# Base Characteristics

In [1]:

```python
# Graph: Distribution of engine lifespans (histogram)

def plot_engine_lifespan_hist(df, dataset_name="FD001", bins=20, savepath=None, ax=None):
    """
    Plot a histogram of engine lifespans (max cycles per unit) using the
    C-MAPSS column names: 'unit_number' and 'time_in_cycles'.

    Args:
        df (pd.DataFrame): Raw or preprocessed dataset with required columns.
        dataset_name (str): Label used in the plot title.
        bins (int): Number of histogram bins.
        savepath (str | Path | None): If provided, saves the figure.
        ax (matplotlib.axes.Axes | None): Optional axes to draw on.

    Returns:
        pd.Series: Lifespan (max cycles) per engine, indexed by unit_number.
    """
    required_cols = {"unit_number", "time_in_cycles"}
    missing = required_cols - set(df.columns)
    if missing:
        raise KeyError(f"Missing columns {missing}. Expected {required_cols}.")

    # Compute lifespan per engine (max cycles before failure)
    lifespans = df.groupby("unit_number")["time_in_cycles"].max().sort_values()

    # Prepare axes
    created_fig = False
    if ax is None:
        fig, ax = plt.subplots(figsize=(10, 6))
        created_fig = True

    # Plot histogram
    ax.hist(lifespans, bins=bins, edgecolor="black")
    ax.set_title(f"Distribution of Engine Lifespans ({dataset_name})")
    ax.set_xlabel("Max Cycles Before Failure")
    ax.set_ylabel("Number of Engines")
    ax.grid(True, linestyle="--", alpha=0.5)

    # Annotate mean ± std for quick reference
    mean_cycles = lifespans.mean()
    std_cycles = lifespans.std()
```

```python
    ax.axvline(mean_cycles, linestyle="--", linewidth=1)
    ax.text(mean_cycles, ax.get_ylim()[1] * 0.95,
            f"mean={mean_cycles:.1f}\nσ={std_cycles:.1f}",
            ha="center", va="top")

    if savepath:
        plt.savefig(savepath, dpi=300, bbox_inches="tight")

    if created_fig:
        plt.tight_layout()
        plt.show()

    return lifespans

#===============================================================================================================

# Graph: example rul trajectories
from typing import Iterable, Dict, List, Optional, Tuple

def plot_example_rul_trajectories(df, unit_ids, max_rul=130,
                                   dataset_name="FD001", savepath=None):
    """Line plots of RUL vs cycles for selected engines."""
    import matplotlib.pyplot as plt
    import pre_processing as pp

    df = df.copy()
    df = pp.calculate_rul(df, max_rul=max_rul)
    plt.figure(figsize=(10, 6))

    for uid in unit_ids:
        g = df[df.unit_number == uid].sort_values('time_in_cycles')
        plt.plot(g['time_in_cycles'], g['RUL'],
                 marker='.', linewidth=1, label=f"Engine {uid}")

    plt.title(f"Example RUL Trajectories ({dataset_name})")
    plt.xlabel("Cycle")
    plt.ylabel("RUL")
    plt.legend(ncol=2, fontsize=9)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

```

```python

#==================================================================================================================

# Graph: Sensor correlation heatmap
def plot_sensor_correlation_heatmap(df, dataset_name="FD001", method="pearson", savepath=None):
    """
    Plot a correlation heatmap for C-MAPSS sensor columns (sensor_measurement_*).

    Args:
        df (pd.DataFrame): DataFrame with sensor columns.
        dataset_name (str): Label for the title.
        method (str): Correlation method ('pearson', 'spearman', 'kendall').
        savepath (str|Path|None): If provided, save the figure.

    Returns:
        pd.DataFrame: Correlation matrix used for the plot.
    """
    import matplotlib.pyplot as plt

    sensor_cols = [c for c in df.columns if c.startswith("sensor_measurement")]
    if not sensor_cols:
        raise ValueError("No sensor_measurement_* columns found.")

    corr = df[sensor_cols].corr(method=method)

    # Prefer seaborn if available, else matplotlib fallback
    try:
        import seaborn as sns
        plt.figure(figsize=(12, 10))
        sns.heatmap(corr, cmap="coolwarm", center=0, square=True,
                    linewidths=0.5, cbar_kws={"shrink": 0.8})
    except Exception:
        plt.figure(figsize=(12, 10))
        im = plt.imshow(corr, cmap="coolwarm", vmin=-1, vmax=1)
        plt.colorbar(im, fraction=0.046, pad=0.04)
        labels = [s.replace("sensor_measurement_", "S") for s in sensor_cols]
        plt.xticks(range(len(sensor_cols)), labels, rotation=90)
        plt.yticks(range(len(sensor_cols)), labels)

    plt.title(f"Sensor Correlation Heatmap ({dataset_name})")
    plt.tight_layout()
    if savepath:
```

```python
        plt.savefig(savepath, dpi=300, bbox_inches="tight")
    plt.show()

    return corr

#=====================================================================================================================

import numpy as np
import pandas as pd

def summarize_sensor_correlations(df, method="pearson", top_k=5):
    """
    Summarize inter-sensor correlations and list top +/- pairs.

    Returns:
        stats (dict): mean/median of corr and |corr| over unique pairs
        top_pos (pd.DataFrame): top-k most positively correlated pairs
        top_neg (pd.DataFrame): top-k most negatively correlated pairs
        corr (pd.DataFrame): full correlation matrix for reuse
    """
    sensor_cols = [c for c in df.columns if c.startswith("sensor_measurement")]
    if len(sensor_cols) < 2:
        raise ValueError("Need at least two sensor_measurement_* columns.")

    corr = df[sensor_cols].corr(method=method)

    # take upper triangle (unique pairs, exclude diagonal)
    iu = np.triu_indices_from(corr, k=1)
    pair_list = []
    for i, j in zip(iu[0], iu[1]):
        pair_list.append({
            "sensor_a": sensor_cols[i],
            "sensor_b": sensor_cols[j],
            "corr": corr.iloc[i, j],
            "abs_corr": abs(corr.iloc[i, j]),
        })
    pairs = pd.DataFrame(pair_list)

    # summary stats
    stats = {
        "mean_corr": float(pairs["corr"].mean()),
        "median_corr": float(pairs["corr"].median()),
```

```python
        "mean_abs_corr": float(pairs["abs_corr"].mean()),
        "median_abs_corr": float(pairs["abs_corr"].median()),
        "n_pairs": int(len(pairs)),
    }

    # top-k lists
    top_pos = pairs.sort_values("corr", ascending=False).head(top_k).reset_index(drop=True)
    top_neg = pairs.sort_values("corr", ascending=True ).head(top_k).reset_index(drop=True)

    return stats, top_pos, top_neg, corr

```

In [2]:

```python
# Load Dataset


# Standard libs
from pathlib import Path
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
import joblib


# Project modules
import data_loader as dl
import pre_processing as pp
import evaluator as ev
import base_model as base
import lstm_model as lstm
import cnn_model as cnn
import cnn_lstm_model as cnnlstm

# ---- Paths ----
ROOT = Path.cwd()
CMAPS = ROOT / "CMaps"  # keep correct folder case
# ==== Minimal config you tweak next time ====
DATASET = "FD002"       # <- change this to FD002/FD003/FD004 later
SEQ_LEN = 30            # sliding window
MAX_RUL = 130           # RUL clipping
VAL_SPLIT = 0.30        # val split by unit

# Files derived from DATASET (so you edit one line only)
TRAIN_PATH = CMAPS / f"train_{DATASET}.txt"
TEST_PATH  = CMAPS / f"test_{DATASET}.txt"
RUL_PATH   = CMAPS / f"RUL_{DATASET}.txt"

# Artifacts folder for this dataset
ART_DIR = ROOT / f"{DATASET} data & artefacts"
ART_DIR.mkdir(exist_ok=True)

print(f"backend: torch | dataset: {DATASET}")
print("Train:", TRAIN_PATH.name, "| Test:", TEST_PATH.name, "| RUL:", RUL_PATH.name)


```

```python
# --- Load FD001 ---
train_df = dl.load_raw_data(CMAPS / f"train_{DATASET}.txt")
test_df, rul_df = dl.load_test_data(
    CMAPS / f"test_{DATASET}.txt",
    CMAPS / f"RUL_{DATASET}.txt"
)

print("Loaded.")
print("  train_df:", train_df.shape, "  test_df:", test_df.shape, "  rul_df:", rul_df.shape)
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: FD002
Train: train_FD002.txt | Test: test_FD002.txt | RUL: RUL_FD002.txt
Loaded.
  train_df: (53759, 26)   test_df: (33991, 26)   rul_df: (259, 1)
```

# Preamble

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

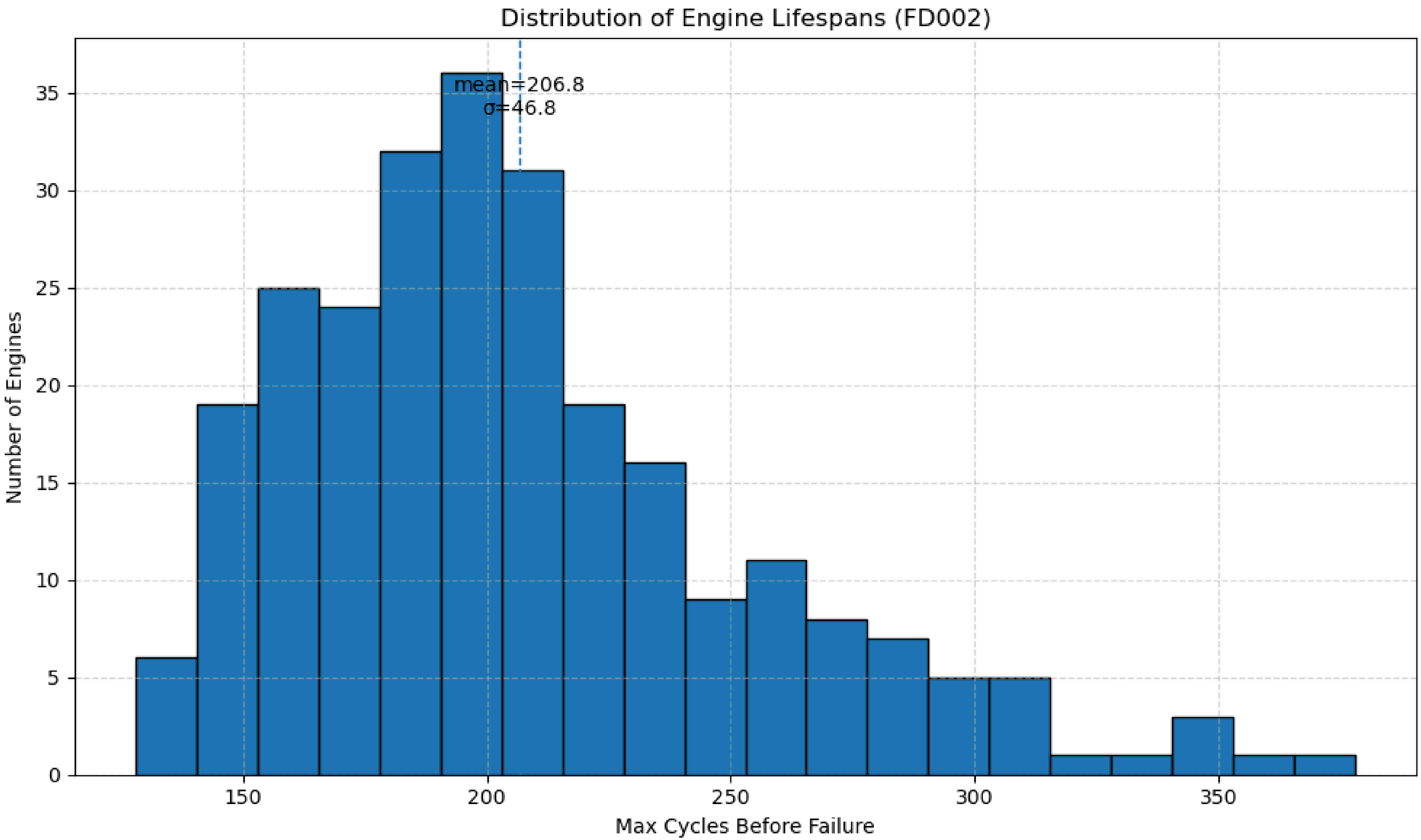
```
Shape: (53759, 26)

Unique engines: 260

Missing values:
 0

Max cycles per engine:
count    260.000000
mean     206.765385
std       46.782198
min      128.000000
25%      174.000000
50%      199.000000
75%      230.250000
max      378.000000
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 | 34.9983 | 0.8400 | 100.0 | 449.44 | 555.32 | 1358.61 | |
| **1** | 1 | 2 | 41.9982 | 0.8408 | 100.0 | 445.00 | 549.90 | 1353.22 | |
| **2** | 1 | 3 | 24.9988 | 0.6218 | 60.0 | 462.54 | 537.31 | 1256.76 | |
| **3** | 1 | 4 | 42.0077 | 0.8416 | 100.0 | 445.00 | 549.51 | 1354.03 | |
| **4** | 1 | 5 | 25.0005 | 0.6203 | 60.0 | 462.54 | 537.07 | 1257.71 | |

5 rows × 26 columns

In [4]:

```
plot_engine_lifespan_hist(train_df, dataset_name=DATASET)
```



```
Out[4]: unit_number
        244    128
        120    129
        192    133
        252    135
        69     136
              ...
        31     343
        118    344
        85     347
        88     365
        112    378
        Name: time_in_cycles, Length: 260, dtype: int64
```

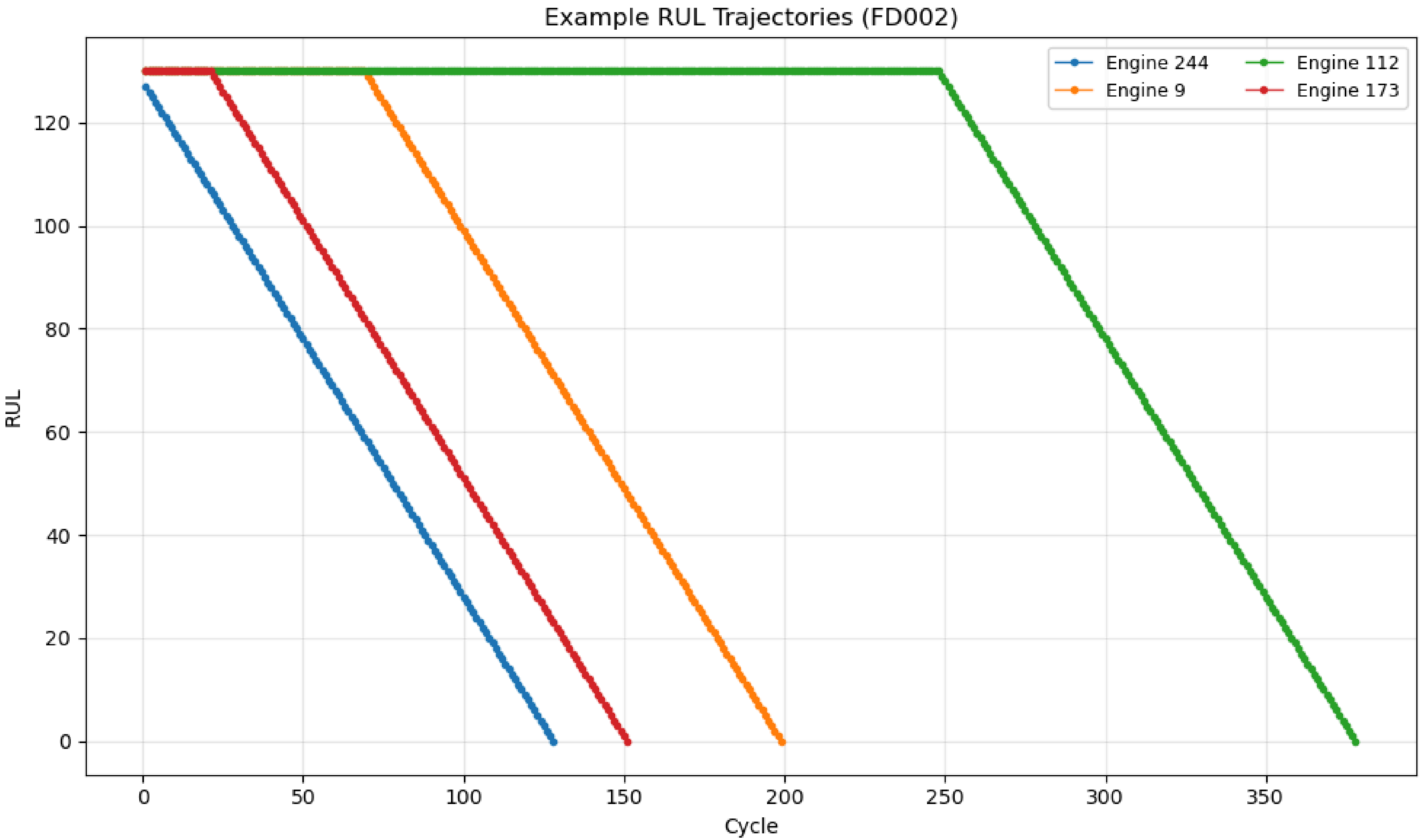
In [5]:

```python
import numpy as np

def select_representative_units(df, random_state=42):
    """
    Select representative engine units:
    - Shortest-lived
    - Median-lived
    - Longest-lived
    - One random unit (reproducible with random_state)

    Args:
        df (pd.DataFrame): Engine dataset with 'unit_number' and 'time_in_cycles'.
        random_state (int): Seed for reproducibility.

    Returns:
        list[int]: List of selected engine IDs.
    """
    # Compute max cycles per engine
    lifespans = df.groupby("unit_number")["time_in_cycles"].max().sort_values()

    # Shortest-lived
    shortest = lifespans.index[0]
    # Median-lived
    median = lifespans.index[len(lifespans) // 2]
    # Longest-lived
    longest = lifespans.index[-1]
    # One random
    rng = np.random.default_rng(random_state)
    random_unit = rng.choice(lifespans.index)

    return [shortest, median, longest, int(random_unit)]

```

In [6]:

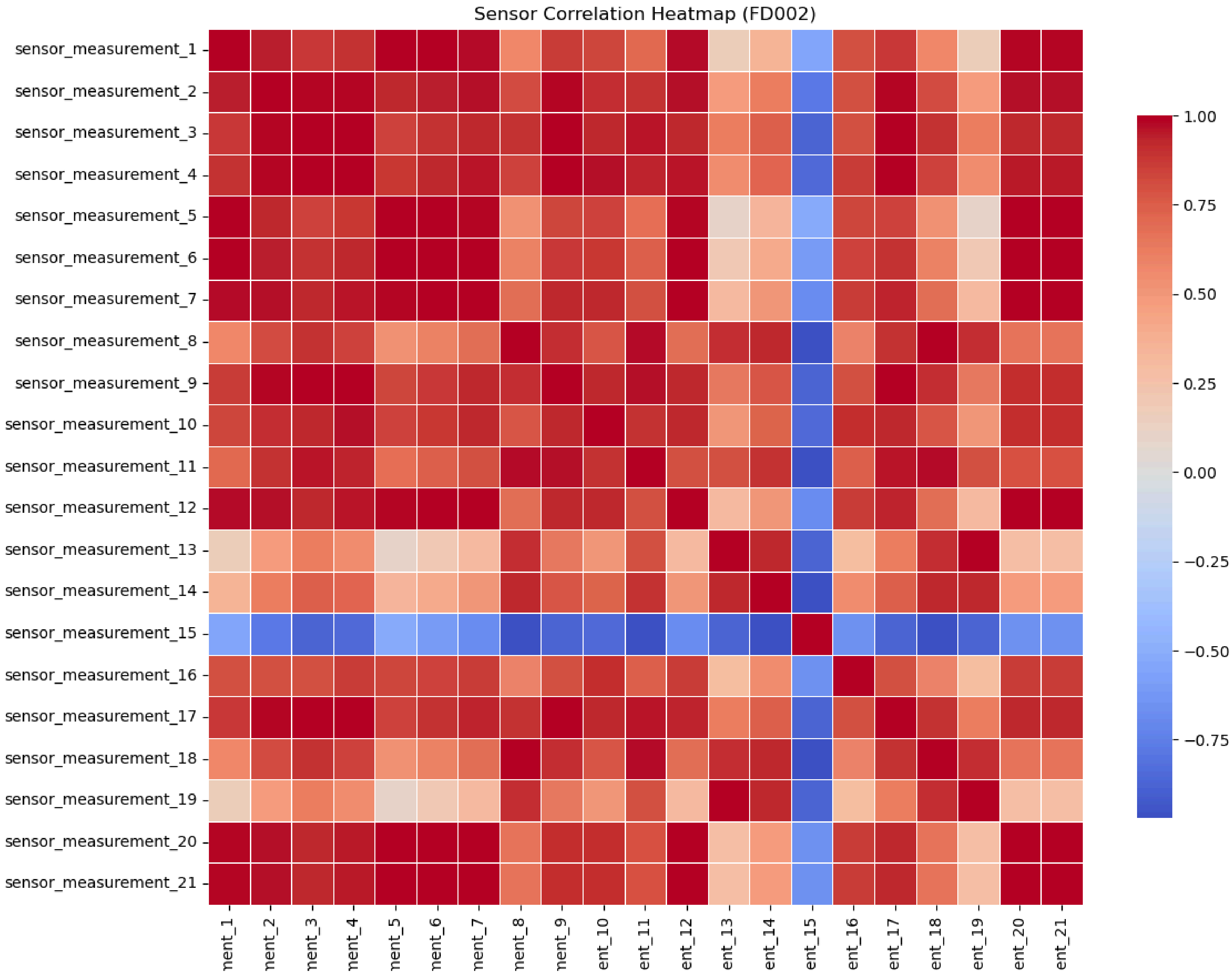
```python
example_units = select_representative_units(train_df, random_state=42)
print("Selected engines:", example_units)

plot_example_rul_trajectories(train_df, unit_ids=example_units,
                              max_rul=MAX_RUL, dataset_name=DATASET)

```

```
Selected engines: [244, 9, 112, 173]
```



In [7]:

```
plot_sensor_correlation_heatmap(train_df, dataset_name=DATASET)
```



Sensor Correlation Heatmap (FD002)
sensor_measurement_1
sensor_measurement_2
sensor_measurement_3
sensor_measurement_4
sensor_measurement_5
sensor_measurement_6
sensor_measurement_7
sensor_measurement_8
sensor_measurement_9
sensor_measurement_10
sensor_measurement_11
sensor_measurement_12
sensor_measurement_13
sensor_measurement_14
sensor_measurement_15
sensor_measurement_16
sensor_measurement_17
sensor_measurement_18
sensor_measurement_19
sensor_measurement_20
sensor_measurement_21
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

sensor_measurer
sensor_measurer
sensor_measurer
sensor_measurer
sensor_measurer
sensor_measurer
sensor_measurer
sensor_measurer
sensor_measurer
sensor_measurem
sensor_measurem
sensor_measurem
sensor_measurem
sensor_measurem
sensor_measurem
sensor_measurem
sensor_measurem
sensor_measurem
sensor_measurem
sensor_measurem
sensor_measurem

Out[7]:

| | sensor_measurement_1 | sensor_measurement_2 | sensor_measurement_3 | sensor_measurement_4 | sensor_measurement_5 | ser |
|---|---|---|---|---|---|---|
| **sensor_measurement_1** | 1.000000 | 0.944089 | 0.870963 | 0.898002 | 0.986372 | |
| **sensor_measurement_2** | 0.944089 | 1.000000 | 0.982225 | 0.981047 | 0.915808 | |
| **sensor_measurement_3** | 0.870963 | 0.982225 | 1.000000 | 0.989565 | 0.842951 | |
| **sensor_measurement_4** | 0.898002 | 0.981047 | 0.989565 | 1.000000 | 0.884242 | |
| **sensor_measurement_5** | 0.986372 | 0.915808 | 0.842951 | 0.884242 | 1.000000 | |
| **sensor_measurement_6** | 0.986424 | 0.944072 | 0.884795 | 0.919684 | 0.996311 | |
| **sensor_measurement_7** | 0.973142 | 0.968620 | 0.929054 | 0.956731 | 0.979787 | |
| **sensor_measurement_8** | 0.572652 | 0.810662 | 0.895718 | 0.843956 | 0.524331 | |
| **sensor_measurement_9** | 0.861836 | 0.978554 | 0.997806 | 0.987319 | 0.832968 | |
| **sensor_measurement_10** | 0.826591 | 0.905135 | 0.928979 | 0.961586 | 0.843905 | |
| **sensor_measurement_11** | 0.706111 | 0.895735 | 0.960683 | 0.936763 | 0.673774 | |
| **sensor_measurement_12** | 0.972867 | 0.968828 | 0.929534 | 0.957146 | 0.979487 | |
| **sensor_measurement_13** | 0.164474 | 0.480202 | 0.620970 | 0.544693 | 0.113778 | |
| **sensor_measurement_14** | 0.352782 | 0.623963 | 0.751865 | 0.715493 | 0.330514 | |
| **sensor_measurement_15** | -0.542743 | -0.777953 | -0.875928 | -0.846726 | -0.525384 | |
| **sensor_measurement_16** | 0.793789 | 0.805280 | 0.804821 | 0.859130 | 0.824095 | |
| **sensor_measurement_17** | 0.873265 | 0.983065 | 0.998680 | 0.990213 | 0.845626 | |
| **sensor_measurement_18** | 0.572171 | 0.810312 | 0.895446 | 0.843647 | 0.523827 | |
| **sensor_measurement_19** | 0.164334 | 0.480073 | 0.620839 | 0.544563 | 0.113635 | |
| **sensor_measurement_20** | 0.977703 | 0.962425 | 0.917144 | 0.946478 | 0.985714 | |
| **sensor_measurement_21** | 0.977718 | 0.962416 | 0.917125 | 0.946469 | 0.985727 | |

21 rows × 21 columns

In [8]:

```
stats, top_pos, top_neg, corr = summarize_sensor_correlations(train_df, method="pearson", top_k=5)

print("Correlation summary:", {k: round(v, 3) if isinstance(v, float) else v for k, v in stats.items()})

display(top_pos.round(3))
display(top_neg.round(3))
```

```
Correlation summary: {'mean_corr': 0.633, 'median_corr': 0.845, 'mean_abs_corr': 0.783, 'median_abs_corr': 0.874, 'n_pa
irs': 210}
```

| | sensor_a | sensor_b | corr | abs_corr |
|---|---|---|---|---|
| **0** | sensor_measurement_8 | sensor_measurement_18 | 1.000 | 1.000 |
| **1** | sensor_measurement_13 | sensor_measurement_19 | 1.000 | 1.000 |
| **2** | sensor_measurement_7 | sensor_measurement_12 | 1.000 | 1.000 |
| **3** | sensor_measurement_20 | sensor_measurement_21 | 1.000 | 1.000 |
| **4** | sensor_measurement_7 | sensor_measurement_20 | 0.999 | 0.999 |

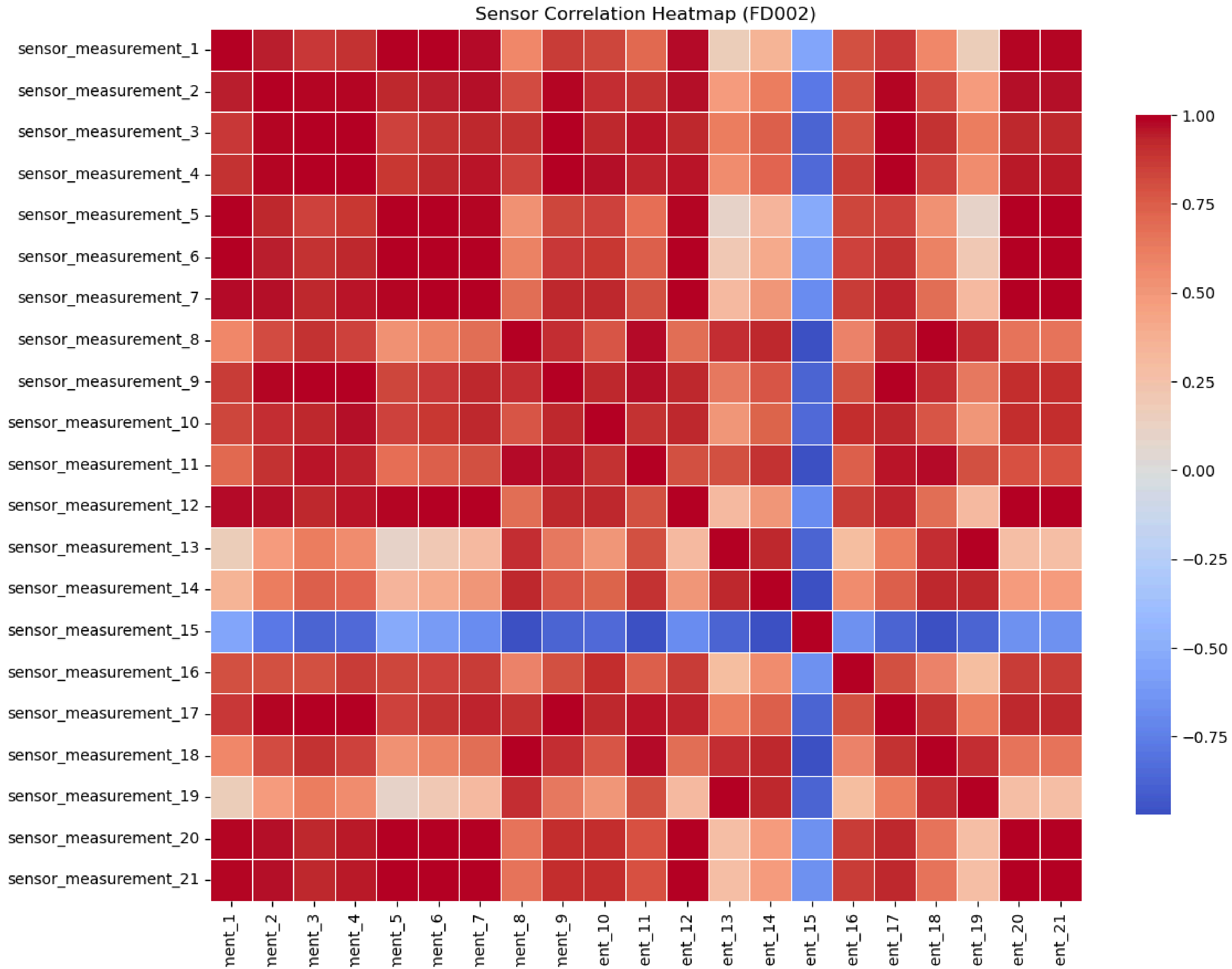
| | sensor_a | sensor_b | corr | abs_corr |
|---|---|---|---|---|
| **0** | sensor_measurement_15 | sensor_measurement_18 | -0.971 | 0.971 |
| **1** | sensor_measurement_8 | sensor_measurement_15 | -0.971 | 0.971 |
| **2** | sensor_measurement_11 | sensor_measurement_15 | -0.965 | 0.965 |
| **3** | sensor_measurement_14 | sensor_measurement_15 | -0.957 | 0.957 |
| **4** | sensor_measurement_9 | sensor_measurement_15 | -0.885 | 0.885 |

In [9]:

```
train__clean = pp.drop_flat_sensors(train_df.copy())
```

In [10]:

```
plot_sensor_correlation_heatmap(train__clean, dataset_name=DATASET)
```


Sensor Correlation Heatmap (FD002)
sensor_measurement_1
sensor_measurement_2
sensor_measurement_3
sensor_measurement_4
sensor_measurement_5
sensor_measurement_6
sensor_measurement_7
sensor_measurement_8
sensor_measurement_9
sensor_measurement_10
sensor_measurement_11
sensor_measurement_12
sensor_measurement_13
sensor_measurement_14
sensor_measurement_15
sensor_measurement_16
sensor_measurement_17
sensor_measurement_18
sensor_measurement_19
sensor_measurement_20
sensor_measurement_21
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

| sensor_measurer | sensor_measurer | sensor_measurer | sensor_measurer | sensor_measurer | sensor_measurer | sensor_measurer | sensor_measurer | sensor_measurer | sensor_measurem | sensor_measurem | sensor_measurem | sensor_measurem | sensor_measurem | sensor_measurem | sensor_measurem | sensor_measurem | sensor_measurem | sensor_measurem | sensor_measurem | sensor_measurem |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Out[10]:

| | sensor_measurement_1 | sensor_measurement_2 | sensor_measurement_3 | sensor_measurement_4 | sensor_measurement_5 | sensor_measurement_ |
|---|---|---|---|---|---|---|
| ment_1 | 1.000000 | 0.944089 | 0.870963 | 0.898002 | 0.986372 | 0.98642 |
| ment_2 | 0.944089 | 1.000000 | 0.982225 | 0.981047 | 0.915808 | 0.94407 |
| ment_3 | 0.870963 | 0.982225 | 1.000000 | 0.989565 | 0.842951 | 0.88479 |
| ment_4 | 0.898002 | 0.981047 | 0.989565 | 1.000000 | 0.884242 | 0.91968 |
| ment_5 | 0.986372 | 0.915808 | 0.842951 | 0.884242 | 1.000000 | 0.99631 |
| ment_6 | 0.986424 | 0.944072 | 0.884795 | 0.919684 | 0.996311 | 1.00000 |
| ment_7 | 0.973142 | 0.968620 | 0.929054 | 0.956731 | 0.979787 | 0.99327 |
| ment_8 | 0.572652 | 0.810662 | 0.895718 | 0.843956 | 0.524331 | 0.59452 |
| ment_9 | 0.861836 | 0.978554 | 0.997806 | 0.987319 | 0.832968 | 0.87620 |
| nent_10 | 0.826591 | 0.905135 | 0.928979 | 0.961586 | 0.843905 | 0.87823 |
| nent_11 | 0.706111 | 0.895735 | 0.960683 | 0.936763 | 0.673774 | 0.73396 |
| nent_12 | 0.972867 | 0.968828 | 0.929534 | 0.957146 | 0.979487 | 0.99309 |
| nent_13 | 0.164474 | 0.480202 | 0.620970 | 0.544693 | 0.113778 | 0.19823 |
| nent_14 | 0.352782 | 0.623963 | 0.751865 | 0.715493 | 0.330514 | 0.40750 |
| nent_15 | -0.542743 | -0.777953 | -0.875928 | -0.846726 | -0.525384 | -0.59556 |
| nent_16 | 0.793789 | 0.805280 | 0.804821 | 0.859130 | 0.824095 | 0.84033 |
| nent_17 | 0.873265 | 0.983065 | 0.998680 | 0.990213 | 0.845626 | 0.88713 |
| nent_18 | 0.572171 | 0.810312 | 0.895446 | 0.843647 | 0.523827 | 0.59404 |
| nent_19 | 0.164334 | 0.480073 | 0.620839 | 0.544563 | 0.113635 | 0.19809 |
| nent_20 | 0.977703 | 0.962425 | 0.917144 | 0.946478 | 0.985714 | 0.99635 |
| nent_21 | 0.977718 | 0.962416 | 0.917125 | 0.946469 | 0.985727 | 0.99636 |

umns

In [11]:

```
stats, top_pos, top_neg, corr = summarize_sensor_correlations(train__clean, method="pearson", top_k=5)

print("Correlation summary:", {k: round(v, 3) if isinstance(v, float) else v for k, v in stats.items()})

display(top_pos.round(3))
display(top_neg.round(3))
```

```
Correlation summary: {'mean_corr': 0.633, 'median_corr': 0.845, 'mean_abs_corr': 0.783, 'median_abs_corr': 0.874, 'n_pa
irs': 210}
```

| | sensor_a | sensor_b | corr | abs_corr |
|---|---|---|---|---|
| **0** | sensor_measurement_8 | sensor_measurement_18 | 1.000 | 1.000 |
| **1** | sensor_measurement_13 | sensor_measurement_19 | 1.000 | 1.000 |
| **2** | sensor_measurement_7 | sensor_measurement_12 | 1.000 | 1.000 |
| **3** | sensor_measurement_20 | sensor_measurement_21 | 1.000 | 1.000 |
| **4** | sensor_measurement_7 | sensor_measurement_20 | 0.999 | 0.999 |

| | sensor_a | sensor_b | corr | abs_corr |
|---|---|---|---|---|
| **0** | sensor_measurement_15 | sensor_measurement_18 | -0.971 | 0.971 |
| **1** | sensor_measurement_8 | sensor_measurement_15 | -0.971 | 0.971 |
| **2** | sensor_measurement_11 | sensor_measurement_15 | -0.965 | 0.965 |
| **3** | sensor_measurement_14 | sensor_measurement_15 | -0.957 | 0.957 |
| **4** | sensor_measurement_9 | sensor_measurement_15 | -0.885 | 0.885 |

In [ ]: