	NaN	
sensor_measurement_21	NaN	-0.668050	-0.633901	-0.745193	NaN	

21 rows × 21 columns



```
In [8]: 1 stats, top_pos, top_neg, corr = summarize_sensor_correlations(train_df, method="pearson", top_k=5)
2
3 print("Correlation summary:", {k: round(v, 3) if isinstance(v, float) else v for k, v in stats.items()})
4
5 display(top_pos.round(3))
6 display(top_neg.round(3))
```

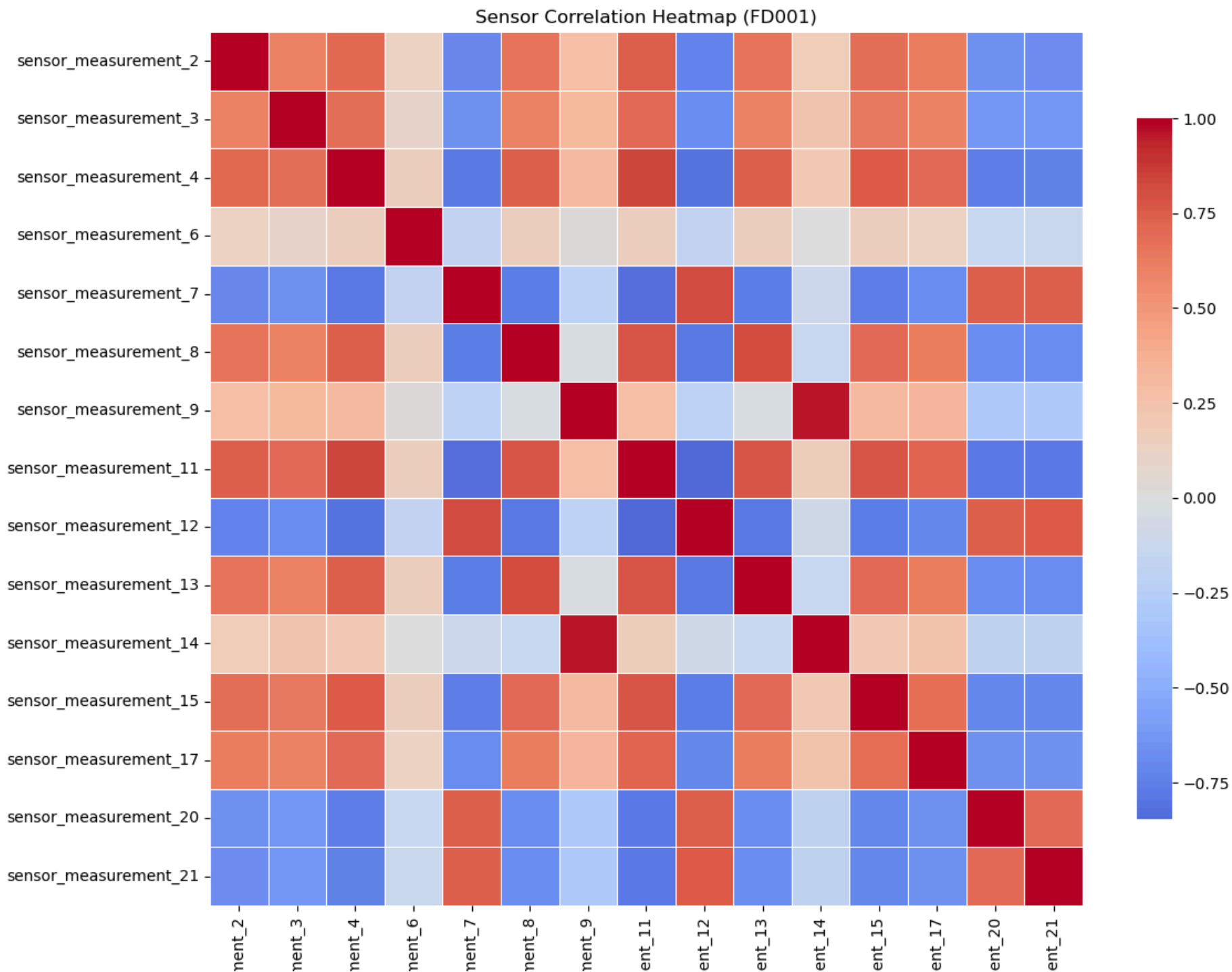
Correlation summary: {'mean_corr': 0.033, 'median_corr': 0.132, 'mean_abs_corr': 0.522, 'median_abs_corr': 0.662, 'n_pairs': 210}

	sensor_a	sensor_b	corr	abs_corr
0	sensor_measurement_9	sensor_measurement_14	0.963	0.963
1	sensor_measurement_4	sensor_measurement_11	0.830	0.830
2	sensor_measurement_8	sensor_measurement_13	0.826	0.826
3	sensor_measurement_7	sensor_measurement_12	0.813	0.813
4	sensor_measurement_8	sensor_measurement_11	0.782	0.782

	sensor_a	sensor_b	corr	abs_corr
0	sensor_measurement_11	sensor_measurement_12	-0.847	0.847
1	sensor_measurement_7	sensor_measurement_11	-0.823	0.823
2	sensor_measurement_4	sensor_measurement_12	-0.816	0.816
3	sensor_measurement_4	sensor_measurement_7	-0.793	0.793
4	sensor_measurement_12	sensor_measurement_13	-0.788	0.788

```
In [9]: 1 train_clean = pp.drop_flat_sensors(train_df.copy())
```

```
In [10]: 1 plot_sensor_correlation_heatmap(train__clean, dataset_name=DATASET)
```

sensor_measurel
sensor_measurel
sensor_measurel
sensor_measurel
sensor_measurel
sensor_measurel
sensor_measurel
sensor_measurem
sensor_measurem
sensor_measurem
sensor_measurem
sensor_measurem
sensor_measurem
sensor_measurem
sensor_measurem

Out[10]:

	sensor_measurement_2	sensor_measurement_3	sensor_measurement_4	sensor_measurement_6	sensor_measurement_7	ser
sensor_measurement_2	1.000000	0.602610	0.714949	0.132242	-0.702136	
sensor_measurement_3	0.602610	1.000000	0.678413	0.116039	-0.664595	
sensor_measurement_4	0.714949	0.678413	1.000000	0.150480	-0.793130	
sensor_measurement_6	0.132242	0.116039	0.150480	1.000000	-0.155720	
sensor_measurement_7	-0.702136	-0.664595	-0.793130	-0.155720	1.000000	
sensor_measurement_8	0.662325	0.602481	0.746852	0.152161	-0.767132	
sensor_measurement_9	0.273764	0.322964	0.297429	0.019347	-0.217835	
sensor_measurement_11	0.740020	0.695900	0.830136	0.160014	-0.822805	
sensor_measurement_12	-0.724578	-0.680307	-0.815591	-0.155884	0.812713	
sensor_measurement_13	0.661792	0.600963	0.745158	0.158276	-0.764611	
sensor_measurement_14	0.179752	0.237137	0.190748	-0.002112	-0.110053	
sensor_measurement_15	0.675975	0.639921	0.758459	0.149042	-0.747051	
sensor_measurement_17	0.629886	0.600017	0.703499	0.130810	-0.692893	
sensor_measurement_20	-0.661841	-0.625941	-0.748067	-0.141419	0.736163	
sensor_measurement_21	-0.668050	-0.633901	-0.745193	-0.137419	0.737447	



```
In [11]: 1 stats, top_pos, top_neg, corr = summarize_sensor_correlations(train__clean, method="pearson", top_k=5)
2
3 print("Correlation summary:", {k: round(v, 3) if isinstance(v, float) else v for k, v in stats.items()})
4
5 display(top_pos.round(3))
6 display(top_neg.round(3))
```

Correlation summary: {'mean_corr': 0.033, 'median_corr': 0.132, 'mean_abs_corr': 0.522, 'median_abs_corr': 0.662, 'n_pairs': 105}

	sensor_a	sensor_b	corr	abs_corr
0	sensor_measurement_9	sensor_measurement_14	0.963	0.963
1	sensor_measurement_4	sensor_measurement_11	0.830	0.830
2	sensor_measurement_8	sensor_measurement_13	0.826	0.826
3	sensor_measurement_7	sensor_measurement_12	0.813	0.813
4	sensor_measurement_8	sensor_measurement_11	0.782	0.782

	sensor_a	sensor_b	corr	abs_corr
0	sensor_measurement_11	sensor_measurement_12	-0.847	0.847
1	sensor_measurement_7	sensor_measurement_11	-0.823	0.823
2	sensor_measurement_4	sensor_measurement_12	-0.816	0.816
3	sensor_measurement_4	sensor_measurement_7	-0.793	0.793
4	sensor_measurement_12	sensor_measurement_13	-0.788	0.788

```
In [ ]: 1
```