Exploratory Data Analysis For CAMPSS Dataset

Importing important Libraries.

```
# 3/85 -> Block number to track process during execution


# All Libraries are retrospectively pasted here for code clearance
# Standard Library
import warnings

# Data Manipulation & Visualization
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# Machine Learning & Preprocessing
import sklearn
from sklearn.ensemble import RandomForestRegressor  # selected as an alternative to XGBoost
from sklearn.model_selection import train_test_split, TimeSeriesSplit
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.metrics import (
    mean_absolute_error as MAE,
    mean_squared_error as MSE,
    mean_absolute_percentage_error as MAPE
)

# Statistical Tools
from scipy.stats import zscore, gaussian_kde

# Imbalanced Data Handling
from imblearn.under_sampling import RandomUnderSampler
from sklearn.model_selection import GroupShuffleSplit

# Gradient Boosting
import xgboost as xgb

# Jupyter-specific
%matplotlib inline

# Suppress Warnings
warnings.filterwarnings('ignore')

print("Library Load Successful")
```

```
Library Load Successful
```

Reading all the "train_FD00[i].txt" files as Dataframes.

```
# 5/85
train1=pd.read_csv('train_FD001.txt',sep="\s+",header=None)
train2=pd.read_csv('train_FD002.txt',sep="\s+",header=None)
train3=pd.read_csv('train_FD003.txt',sep="\s+",header=None)
train4=pd.read_csv('train_FD004.txt',sep="\s+",header=None)

test1 = pd.read_csv('test_FD001.txt', sep="\s+", header=None)
test2 = pd.read_csv('test_FD002.txt', sep="\s+", header=None)
test3 = pd.read_csv('test_FD003.txt', sep="\s+", header=None)
test4 = pd.read_csv('test_FD004.txt', sep="\s+", header=None)

rul1 = pd.read_csv('RUL_FD001.txt', header=None, names=['RUL'])
rul2 = pd.read_csv('RUL_FD002.txt', header=None, names=['RUL'])
rul3 = pd.read_csv('RUL_FD003.txt', header=None, names=['RUL'])
rul4 = pd.read_csv('RUL_FD004.txt', header=None, names=['RUL'])

print("Reading .txt to dataframes Successful")
```

```
Reading .txt to dataframes Successful
```

Getting the shapes of all the "train_FD00[i].txt" files.

```
# 7/85
print("Shape of 'train_FD001.txt' is: ",train1.shape)
print("Shape of 'train_FD002.txt' is: ",train2.shape)
print("Shape of 'train_FD003.txt' is: ",train3.shape)
print("Shape of 'train_FD004.txt' is: ",train4.shape)
print("-"*40)
print("Shape of 'test_FD001.txt' is: ",test1.shape)
print("Shape of 'test_FD002.txt' is: ",test2.shape)
print("Shape of 'test_FD003.txt' is: ",test3.shape)
print("Shape of 'test_FD004.txt' is: ",test4.shape)
print("-"*40)
print("Shape of 'RUL_FD001.txt' is: ",rul1.shape)
print("Shape of 'RUL_FD002.txt' is: ",rul2.shape)
print("Shape of 'RUL_FD003.txt' is: ",rul3.shape)
print("Shape of 'RUL_FD004.txt' is: ",rul4.shape)
print("="*40)
print("Shape printing Successful")
```

```
Shape of 'train_FD001.txt' is:  (20631, 26)
Shape of 'train_FD002.txt' is:  (53759, 26)
Shape of 'train_FD003.txt' is:  (24720, 26)
Shape of 'train_FD004.txt' is:  (61249, 26)
----------------------------------------
Shape of 'test_FD001.txt' is:  (13096, 26)
Shape of 'test_FD002.txt' is:  (33991, 26)
Shape of 'test_FD003.txt' is:  (16596, 26)
Shape of 'test_FD004.txt' is:  (41214, 26)
----------------------------------------
Shape of 'RUL_FD001.txt' is:  (100, 1)
Shape of 'RUL_FD002.txt' is:  (259, 1)
Shape of 'RUL_FD003.txt' is:  (100, 1)
Shape of 'RUL_FD004.txt' is:  (248, 1)
========================================
Shape printing Successful
```

Displaying first 5 rows of "train_FD00[i].txt"

```
# 9/85
#train1

print("First 5 rows of 'train_FD001.txt' are: ")
train1.head()
```

```
First 5 rows of 'train_FD001.txt' are:
```

	0	1	2	3	4	5	6	7	8	9	...	16	17	18	19	20	21	22	23	24	25
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044

5 rows × 26 columns

Providing the Feature names.

```
# 11/85
                                        # 1. Index Names:
sensor_names = ['unit',                 # "unit" : Engine No.
                'cycle',                # "cycle" : Time, In Cycles
                                        # 2. Setting Names:
                'opset1',               # Operation Setting 1
                'opset2',               # Operation Setting 2
                'opset3',               # Operation Setting 3
                                        # 3. Sensor Names:
                'Fan_Inlet_Temp',       # "sensor1" : Fan Inlet Temperature (◦R)
                'LPC_Outlet_Temp',      # "sensor2" : LPC Outlet Temperature (◦R)
                'HPC_Outlet_Temp',      # "sensor3" : HPC Outlet Temperature (◦R)
                'LPT_Outlet_Temp',      # "sensor4" : LPT Outlet Temperature (◦R)
                'Fan_Inlet_Press',      # "sensor5" : Fan Inlet Pressure (psia)
                'Bypass_Duct',          # "sensor6" : Bypass-Duct Pressure (psia)
                'HPC_Outlet_Press',     # "sensor7" : HPC Outlet Pressure (psia)
                'Physical_Fan_Speed',   # "sensor8" : Physical Fan Speed (rpm)
                'Physical_Core_Speed',  # "sensor9" : Physical Core Speed (rpm)
                'Engine_Press_Ratio',   # "sensor10" : Engine Pressure Ratio(P50/P2)
                'HPC_Outlet_Static_Press',  # "sensor11" : HPC Outlet Static Pressure (psia)
                'Fuel_Flow_Ratio',      # "sensor12" : Ratio of Fuel Flow to Ps30 (pps/psia)
                'Corrected_Fan_Speed',  # "sensor13" : Corrected Fan Speed (rpm)
                'Corrected_Core_Speed',  # "sensor14" : Corrected Core Speed (rpm)
                'Bypass_Ratio',         # "sensor15" : Bypass Ratio
                'Fuel_Air_Ratio',       # "sensor16" : Burner Fuel-Air Ratio
                'Bleed_Enthalpy',       # "sensor17" : Bleed Enthalpy
                'Req_Fan_Speed',        # "sensor18" : Required Fan Speed
                'Req_Core_Speed',       # "sensor19" : Required Fan Conversion Speed
                'HPT_Cool_Air_F',       # "sensor20" : High-Pressure Turbines Cool Air Flow
                'LPT_Cool_Air_F'        # "sensor21" : Low-Pressure Turbines Cool Air Flow
]

# Assigning the Feature names to the corresponding Dataframe.
datasets = [train1, train2, train3, train4, test1, test2, test3, test4]
for df in datasets:
    df.columns = sensor_names

print("Columns renamed. Sample train1 columns:", train1.columns.tolist()[:5])
print("train1 head:\n", train1.head(2))
```

```
Columns renamed. Sample train1 columns: ['unit', 'cycle', 'opset1', 'opset2', 'opset3']
train1 head:
    unit  cycle  opset1  opset2  opset3  Fan_Inlet_Temp  LPC_Outlet_Temp  \
0     1      1 -0.0007 -0.0004   100.0          518.67           641.82
1     1      2  0.0019 -0.0003   100.0          518.67           642.15

   HPC_Outlet_Temp  LPT_Outlet_Temp  Fan_Inlet_Press  ...  Fuel_Flow_Ratio  \
0          1589.70          1400.60           14.62  ...           521.66
1          1591.82          1403.14           14.62  ...           522.28

   Corrected_Fan_Speed  Corrected_Core_Speed  Bypass_Ratio  Fuel_Air_Ratio  \
0               2388.02               8138.62        8.4195            0.03
1               2388.07               8131.49        8.4318            0.03

   Bleed_Enthalpy  Req_Fan_Speed  Req_Core_Speed  HPT_Cool_Air_F  \
0             392           2388           100.0           39.06
1             392           2388           100.0           39.00

   LPT_Cool_Air_F
0         23.4190
1         23.4236

[2 rows x 26 columns]
```

Sensor Names and Explanations

Sensor	What It Measures (Simply)	Purpose / What It Indicates
Fan Inlet Temperature (°R)	Temperature of air entering the engine	Ambient air temperature affects engine efficiency; hot air = lower thrust.
LPC Outlet Temperature (°R)	Temperature of air exiting the low-pressure compressor	Indicates compressor heating and compression efficiency.
HPC Outlet Temperature (°R)	Temperature of air exiting the high-pressure compressor	Shows compressor load and potential overheating.
LPT Outlet Temperature (°R)	Temperature of gases exiting the low-pressure turbine	Reflects turbine energy output and cooling level.
Fan Inlet Pressure (psia)	Pressure of air entering the engine	Provides info on ambient pressure, altitude, and air density.
Bypass Duct Pressure (psia)	Pressure in the bypass duct (air not passing through combustion)	Indicates fan system performance and airflow.
HPC Outlet Pressure (psia)	Pressure of air exiting the high-pressure compressor	Monitors compressor performance and potential blockage/wear.
Physical Fan Speed (rpm)	Actual rotational speed of the fan	Directly related to thrust; drop indicates performance loss.
Physical Core Speed (rpm)	Rotational speed of the engine core (main turbine shaft)	Reflects overall engine power and load status.
Engine Pressure Ratio (P50/P2)	Ratio of engine outlet pressure to inlet pressure	Key indicator of overall engine performance; lower ratio = reduced efficiency.
HPC Outlet Static Pressure (psia)	Static pressure at compressor outlet	Shows internal flow dynamics; even small changes can be significant.
Ratio of Fuel Flow to HPC Outlet Pressure (pps/psia)	Fuel flow ratio	Provides insight into fuel consumption and power balance.
Corrected Fan Speed (rpm)	Fan speed corrected for ambient conditions	Enables comparison under varying temperature/pressure conditions.
Corrected Core Speed (rpm)	Core speed corrected for ambient conditions	Similarly normalized speed for comparative analysis.
Bypass Ratio	Ratio of bypass air (not passing through combustion)	Higher ratio = better efficiency in modern jets.
Burner Fuel-Air Ratio	Fuel-air mixture ratio in the combustion chamber	Excess fuel > temperature rise > risk of turbine damage.
Bleed Enthalpy	Energy of hot air extracted from the compressor	Indicates energy loss in internal air systems.
Required Fan Speed	Desired fan speed from engine control system	Target output of engine control algorithm.
Required Fan Conversion Speed	Converted control signal for fan speed	Difference between actual and target speed reflects performance.
High-Pressure Turbines Cool Air Flow	Cooling airflow for high-pressure turbine	Insufficient cooling leads to engine overheating.
Low-Pressure Turbines Cool Air Flow	Cooling airflow for low-pressure turbine	Similarly monitors cooling efficiency.

Setting	What It Means	Purpose / What It Indicates
Operation Setting 1	Typically related to altitude or air density.	Indicates the environmental conditions under which the engine operates.
Operation Setting 2	Corresponds to aircraft speed (Mach number) or airflow velocity.	As speed increases, **engine load and temperature change.**
Operation Setting 3	Throttle or power setting.	Shows how much power the engine is producing, **hence the load level.**

Information about all the train Dataframes.

```
# 15/85

print("=======train_FD001.txt' file: ======= \n")
train1.info()
print("\n==========================================\n")

print("=======train_FD002.txt' file: ======= \n")
train2.info()
print("\n==========================================\n")

print("=======train_FD003.txt' file: ======= \n")
train3.info()
print("\n==========================================\n")

print("=======train_FD004.txt' file: ======= \n")
train4.info()
print("\n==========================================\n")
```

```
=======train_FD001.txt' file: =======

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20631 entries, 0 to 20630
Data columns (total 26 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   unit                    20631 non-null  int64
 1   cycle                   20631 non-null  int64
 2   opset1                  20631 non-null  float64
 3   opset2                  20631 non-null  float64
 4   opset3                  20631 non-null  float64
 5   Fan_Inlet_Temp          20631 non-null  float64
 6   LPC_Outlet_Temp         20631 non-null  float64
 7   HPC_Outlet_Temp         20631 non-null  float64
 8   LPT_Outlet_Temp         20631 non-null  float64
 9   Fan_Inlet_Press         20631 non-null  float64
 10  Bypass_Duct             20631 non-null  float64
 11  HPC_Outlet_Press        20631 non-null  float64
 12  Physical_Fan_Speed      20631 non-null  float64
 13  Physical_Core_Speed     20631 non-null  float64
 14  Engine_Press_Ratio      20631 non-null  float64
 15  HPC_Outlet_Static_Press  20631 non-null  float64
 16  Fuel_Flow_Ratio         20631 non-null  float64
 17  Corrected_Fan_Speed     20631 non-null  float64
 18  Corrected_Core_Speed    20631 non-null  float64
 19  Bypass_Ratio            20631 non-null  float64
```

```
 20  Fuel_Air_Ratio           20631 non-null  float64
 21  Bleed_Enthalpy           20631 non-null  int64
 22  Req_Fan_Speed            20631 non-null  int64
 23  Req_Core_Speed           20631 non-null  float64
 24  HPT_Cool_Air_F           20631 non-null  float64
 25  LPT_Cool_Air_F           20631 non-null  float64
dtypes: float64(22), int64(4)
memory usage: 4.1 MB

=========================================

======='train_FD002.txt' file: =======

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53759 entries, 0 to 53758
Data columns (total 26 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   unit                     53759 non-null  int64
 1   cycle                    53759 non-null  int64
 2   opset1                   53759 non-null  float64
 3   opset2                   53759 non-null  float64
 4   opset3                   53759 non-null  float64
 5   Fan_Inlet_Temp           53759 non-null  float64
 6   LPC_Outlet_Temp          53759 non-null  float64
 7   HPC_Outlet_Temp          53759 non-null  float64
 8   LPT_Outlet_Temp          53759 non-null  float64
 9   Fan_Inlet_Press          53759 non-null  float64
 10  Bypass_Duct              53759 non-null  float64
 11  HPC_Outlet_Press         53759 non-null  float64
 12  Physical_Fan_Speed       53759 non-null  float64
```

Null Values of all the train Dataframes.

```
# 17/85

print(f"Null Values of 'train_FD001.txt' file are: {train1.isnull().sum().sum()}\n\n{train1.isnull().sum()}")
print(f"\nNull Values of 'train_FD002.txt' file are: {train2.isnull().sum().sum()}\n\n{train2.isnull().sum()}")
print(f"\nNull Values of 'train_FD003.txt' file are: {train3.isnull().sum().sum()}\n\n{train3.isnull().sum()}")
print(f"\nNull Values of 'train_FD004.txt' file are: {train4.isnull().sum().sum()}\n\n{train4.isnull().sum()}")
```

```
Null Values of 'train_FD001.txt' file are: 0

unit                        0
cycle                       0
opset1                      0
opset2                      0
opset3                      0
Fan_Inlet_Temp              0
LPC_Outlet_Temp             0
HPC_Outlet_Temp             0
LPT_Outlet_Temp             0
Fan_Inlet_Press             0
Bypass_Duct                 0
HPC_Outlet_Press            0
Physical_Fan_Speed          0
Physical_Core_Speed         0
Engine_Press_Ratio          0
HPC_Outlet_Static_Press     0
Fuel_Flow_Ratio             0
Corrected_Fan_Speed         0
Corrected_Core_Speed        0
Bypass_Ratio                0
Fuel_Air_Ratio              0
Bleed_Enthalpy              0
Req_Fan_Speed               0
Req_Core_Speed              0
HPT_Cool_Air_F              0
LPT_Cool_Air_F              0
dtype: int64

Null Values of 'train_FD002.txt' file are: 0

unit                        0
cycle                       0
opset1                      0
opset2                      0
opset3                      0
Fan_Inlet_Temp              0
LPC_Outlet_Temp             0
HPC_Outlet_Temp             0
LPT_Outlet_Temp             0
Fan_Inlet_Press             0
Bypass_Duct                 0
HPC_Outlet_Press            0
Physical_Fan_Speed          0
Physical_Core_Speed         0
Engine_Press_Ratio          0
HPC_Outlet_Static_Press     0
Fuel_Flow_Ratio             0
Corrected_Fan_Speed         0
Corrected_Core_Speed        0
Bypass_Ratio                0
Fuel_Air_Ratio              0
Bleed_Enthalpy              0
Req_Fan_Speed               0
Req_Core_Speed              0
HPT_Cool_Air_F              0
LPT_Cool_Air_F              0
```

Checking for the duplicated values in all the train Dataframes.

```
# 19/85

print(f"Duplicated values in 'train1' Dataframe are: {train1.duplicated().sum()}")
print(f"Duplicated values in 'train2' Dataframe are: {train1.duplicated().sum()}")
print(f"Duplicated values in 'train3' Dataframe are: {train1.duplicated().sum()}")
print(f"Duplicated values in 'train4' Dataframe are: {train1.duplicated().sum()}")
```

```
Duplicated values in 'train1' Dataframe are: 0
Duplicated values in 'train2' Dataframe are: 0
Duplicated values in 'train3' Dataframe are: 0
Duplicated values in 'train4' Dataframe are: 0
```

Describing the Dataframes.

```
# 21/85

#train 1
print("Describing the 'train1' Dataframe:")
train1.describe().T
```

```
Describing the 'train1' Dataframe:
```

	count	mean	std	min	25%	50%	75%	max
unit	20631.0	51.506568	2.922763e+01	1.0000	26.0000	52.0000	77.0000	100.0000
cycle	20631.0	108.807862	6.888099e+01	1.0000	52.0000	104.0000	156.0000	362.0000
opset1	20631.0	-0.000009	2.187313e-03	-0.0087	-0.0015	0.0000	0.0015	0.0087
opset2	20631.0	0.000002	2.930621e-04	-0.0006	-0.0002	0.0000	0.0003	0.0006
opset3	20631.0	100.000000	0.000000e+00	100.0000	100.0000	100.0000	100.0000	100.0000
Fan_Inlet_Temp	20631.0	518.670000	0.000000e+00	518.6700	518.6700	518.6700	518.6700	518.6700
LPC_Outlet_Temp	20631.0	642.680934	5.000533e-01	641.2100	642.3250	642.6400	643.0000	644.5300
HPC_Outlet_Temp	20631.0	1590.523119	6.131150e+00	1571.0400	1586.2600	1590.1000	1594.3800	1616.9100
LPT_Outlet_Temp	20631.0	1408.933782	9.000605e+00	1382.2500	1402.3600	1408.0400	1414.5550	1441.4900
Fan_Inlet_Press	20631.0	14.620000	5.329200e-15	14.6200	14.6200	14.6200	14.6200	14.6200
Bypass_Duct	20631.0	21.609803	1.388985e-03	21.6000	21.6100	21.6100	21.6100	21.6100
HPC_Outlet_Press	20631.0	553.367711	8.850923e-01	549.8500	552.8100	553.4400	554.0100	556.0600
Physical_Fan_Speed	20631.0	2388.096652	7.098548e-02	2387.9000	2388.0500	2388.0900	2388.1400	2388.5600
Physical_Core_Speed	20631.0	9065.242941	2.208288e+01	9021.7300	9053.1000	9060.6600	9069.4200	9244.5900
Engine_Press_Ratio	20631.0	1.300000	0.000000e+00	1.3000	1.3000	1.3000	1.3000	1.3000
HPC_Outlet_Static_Press	20631.0	47.541168	2.670874e-01	46.8500	47.3500	47.5100	47.7000	48.5300
Fuel_Flow_Ratio	20631.0	521.413470	7.375534e-01	518.6900	520.9600	521.4800	521.9500	523.3800
Corrected_Fan_Speed	20631.0	2388.096152	7.191892e-02	2387.8800	2388.0400	2388.0900	2388.1400	2388.5600
Corrected_Core_Speed	20631.0	8143.752722	1.907618e+01	8099.9400	8133.2450	8140.5400	8148.3100	8293.7200
Bypass_Ratio	20631.0	8.442146	3.750504e-02	8.3249	8.4149	8.4389	8.4656	8.5848
Fuel_Air_Ratio	20631.0	0.030000	3.469531e-18	0.0300	0.0300	0.0300	0.0300	0.0300
Bleed_Enthalpy	20631.0	393.210654	1.548763e+00	388.0000	392.0000	393.0000	394.0000	400.0000
Req_Fan_Speed	20631.0	2388.000000	0.000000e+00	2388.0000	2388.0000	2388.0000	2388.0000	2388.0000
Req_Core_Speed	20631.0	100.000000	0.000000e+00	100.0000	100.0000	100.0000	100.0000	100.0000
HPT_Cool_Air_F	20631.0	38.816271	1.807464e-01	38.1400	38.7000	38.8300	38.9500	39.4300
LPT_Cool_Air_F	20631.0	23.289705	1.082509e-01	22.8942	23.2218	23.2979	23.3668	23.6184

```
# 22/85

#train 2
print("Describing the 'train2' Dataframe:")
train2.describe().T
```

```
Describing the 'train2' Dataframe:
```

	count	mean	std	min	25%	50%	75%	max
unit	53759.0	131.082981	74.463862	1.0000	68.0000	131.0000	195.0000	260.0000
cycle	53759.0	109.154746	69.180569	1.0000	52.0000	104.0000	157.0000	378.0000
opset1	53759.0	23.998407	14.747376	0.0000	10.0046	25.0013	41.9980	42.0080
opset2	53759.0	0.572056	0.310016	0.0000	0.2507	0.7000	0.8400	0.8420
opset3	53759.0	94.046020	14.237735	60.0000	100.0000	100.0000	100.0000	100.0000
Fan_Inlet_Temp	53759.0	472.910207	26.389707	445.0000	445.0000	462.5400	491.1900	518.6700
LPC_Outlet_Temp	53759.0	579.672399	37.289399	535.5300	549.5700	555.9800	607.3400	644.5200
HPC_Outlet_Temp	53759.0	1419.971013	105.946341	1243.7300	1352.7600	1369.1800	1499.3700	1612.8800
LPT_Outlet_Temp	53759.0	1205.442024	119.123428	1023.7700	1123.6550	1138.8900	1306.8500	1439.2300
Fan_Inlet_Press	53759.0	8.031986	3.613839	3.9100	3.9100	7.0500	10.5200	14.6200
Bypass_Duct	53759.0	11.600746	5.431802	5.7100	5.7200	9.0300	15.4900	21.6100
HPC_Outlet_Press	53759.0	282.606787	146.005306	136.8000	139.9350	194.6600	394.0800	555.8200
Physical_Fan_Speed	53759.0	2228.879188	145.209816	1914.7700	2211.8800	2223.0700	2323.9600	2388.3900
Physical_Core_Speed	53759.0	8525.200837	335.812013	7985.5600	8321.6600	8361.2000	8778.0300	9215.6600
Engine_Press_Ratio	53759.0	1.094962	0.127469	0.9300	1.0200	1.0200	1.2600	1.3000
HPC_Outlet_Static_Press	53759.0	42.985172	3.232372	36.2300	41.9100	42.3900	45.3500	48.5100
Fuel_Flow_Ratio	53759.0	266.069034	137.659507	129.1200	131.5200	183.2000	371.2600	523.3700
Corrected_Fan_Speed	53759.0	2334.557253	128.068271	2027.6100	2387.9000	2388.0800	2388.1700	2390.4800
Corrected_Core_Speed	53759.0	8066.597682	84.837950	7848.3600	8062.1400	8082.5400	8127.1950	8268.5000
Bypass_Ratio	53759.0	9.329654	0.749335	8.3357	8.6778	9.3109	9.3869	11.0669
Fuel_Air_Ratio	53759.0	0.023326	0.004711	0.0200	0.0200	0.0200	0.0300	0.0300
Bleed_Enthalpy	53759.0	348.309511	27.754515	303.0000	331.0000	335.0000	369.0000	399.0000
Req_Fan_Speed	53759.0	2228.806358	145.327980	1915.0000	2212.0000	2223.0000	2324.0000	2388.0000
Req_Core_Speed	53759.0	97.756838	5.364067	84.9300	100.0000	100.0000	100.0000	100.0000
HPT_Cool_Air_F	53759.0	20.789296	9.869331	10.1800	10.9100	14.8800	28.4700	39.3400
LPT_Cool_Air_F	53759.0	12.473423	5.921615	6.0105	6.5463	8.9292	17.0832	23.5901

```
# 23/85

# train 3
print("Describing the 'train3' Dataframe:")
train3.describe().T
```

```
Describing the 'train3' Dataframe:
```

	count	mean	std	min	25%	50%	75%	max
unit	24720.0	48.631877	2.934899e+01	1.0000	23.0000	47.0000	74.0000	100.0000
cycle	24720.0	139.077063	9.884668e+01	1.0000	62.0000	124.0000	191.0000	525.0000
opset1	24720.0	-0.000024	2.193545e-03	-0.0086	-0.0015	-0.0000	0.0015	0.0086
opset2	24720.0	0.000005	2.940429e-04	-0.0006	-0.0002	-0.0000	0.0003	0.0007
opset3	24720.0	100.000000	0.000000e+00	100.0000	100.0000	100.0000	100.0000	100.0000
Fan_Inlet_Temp	24720.0	518.670000	0.000000e+00	518.6700	518.6700	518.6700	518.6700	518.6700
LPC_Outlet_Temp	24720.0	642.457858	5.230311e-01	640.8400	642.0800	642.4000	642.7900	645.1100
HPC_Outlet_Temp	24720.0	1588.079175	6.810418e+00	1564.3000	1583.2800	1587.5200	1592.4125	1615.3900
LPT_Outlet_Temp	24720.0	1404.471212	9.773178e+00	1377.0600	1397.1875	1402.9100	1410.6000	1441.1600
Fan_Inlet_Press	24720.0	14.620000	3.552786e-15	14.6200	14.6200	14.6200	14.6200	14.6200
Bypass_Duct	24720.0	21.595841	1.811600e-02	21.4500	21.5800	21.6000	21.6100	21.6100
HPC_Outlet_Press	24720.0	555.143808	3.437343e+00	549.6100	553.1100	554.0500	556.0400	570.4900
Physical_Fan_Speed	24720.0	2388.071555	1.582849e-01	2386.9000	2388.0000	2388.0700	2388.1400	2388.6000
Physical_Core_Speed	24720.0	9064.110809	1.998029e+01	9017.9800	9051.9200	9060.0100	9070.0925	9234.3500
Engine_Press_Ratio	24720.0	1.301232	3.484849e-03	1.2900	1.3000	1.3000	1.3000	1.3200
HPC_Outlet_Static_Press	24720.0	47.415707	3.000742e-01	46.6900	47.1900	47.3600	47.6000	48.4400
Fuel_Flow_Ratio	24720.0	523.050873	3.255314e+00	517.7700	521.1500	521.9800	523.8400	537.4000
Corrected_Fan_Speed	24720.0	2388.071643	1.581207e-01	2386.9300	2388.0100	2388.0700	2388.1400	2388.6100
Corrected_Core_Speed	24720.0	8144.202916	1.650412e+01	8099.6800	8134.5100	8141.2000	8149.2300	8290.5500
Bypass_Ratio	24720.0	8.396176	6.051161e-02	8.1563	8.3606	8.3983	8.4370	8.5705
Fuel_Air_Ratio	24720.0	0.030000	6.939034e-18	0.0300	0.0300	0.0300	0.0300	0.0300
Bleed_Enthalpy	24720.0	392.566545	1.761459e+00	388.0000	391.0000	392.0000	394.0000	399.0000
Req_Fan_Speed	24720.0	2388.000000	0.000000e+00	2388.0000	2388.0000	2388.0000	2388.0000	2388.0000
Req_Core_Speed	24720.0	100.000000	0.000000e+00	100.0000	100.0000	100.0000	100.0000	100.0000
HPT_Cool_Air_F	24720.0	38.988552	2.488647e-01	38.1700	38.8300	38.9900	39.1400	39.8500
LPT_Cool_Air_F	24720.0	23.393024	1.492338e-01	22.8726	23.2962	23.3916	23.4833	23.9505

```
# 24/85

#train 4
print("Describing the 'train4' Dataframe:")
train4.describe().T
```

Describing the 'train4' Dataframe:

	count	mean	std	min	25%	50%	75%	max
unit	61249.0	124.325181	71.995350	1.0000	60.0000	126.0000	185.0000	249.0000
cycle	61249.0	134.311417	89.783389	1.0000	62.0000	123.0000	191.0000	543.0000
opset1	61249.0	23.999823	14.780722	0.0000	10.0046	25.0014	41.9981	42.0080
opset2	61249.0	0.571347	0.310703	0.0000	0.2507	0.7000	0.8400	0.8420
opset3	61249.0	94.031576	14.251954	60.0000	100.0000	100.0000	100.0000	100.0000
Fan_Inlet_Temp	61249.0	472.882435	26.436832	445.0000	445.0000	462.5400	491.1900	518.6700
LPC_Outlet_Temp	61249.0	579.420056	37.342647	535.4800	549.3300	555.7400	607.0700	644.4200
HPC_Outlet_Temp	61249.0	1417.896600	106.167598	1242.6700	1350.5500	1367.6800	1497.4200	1613.0000
LPT_Outlet_Temp	61249.0	1201.915359	119.327591	1024.4200	1119.4900	1136.9200	1302.6200	1440.7700
Fan_Inlet_Press	61249.0	8.031626	3.622872	3.9100	3.9100	7.0500	10.5200	14.6200
Bypass_Duct	61249.0	11.589457	5.444017	5.6700	5.7200	9.0300	15.4800	21.6100
HPC_Outlet_Press	61249.0	283.328633	146.880210	136.1700	142.9200	194.9600	394.2800	570.8100
Physical_Fan_Speed	61249.0	2228.686034	145.348243	1914.7200	2211.9500	2223.0700	2323.9300	2388.6400
Physical_Core_Speed	61249.0	8524.673301	336.927547	7984.5100	8320.5900	8362.7600	8777.2500	9196.8100
Engine_Press_Ratio	61249.0	1.096445	0.127681	0.9300	1.0200	1.0300	1.2600	1.3200
HPC_Outlet_Static_Press	61249.0	42.874529	3.243492	36.0400	41.7600	42.3300	45.2200	48.3600
Fuel_Flow_Ratio	61249.0	266.735665	138.479109	128.3100	134.5200	183.4500	371.4000	537.4900
Corrected_Fan_Speed	61249.0	2334.427590	128.197859	2027.5700	2387.9100	2388.0600	2388.1700	2390.4900
Corrected_Core_Speed	61249.0	8067.811812	85.670543	7845.7800	8062.6300	8083.8100	8128.3500	8261.6500
Bypass_Ratio	61249.0	9.285604	0.750374	8.1757	8.6480	9.2556	9.3658	11.0663
Fuel_Air_Ratio	61249.0	0.023252	0.004685	0.0200	0.0200	0.0200	0.0300	0.0300
Bleed_Enthalpy	61249.0	347.760029	27.808283	302.0000	330.0000	334.0000	368.0000	399.0000
Req_Fan_Speed	61249.0	2228.613283	145.472491	1915.0000	2212.0000	2223.0000	2324.0000	2388.0000
Req_Core_Speed	61249.0	97.751396	5.369424	84.9300	100.0000	100.0000	100.0000	100.0000
HPT_Cool_Air_F	61249.0	20.864333	9.936396	10.1600	10.9400	14.9300	28.5600	39.8900
LPT_Cool_Air_F	61249.0	12.518995	5.962697	6.0843	6.5661	8.9601	17.1355	23.8852

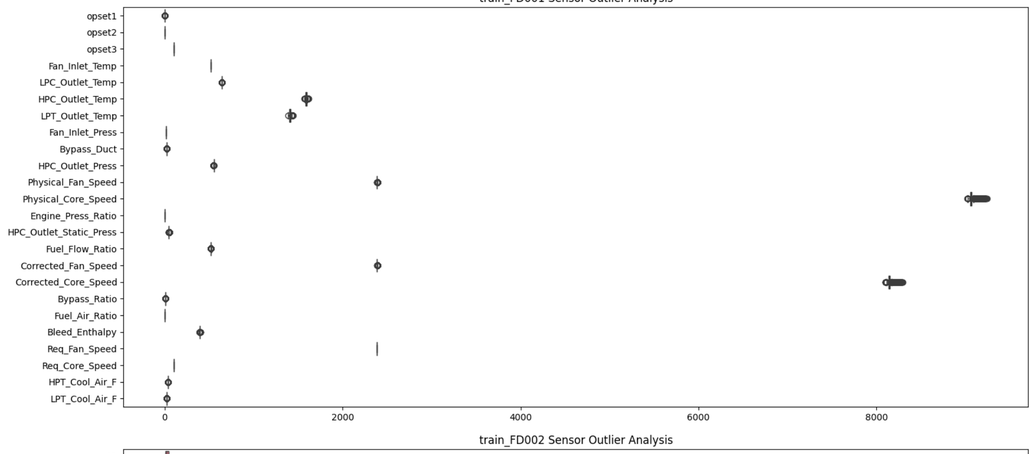
Unique categories of each feature in train Dataframes.

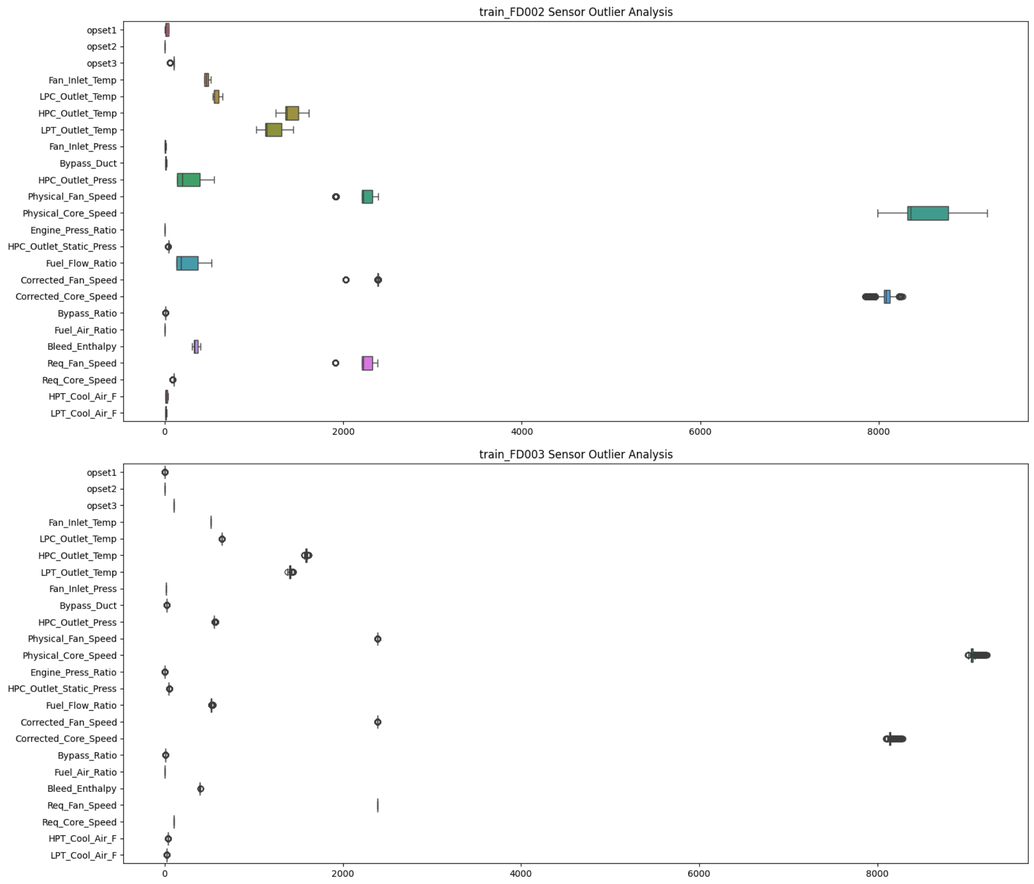
```
# 26/85

print(f"Unique categories of each feature in train1 Dataframe: {len(train1.columns)}\n\n{train1.nunique()}")
print("=================================================================\n")

print(f"Unique categories of each feature in train2 Dataframe: {len(train2.columns)}\n\n{train2.nunique()}")
print("=================================================================\n")

print(f"Unique categories of each feature in train3 Dataframe: {len(train3.columns)}\n\n{train3.nunique()}")
print("=================================================================\n")

print(f"Unique categories of each feature in train4 Dataframe: {len(train4.columns)}\n\n{train4.nunique()}")
```

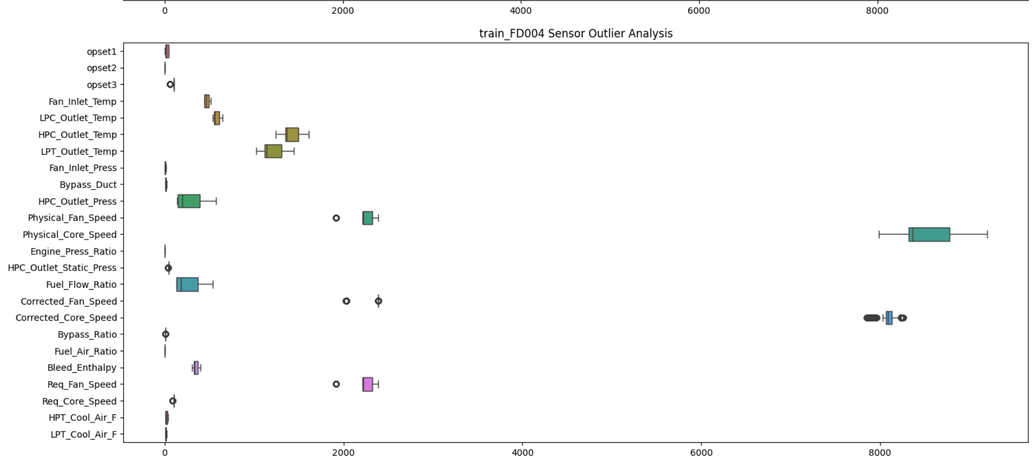
```
Unique categories of each feature in train1 Dataframe: 26

unit                       100
cycle                      362
opset1                     158
opset2                      13
opset3                       1
Fan_Inlet_Temp               1
LPC_Outlet_Temp            310
HPC_Outlet_Temp           3012
LPT_Outlet_Temp           4051
Fan_Inlet_Press              1
Bypass_Duct                  2
HPC_Outlet_Press           513
Physical_Fan_Speed          53
Physical_Core_Speed       6403
Engine_Press_Ratio           1
HPC_Outlet_Static_Press    159
Fuel_Flow_Ratio            427
Corrected_Fan_Speed         56
Corrected_Core_Speed      6078
Bypass_Ratio              1918
Fuel_Air_Ratio               1
Bleed_Enthalpy              13
Req_Fan_Speed                1
Req_Core_Speed               1
HPT_Cool_Air_F             120
LPT_Cool_Air_F            4745
dtype: int64
=================================================================

Unique categories of each feature in train2 Dataframe: 26

unit                       260
cycle                      378
opset1                     536
opset2                     105
opset3                       2
Fan_Inlet_Temp               6
LPC_Outlet_Temp           1590
HPC_Outlet_Temp          12305
LPT_Outlet_Temp          15411
Fan_Inlet_Press              6
Bypass_Duct                 14
HPC_Outlet_Press          2067
Physical_Fan_Speed         897
Physical_Core_Speed      22434
Engine_Press_Ratio           9
HPC_Outlet_Static_Press    681
Fuel_Flow_Ratio           1672
Corrected_Fan_Speed        514
Corrected_Core_Speed     14905
Bypass_Ratio              8464
Fuel_Air_Ratio               2
Bleed_Enthalpy              53
Req_Fan_Speed                6
Req_Core_Speed               2
HPT_Cool_Air_F             510
```

Outlier Analysis

```
# 28/85

# Boxplot outlier analysis for all sensors
def plot_outliers(df, title):
    plt.figure(figsize=(18, 8))
    sns.boxplot(data=df, orient="h")
    plt.title(title)
    plt.show()

plot_outliers(train1.drop(["unit", "cycle"], axis=1), "train_FD001 Sensor Outlier Analysis")
plot_outliers(train2.drop(["unit", "cycle"], axis=1), "train_FD002 Sensor Outlier Analysis")
plot_outliers(train3.drop(["unit", "cycle"], axis=1), "train_FD003 Sensor Outlier Analysis")
plot_outliers(train4.drop(["unit", "cycle"], axis=1), "train_FD004 Sensor Outlier Analysis")
```







Z-Score

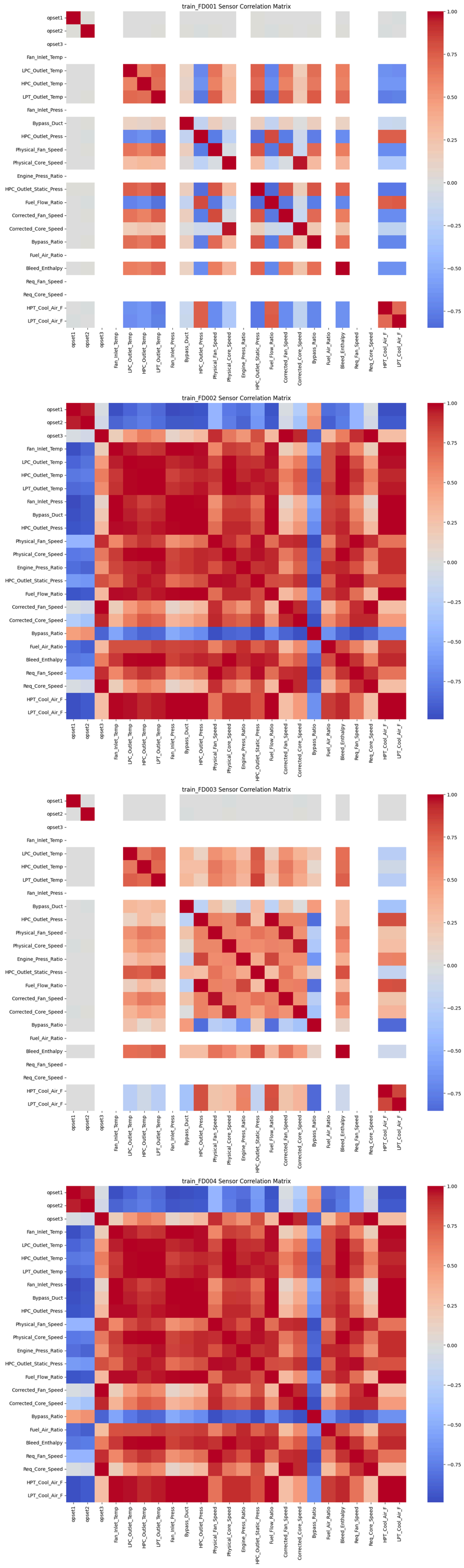
```
# 30/85

# Z-score > 3 or < -3 are considered outliers"
def zscore_outlier_summary(df, name):
    df_numeric = df.drop(["unit", "cycle"], axis=1)
    z_scores = np.abs(zscore(df_numeric))
```

```
    outlier_counts = (z_scores > 3).sum(axis=0)
    print(f"\nFor {name} outliers by sensors (z-score > 3):\n")
    print(outlier_counts)
    print("\nTotal observed outlier number:", (z_scores > 3).sum().sum())
    print("-"*60)

zscore_outlier_summary(train1, "train_FD001")
zscore_outlier_summary(train2, "train_FD002")
zscore_outlier_summary(train3, "train_FD003")
zscore_outlier_summary(train4, "train_FD004")
```

```
For train_FD001 outliers by sensors (z-score > 3):

[ 53   0   0   0  31  47  36   0 406  34 107 521   0  23  34  83 496  33
   0  81   0   0  43  34]

Total observed outlier number: 2062
------------------------------------------------------------

For train_FD002 outliers by sensors (z-score > 3):

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Total observed outlier number: 0
------------------------------------------------------------

For train_FD003 outliers by sensors (z-score > 3):

[ 65   0   0   0  81  57  50   0 182 655 579 367 166  31 691 570 334  43
   0  90   0   0  58  47]

Total observed outlier number: 4066
------------------------------------------------------------

For train_FD004 outliers by sensors (z-score > 3):

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Total observed outlier number: 0
------------------------------------------------------------
```

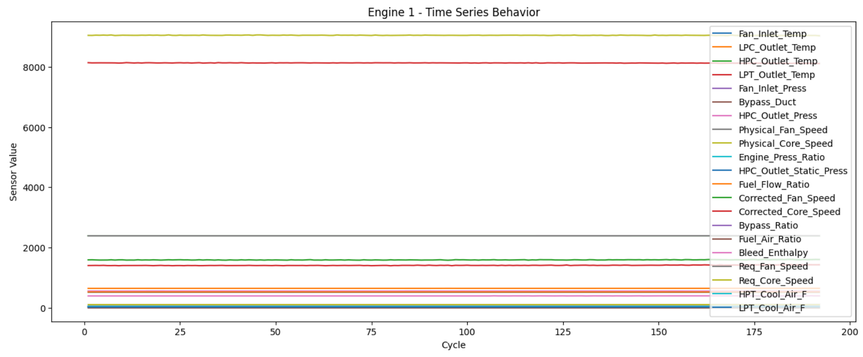
Correlation Matrix and Heat Map

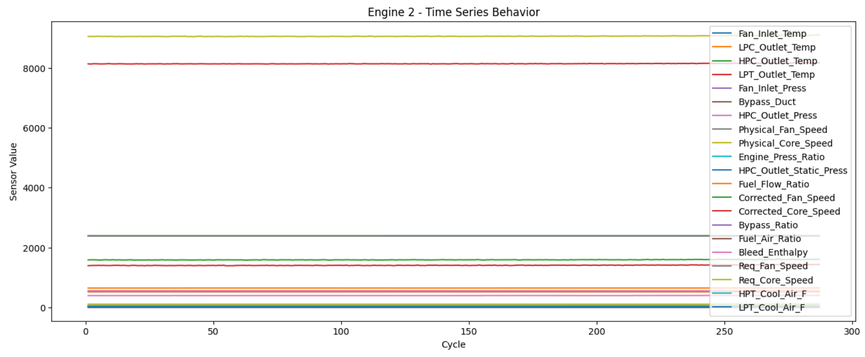
```
# 32/85

def plot_corr_heatmap(df, title):
    df_numeric = df.drop(["unit", "cycle"], axis=1)
    corr = df_numeric.corr()
    plt.figure(figsize=(16, 12))
    sns.heatmap(corr, annot=False, cmap="coolwarm", center=0)
    plt.title(title)
    plt.show()

plot_corr_heatmap(train1, "train_FD001 Sensor Correlation Matrix")
plot_corr_heatmap(train2, "train_FD002 Sensor Correlation Matrix")
plot_corr_heatmap(train3, "train_FD003 Sensor Correlation Matrix")
plot_corr_heatmap(train4, "train_FD004 Sensor Correlation Matrix")
```

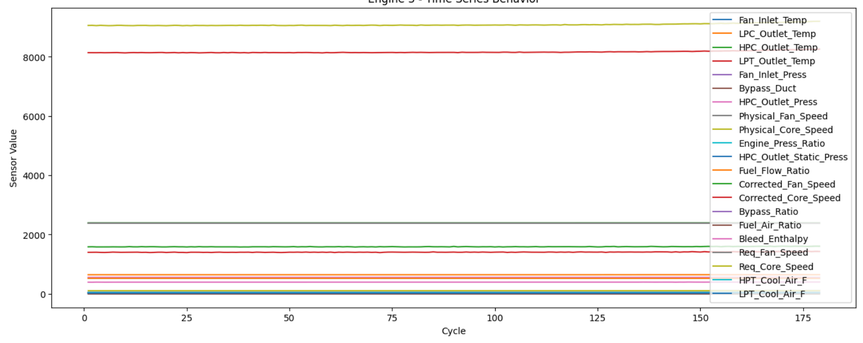


Time Series Behavior Analysis

```
# 34/85

#train 1

sensors = [     'Fan_Inlet_Temp',
                'LPC_Outlet_Temp',
                'HPC_Outlet_Temp',
                'LPT_Outlet_Temp',
                'Fan_Inlet_Press',  # 5-9 (s1-s5)
                'Bypass_Duct',
                'HPC_Outlet_Press',
                'Physical_Fan_Speed',
                'Physical_Core_Speed',
                'Engine_Press_Ratio',  # 10-14 (s6-s10)
                'HPC_Outlet_Static_Press',
                'Fuel_Flow_Ratio',
                'Corrected_Fan_Speed',
                'Corrected_Core_Speed',
                'Bypass_Ratio',  # 15-19 (s11-s15)
                'Fuel_Air_Ratio',
                'Bleed_Enthalpy',
                'Req_Fan_Speed',
                'Req_Core_Speed',
                'HPT_Cool_Air_F',
                'LPT_Cool_Air_F'  # 20-25 (s16-s21, approx)
            ]

for engine_id in train1["unit"].unique()[:3]:
    df_engine = train1[train1["unit"] == engine_id]
    plt.figure(figsize=(16, 6))
    for sensor in sensors:
        plt.plot(df_engine["cycle"], df_engine[sensor], label=sensor)

    plt.title(f"Engine {engine_id} - Time Series Behavior")
    plt.xlabel("Cycle")
    plt.ylabel("Sensor Value")
    plt.legend()
    plt.show()
```



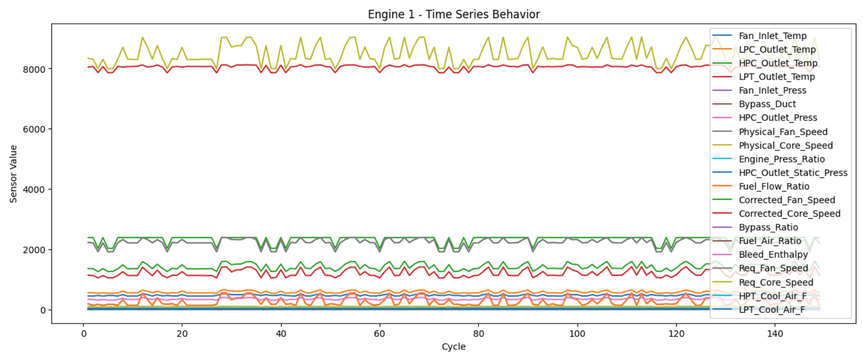


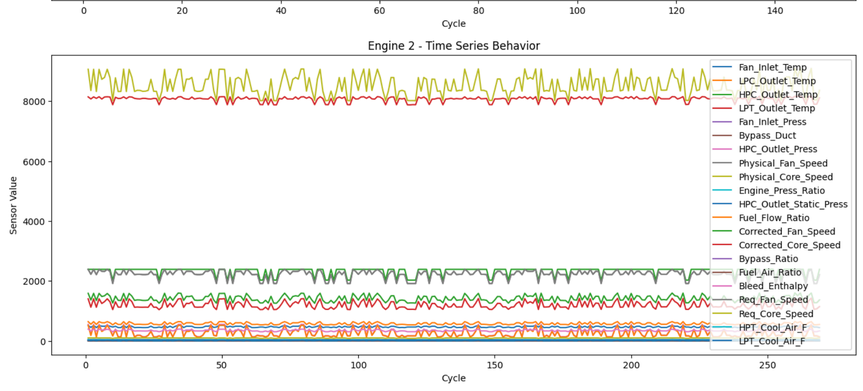


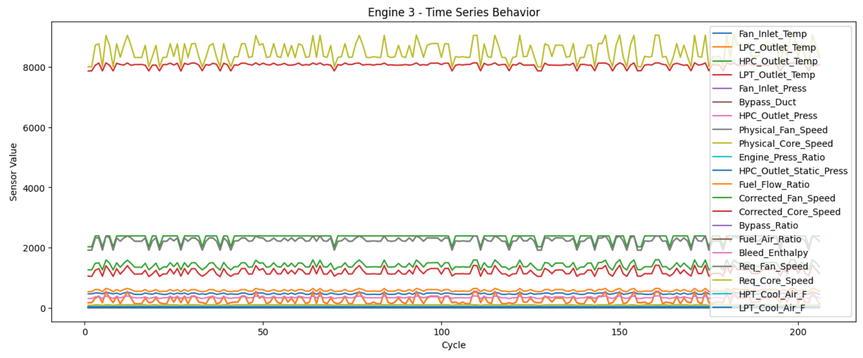
35/85

#train 2

```
for engine_id in train2["unit"].unique()[:3]:
    df_engine = train2[train2["unit"] == engine_id]
    plt.figure(figsize=(16, 6))
    for sensor in sensors:
        plt.plot(df_engine["cycle"], df_engine[sensor], label=sensor)

    plt.title(f"Engine {engine_id} - Time Series Behavior")
    plt.xlabel("Cycle")
    plt.ylabel("Sensor Value")
    plt.legend()
    plt.show()
```



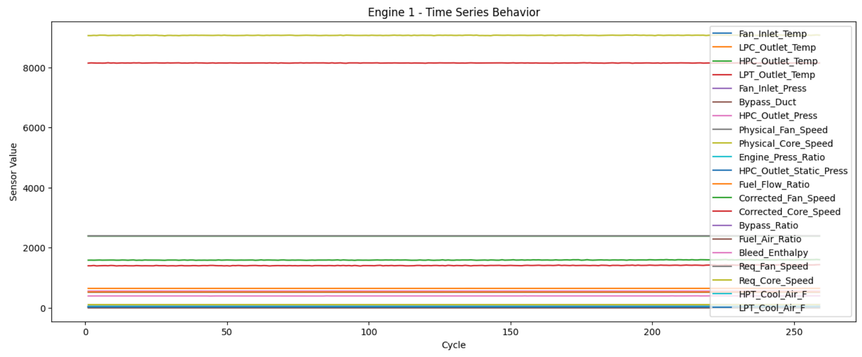


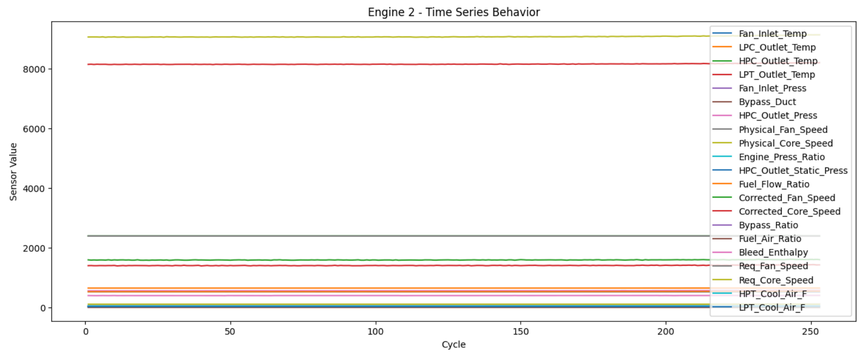


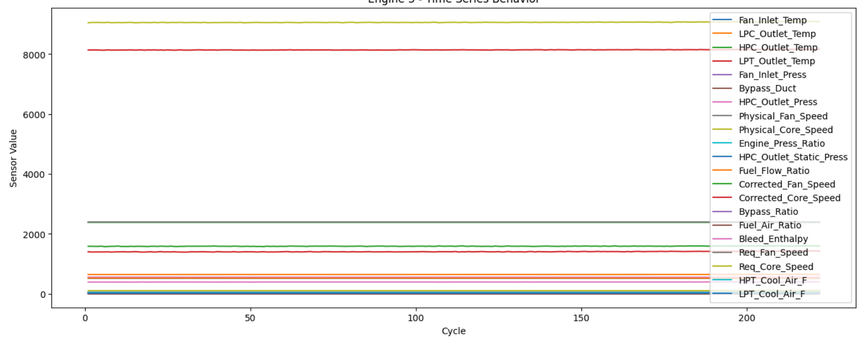
36/85

#train 3

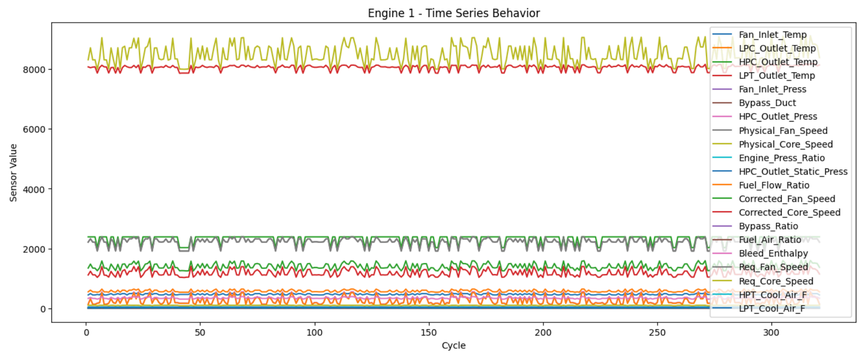
```
for engine_id in train3["unit"].unique()[:3]:
    df_engine = train3[train3["unit"] == engine_id]
    plt.figure(figsize=(16, 6))
    for sensor in sensors:
        plt.plot(df_engine["cycle"], df_engine[sensor], label=sensor)

    plt.title(f"Engine {engine_id} - Time Series Behavior")
    plt.xlabel("Cycle")
    plt.ylabel("Sensor Value")
    plt.legend()
    plt.show()
```

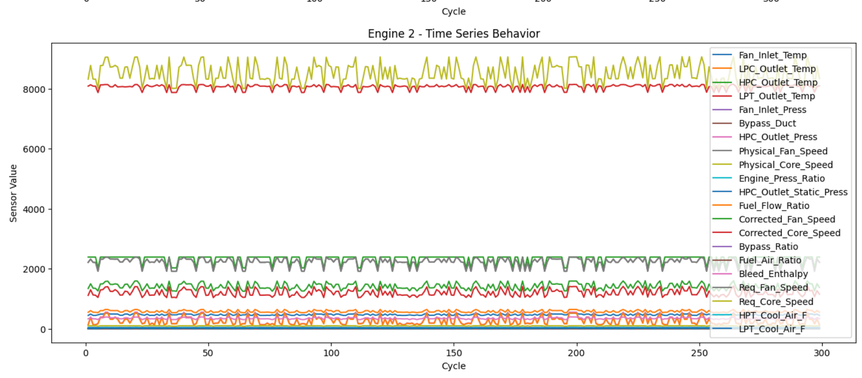


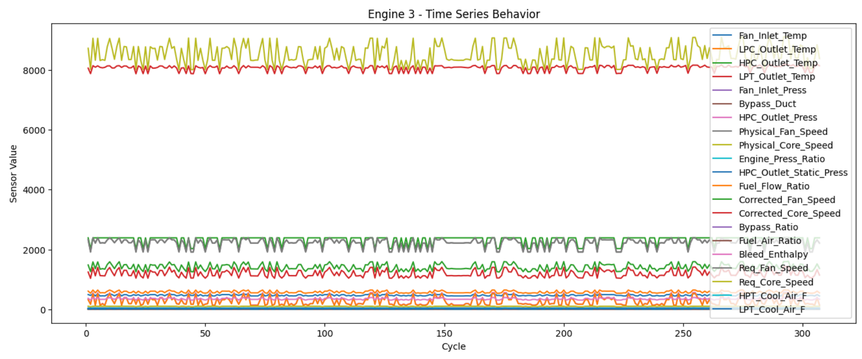





```
# 37/85
#train 4

for engine_id in train4["unit"].unique()[:3]:
    df_engine = train4[train4["unit"] == engine_id]
    plt.figure(figsize=(16, 6))
    for sensor in sensors:
        plt.plot(df_engine["cycle"], df_engine[sensor], label=sensor)

    plt.title(f"Engine {engine_id} - Time Series Behavior")
    plt.xlabel("Cycle")
    plt.ylabel("Sensor Value")
    plt.legend()
    plt.show()
```

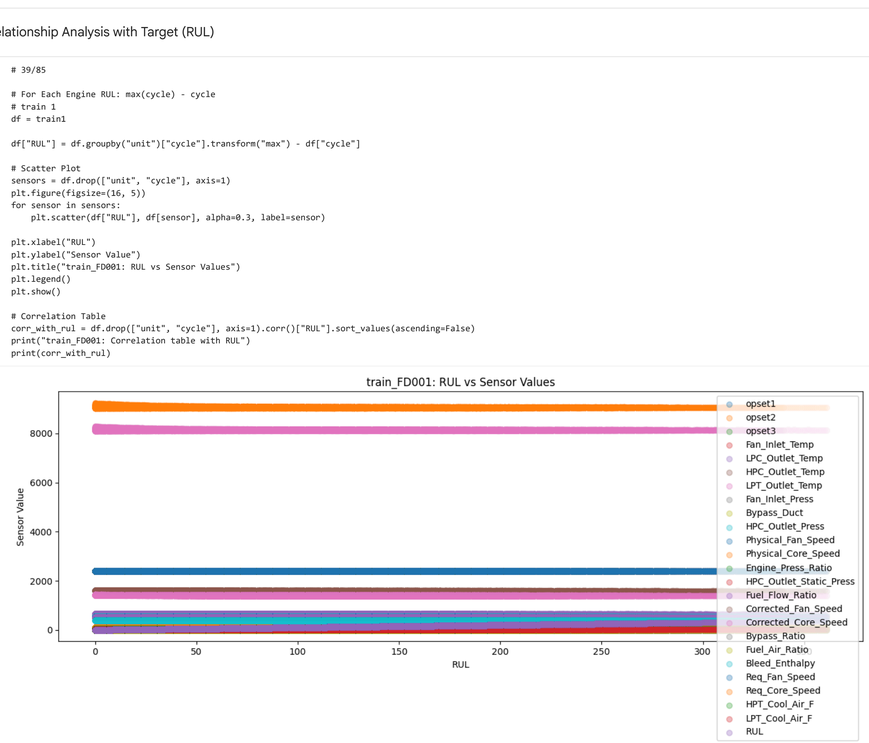






Relationship Analysis with Target (RUL)

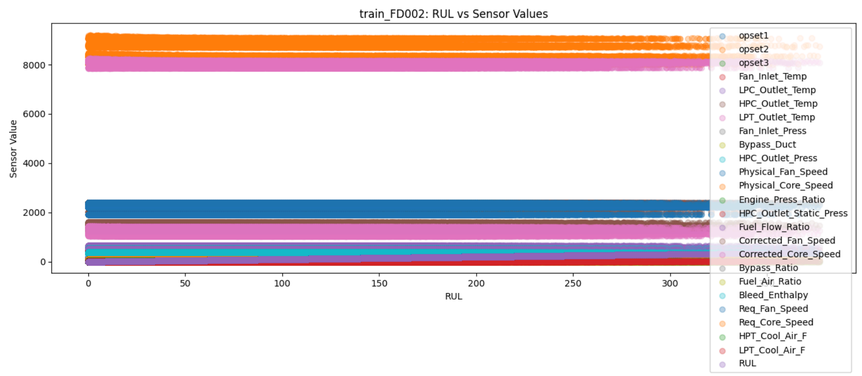
```
# 39/85

# For Each Engine RUL: max(cycle) - cycle
# train 1
df = train1

df["RUL"] = df.groupby("unit")["cycle"].transform("max") - df["cycle"]

# Scatter Plot
sensors = df.drop(["unit", "cycle"], axis=1)
plt.figure(figsize=(16, 5))
for sensor in sensors:
    plt.scatter(df["RUL"], df[sensor], alpha=0.3, label=sensor)

plt.xlabel("RUL")
plt.ylabel("Sensor Value")
plt.title("train_FD001: RUL vs Sensor Values")
plt.legend()
plt.show()

# Correlation Table
corr_with_rul = df.drop(["unit", "cycle"], axis=1).corr()["RUL"].sort_values(ascending=False)
print("train_FD001: Correlation table with RUL")
print(corr_with_rul)
```




```
train_FD001: Correlation table with RUL
RUL                       1.000000
Fuel_Flow_Ratio           0.671983
HPC_Outlet_Press          0.657223
LPT_Cool_Air_F            0.635662
HPT_Cool_Air_F            0.629428
opset2                   -0.001948
opset1                   -0.003198
Bypass_Duct              -0.128348
Corrected_Core_Speed     -0.306769
Physical_Core_Speed      -0.390102
Corrected_Fan_Speed      -0.562569
Physical_Fan_Speed       -0.563968
HPC_Outlet_Temp          -0.584520
Bleed_Enthalpy           -0.606154
LPC_Outlet_Temp          -0.606484
Bypass_Ratio             -0.642667
LPT_Outlet_Temp          -0.678948
HPC_Outlet_Static_Press  -0.696228
opset3                         NaN
Fan_Inlet_Temp                 NaN
Fan_Inlet_Press                NaN
Engine_Press_Ratio             NaN
Fuel_Air_Ratio                 NaN
Req_Fan_Speed                  NaN
Req_Core_Speed                 NaN
Name: RUL, dtype: float64
```

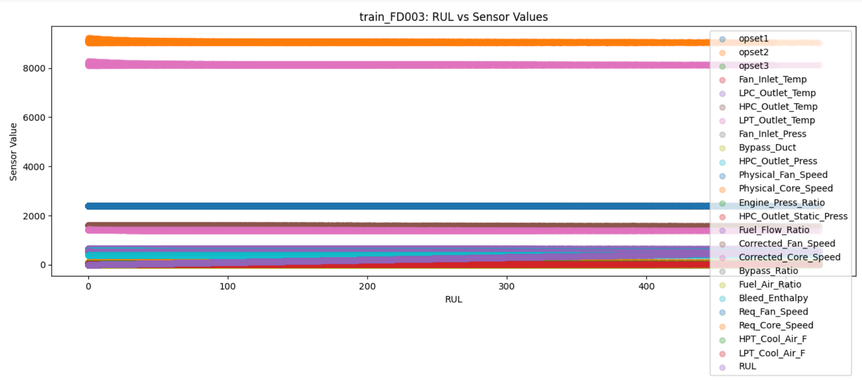
```
# 40/85

# train 2

df = train2

df["RUL"] = df.groupby("unit")["cycle"].transform("max") - df["cycle"]

# Scatter Plot
sensors = df.drop(["unit", "cycle"], axis=1)
plt.figure(figsize=(16, 5))
for sensor in sensors:
    plt.scatter(df["RUL"], df[sensor], alpha=0.3, label=sensor)
```

```
plt.xlabel("RUL")
plt.ylabel("Sensor Value")
plt.title("train_FD002: RUL vs Sensor Values")
plt.legend()
plt.show()

# Correlation Table
corr_with_rul = df.drop(["unit", "cycle"], axis=1).corr()["RUL"].sort_values(ascending=False)
print("train_FD002: Correlation table with RUL")
print(corr_with_rul)
```



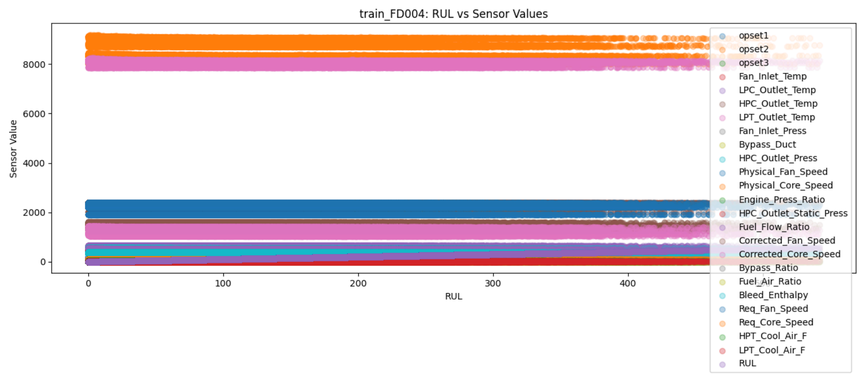

```
train_FD002: Correlation table with RUL
RUL                        1.000000
HPT_Cool_Air_F             0.006287
LPT_Cool_Air_F             0.006165
Req_Core_Speed             0.005761
opset3                     0.005761
Corrected_Fan_Speed        0.005245
Req_Fan_Speed              0.004780
Physical_Fan_Speed         0.004345
Engine_Press_Ratio         0.004306
HPC_Outlet_Press           0.002430
Fuel_Flow_Ratio            0.002249
opset1                     0.001135
opset2                     0.000047
Fan_Inlet_Temp            -0.000023
Bypass_Duct               -0.000496
Fan_Inlet_Press           -0.000758
LPC_Outlet_Temp           -0.004928
Physical_Core_Speed       -0.015306
HPC_Outlet_Temp           -0.026942
Bleed_Enthalpy            -0.027031
Bypass_Ratio              -0.038455
LPT_Outlet_Temp           -0.040978
Corrected_Core_Speed      -0.042325
HPC_Outlet_Static_Press   -0.046950
Fuel_Air_Ratio            -0.071352
Name: RUL, dtype: float64
```

```
# 41/85

# train 3

df = train3

df["RUL"] = df.groupby("unit")["cycle"].transform("max") - df["cycle"]

# Scatter Plot
sensors = df.drop(["unit", "cycle"], axis=1)
plt.figure(figsize=(16, 5))
for sensor in sensors:
    plt.scatter(df["RUL"], df[sensor], alpha=0.3, label=sensor)

plt.xlabel("RUL")
plt.ylabel("Sensor Value")
plt.title("train_FD003: RUL vs Sensor Values")
plt.legend()
plt.show()

# Correlation Table
corr_with_rul = df.drop(["unit", "cycle"], axis=1).corr()["RUL"].sort_values(ascending=False)
print("train_FD003: Correlation table with RUL")
print(corr_with_rul)
```




```
train_FD003: Correlation table with RUL
RUL                        1.000000
HPT_Cool_Air_F             0.037782
LPT_Cool_Air_F             0.033465
opset1                     0.003700
opset2                    -0.001204
Bypass_Ratio              -0.016501
Bypass_Duct               -0.215352
HPC_Outlet_Press          -0.315048
Fuel_Flow_Ratio           -0.329452
Engine_Press_Ratio        -0.363432
Corrected_Core_Speed      -0.481807
Physical_Core_Speed       -0.551843
LPC_Outlet_Temp           -0.590564
HPC_Outlet_Temp           -0.624236
Bleed_Enthalpy            -0.649220
Physical_Fan_Speed        -0.655345
Corrected_Fan_Speed       -0.656340
LPT_Outlet_Temp           -0.657224
HPC_Outlet_Static_Press   -0.688915
opset3                          NaN
Fan_Inlet_Temp                  NaN
Fan_Inlet_Press                 NaN
Fuel_Air_Ratio                  NaN
Req_Fan_Speed                   NaN
Req_Core_Speed                  NaN
Name: RUL, dtype: float64
```

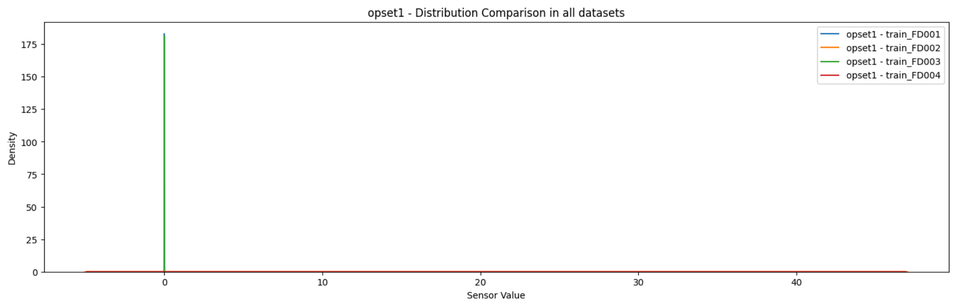
```
# 42/85

# train 4

df = train4

df["RUL"] = df.groupby("unit")["cycle"].transform("max") - df["cycle"]

# Scatter Plot
sensors = df.drop(["unit", "cycle"], axis=1)
plt.figure(figsize=(16, 5))
for sensor in sensors:
    plt.scatter(df["RUL"], df[sensor], alpha=0.3, label=sensor)

plt.xlabel("RUL")
plt.ylabel("Sensor Value")
plt.title("train_FD004: RUL vs Sensor Values")
plt.legend()
plt.show()

# Correlation Table
corr_with_rul = df.drop(["unit", "cycle"], axis=1).corr()["RUL"].sort_values(ascending=False)
print("train_FD004: Correlation table with RUL")
print(corr_with_rul)
```

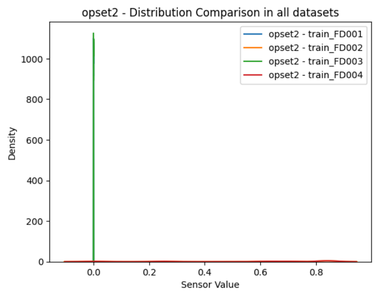


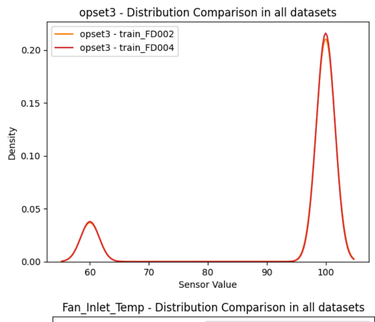

```
train_FD004: Correlation table with RUL
RUL                        1.000000
HPT_Cool_Air_F             0.002812
LPT_Cool_Air_F             0.002791
Req_Fan_Speed              0.002765
opset3                     0.002303
Req_Core_Speed             0.002303
Physical_Fan_Speed         0.002086
Fan_Inlet_Temp             0.001889
Fan_Inlet_Press            0.001679
Corrected_Fan_Speed        0.001501
Bypass_Duct                0.001349
HPC_Outlet_Press          -0.001429
Fuel_Flow_Ratio           -0.001639
opset2                    -0.002280
opset1                    -0.002380
Bypass_Ratio              -0.003957
LPC_Outlet_Temp           -0.004443
Engine_Press_Ratio        -0.008924
Physical_Core_Speed       -0.024727
HPC_Outlet_Temp           -0.032924
Bleed_Enthalpy            -0.032939
LPT_Outlet_Temp           -0.045881
Fuel_Air_Ratio            -0.053804
HPC_Outlet_Static_Press   -0.056639
Corrected_Core_Speed      -0.078126
Name: RUL, dtype: float64
```

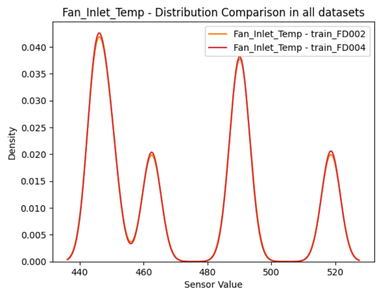
Comparison of All Sensors Across All Datasets

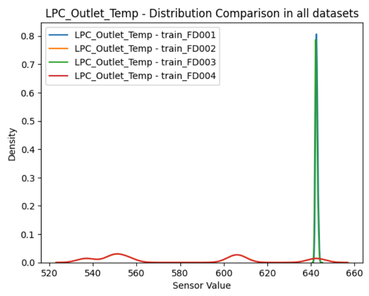
```
# 44/85

sensors = df.drop(["unit", "cycle"], axis=1)
plt.figure(figsize=(18, 5))

for sensor in sensors:
    for i, df in enumerate([train1, train2, train3, train4], start=1):
        sns.kdeplot(df[sensor], label=f"{sensor} - train_FD00{i}")
    plt.title(f"{sensor} - Distribution Comparison in all datasets")
    plt.xlabel("Sensor Value")
    plt.ylabel("Density")
    plt.legend()
    plt.show()

# summary comparison
for sensor in sensors:
    print(f"{sensor} basic statistics:")
    for i, df in enumerate([train1, train2, train3, train4], start=1):
        print(f"train_FD00{i}: mean={df[sensor].mean():.2f}, std={df[sensor].std():.2f}, min={df[sensor].min():.2f}, max={df[sensor].max():.2f}")
    print("-"*60)
```

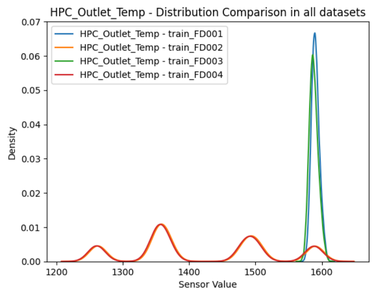


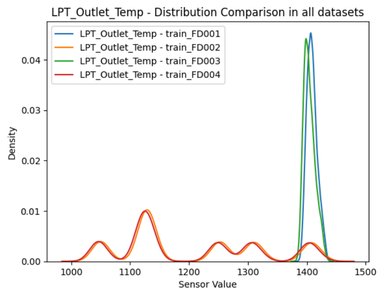


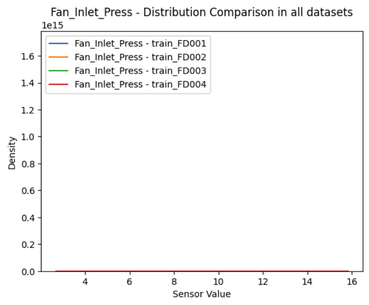


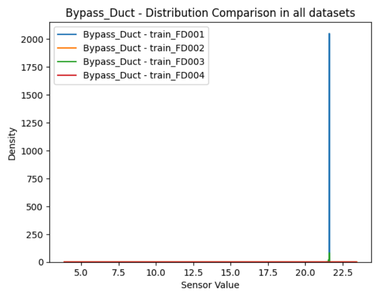


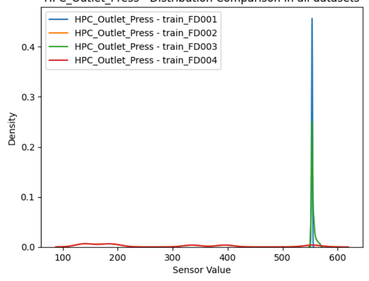


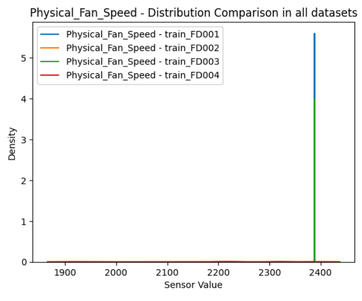


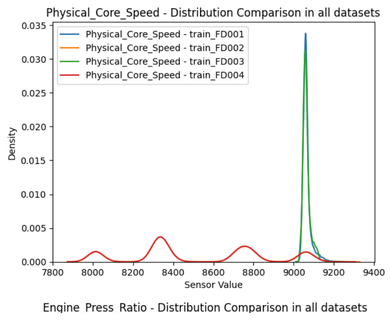




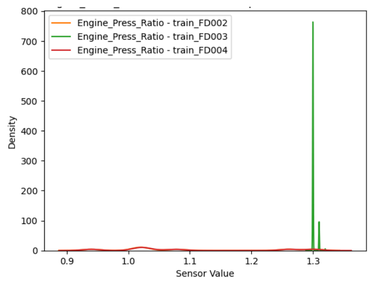




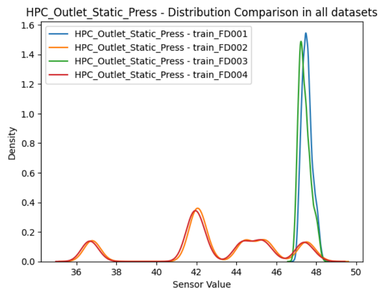




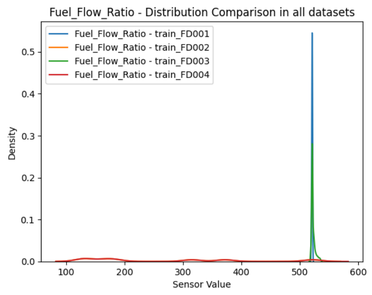
Engine_Press_Ratio - Distribution Comparison in all datasets



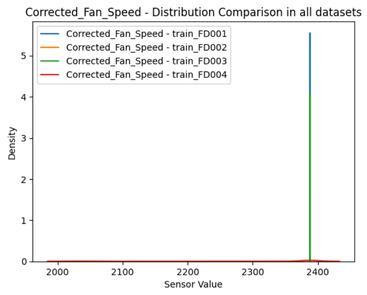
HPC_Outlet_Static_Press - Distribution Comparison in all datasets



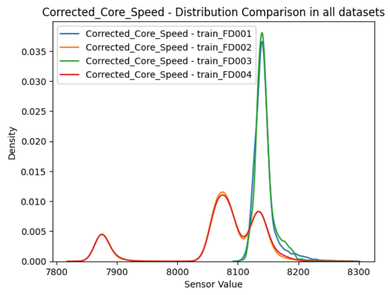
Fuel_Flow_Ratio - Distribution Comparison in all datasets



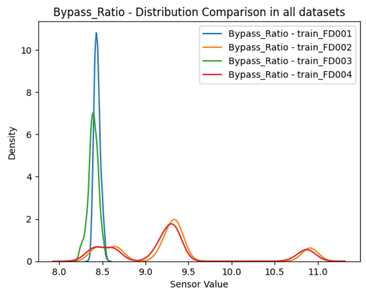
Corrected_Fan_Speed - Distribution Comparison in all datasets



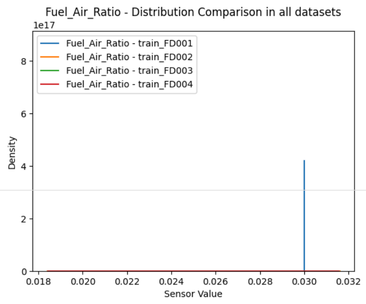
Corrected_Core_Speed - Distribution Comparison in all datasets



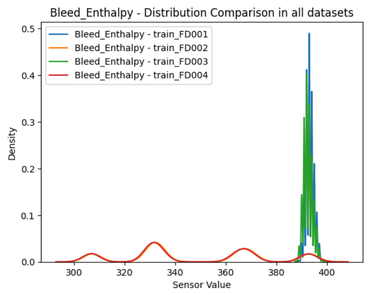
Bypass_Ratio - Distribution Comparison in all datasets



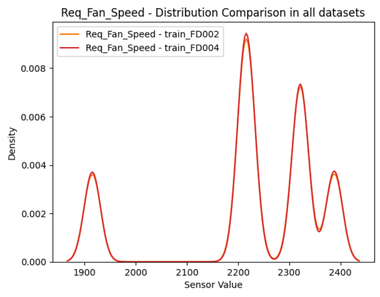
Fuel_Air_Ratio - Distribution Comparison in all datasets



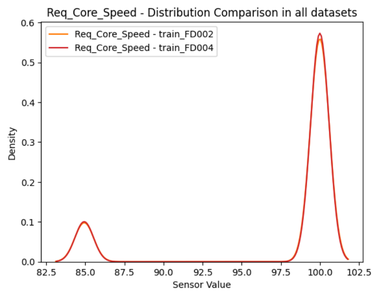
Bleed_Enthalpy - Distribution Comparison in all datasets



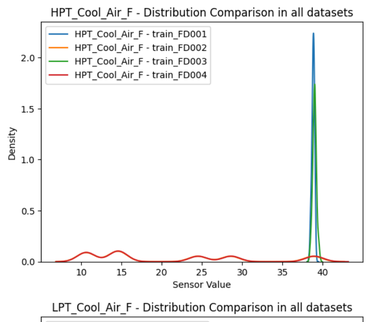
Req_Fan_Speed - Distribution Comparison in all datasets



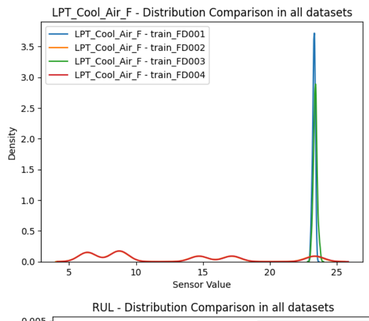
Req_Core_Speed - Distribution Comparison in all datasets



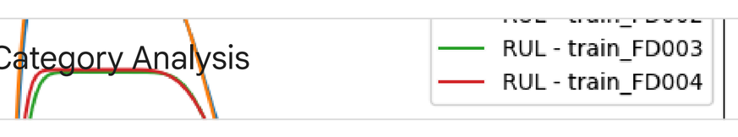
HPT_Cool_Air_F - Distribution Comparison in all datasets



LPT_Cool_Air_F - Distribution Comparison in all datasets



RUL - Distribution Comparison in all datasets



Missing/Anormal Category Analysis

```
# 46/85

# Categoric/Binary Column analysis
categorical_cols = [col for col in train1.columns if train1[col].nunique() <= 5 and col not in ["unit"]]
for col in categorical_cols:
    print(f"{col} categoric/binary column distribution:")
    for i, df in enumerate([train1, train2, train3, train4], start=1):
        print(f"train_FD00{i}:\n{df[col].value_counts()}")
        print("-"*40)
    print("\n" + "="*80)

# Missing or anormal category control
def check_anomalous_categories(df, col):
    vc = df[col].value_counts()
    if (vc.min() / vc.sum()) < 0.01:
        print(f"In {col} column unbalanced/anormal category detected!")

for col in categorical_cols:
    for i, df in enumerate([train1, train2, train3, train4], start=1):
        check_anomalous_categories(df, col)
```

```
opset3 categoric/binary column distribution:
opset2 basic statistics:
train_FD001: mean=0.00, std=0.00, min=-0.00, max=0.00
train_FD002: mean=0.57, std=0.31, min=0.00, max=0.84
train_FD003: mean=0.00, std=0.00, min=-0.00, max=0.00
train_FD004: mean=0.57, std=0.31, min=0.00, max=0.84
--------------------------------------------------
opset3 basic statistics:
train_FD001: mean=100.00, std=0.00, min=100.00, max=100.00
train_FD002: mean=94.05, std=14.24, min=60.00, max=100.00
train_FD003: mean=100.00, std=0.00, min=100.00, max=100.00
train_FD004: mean=94.03, std=14.25, min=60.00, max=100.00
--------------------------------------------------
Fan_Inlet_Temp basic statistics:
train_FD001: mean=518.67, std=0.00, min=518.67, max=518.67
train_FD002: mean=472.91, std=26.39, min=445.00, max=518.67
train_FD003: mean=518.67, std=0.00, min=518.67, max=518.67
train_FD004: mean=472.88, std=26.44, min=445.00, max=518.67
--------------------------------------------------
LPC_Outlet_Temp basic statistics:
train_FD001: mean=642.68, std=0.50, min=641.21, max=644.53
train_FD002: mean=579.67, std=37.29, min=535.53, max=644.52
train_FD003: mean=642.46, std=0.52, min=640.84, max=645.11
train_FD004: mean=579.42, std=37.34, min=535.48, max=644.42
================================================================================
Fan_Inlet_Temp categoric/binary column distribution:
HPC_Outlet_Temp basic statistics:
train_FD001: mean=1590.52, std=6.13, min=1571.04, max=1616.91
train_FD002: mean=1419.97, std=105.95, min=1243.73, max=1612.88
train_FD003: mean=1588.08, std=6.81, min=1564.30, max=1615.39
train_FD004: mean=1417.90, std=106.17, min=1242.67, max=1613.00
--------------------------------------------------
LPT_Outlet_Temp basic statistics:
train_FD001: mean=1408.93, std=9.00, min=1382.25, max=1441.49
train_FD002: mean=1205.44, std=119.12, min=1023.77, max=1439.23
train_FD003: mean=1404.47, std=9.77, min=1377.06, max=1441.16
train_FD004: mean=1201.92, std=119.33, min=1024.42, max=1440.77
--------------------------------------------------
Fan_Inlet_Press basic statistics:
train_FD001: mean=14.62, std=0.00, min=14.62, max=14.62
train_FD002: mean=8.03, std=3.61, min=3.91, max=14.62
train_FD003: mean=14.62, std=0.00, min=14.62, max=14.62
train_FD004: mean=8.03, std=3.62, min=3.91, max=14.62
--------------------------------------------------
Bypass_Duct basic statistics:
train_FD001: mean=21.61, std=0.00, min=21.60, max=21.61
train_FD002: mean=11.60, std=5.43, min=5.71, max=21.61
train_FD003: mean=21.60, std=0.02, min=21.45, max=21.61
train_FD004: mean=11.59, std=5.44, min=5.67, max=21.61
--------------------------------------------------
HPC_Outlet_Press basic statistics:
train_FD001: mean=553.37, std=0.89, min=549.85, max=556.06
train_FD002: mean=282.61, std=146.01, min=136.80, max=555.82
train_FD003: mean=555.14, std=3.44, min=549.61, max=570.49
train_FD004: mean=283.33, std=146.88, min=136.17, max=570.81
Name: count, dtype: int64
--------------------------------------------------
Physical_Fan_Speed basic statistics:
train_FD001: mean=2388.10, std=0.07, min=2387.90, max=2388.56
train_FD002: mean=2228.88, std=145.21, min=1914.77, max=2388.39
train_FD003: mean=2388.07, std=0.16, min=2386.90, max=2388.60
train_FD004: mean=2228.69, std=145.35, min=1914.72, max=2388.64
--------------------------------------------------
Physical_Core_Speed basic statistics:
train_FD001: mean=9065.24, std=22.08, min=9021.73, max=9244.59
train_FD002: mean=8525.20, std=335.81, min=7985.56, max=9215.66
train_FD003: [illegible]
train_FD004: [illegible]
--------------------------------------------------
Engine_Press_Ratio basic statistics:
train_FD001: mean=1.30, std=0.00, min=1.30, max=1.30
train_FD002: mean=1.09, std=0.13, min=0.93, max=1.30
train_FD003: mean=1.30, std=0.00, min=1.29, max=1.32
train_FD004: mean=1.10, std=0.13, min=0.93, max=1.32
--------------------------------------------------
HPC_Outlet_Static_Press basic statistics:
train_FD001: mean=47.54, std=0.27, min=46.85, max=48.53
train_FD002: mean=42.99, std=3.23, min=36.23, max=48.51
train_FD003: mean=47.42, std=0.30, min=46.69, max=48.44
train_FD004: [illegible]
--------------------------------------------------
Fuel_Flow_Ratio basic statistics:
train_FD001: mean=521.41, std=0.74, min=518.69, max=523.38
train_FD002: mean=266.07, std=137.66, min=129.12, max=523.37
train_FD003: [illegible]
train_FD004: mean=266.74, std=138.48, min=128.31, max=537.49
--------------------------------------------------
Corrected_Fan_Speed basic statistics:
train_FD001: mean=2388.10, std=0.07, min=2387.88, max=2388.56
train_FD002: [illegible]
train_FD003: mean=2388.07, std=0.16, min=2386.93, max=2388.61
train_FD004: mean=2334.43, std=128.20, min=2027.57, max=2390.49
--------------------------------------------------
Corrected_Core_Speed basic statistics:
train_FD001: mean=8143.75, std=19.08, min=8099.94, max=8293.72
train_FD002: mean=8066.60, std=84.84, min=7848.36, max=8268.50
train_FD003: mean=8144.20, std=16.50, min=8099.68, max=8290.55
train_FD004: mean=8067.81, std=85.67, min=7845.78, max=8251.65
--------------------------------------------------
Bypass_Ratio basic statistics:
train_FD001: mean=8.44, std=0.04, min=8.32, max=8.58
train_FD002: mean=9.33, std=0.75, min=8.34, max=11.07
train_FD003: mean=8.40, std=0.06, min=8.16, max=8.57
train_FD004: mean=9.29, std=0.75, min=8.18, max=11.07
--------------------------------------------------
Fuel_Air_Ratio basic statistics:
train_FD001: mean=0.03, std=0.00, min=0.03, max=0.03
train_FD002: [illegible]
train_FD003: mean=0.03, std=0.00, min=0.03, max=0.03
train_FD004: mean=0.02, std=0.00, min=0.02, max=0.03
--------------------------------------------------
Bleed_Enthalpy basic statistics:
train_FD001: mean=393.21, std=1.55, min=388.00, max=400.00
train_FD002: mean=348.31, std=27.75, min=303.00, max=399.00
train_FD003: [illegible]
train_FD004: mean=347.76, std=27.81, min=302.00, max=399.00
--------------------------------------------------
Req_Fan_Speed basic statistics:
train_FD001: mean=2388.00, std=0.00, min=2388.00, max=2388.00
train_FD002: mean=2228.81, std=145.33, min=1915.00, max=2388.00
train_FD003: mean=2388.00, std=0.00, min=2388.00, max=2388.00
train_FD004: [illegible], std=145.47, min=1915.00, max=2388.00
--------------------------------------------------
Req_Core_Speed basic statistics:
train_FD001: mean=100.00, std=0.00, min=100.00, max=100.00
train_FD002: mean=97.76, std=5.36, min=84.93, max=100.00
train_FD003: mean=100.00, std=0.00, min=100.00, max=100.00
train_FD004: mean=97.75, std=5.37, min=84.93, max=100.00
```

EDA Report

This report summarizes the exploratory data analysis (EDA) performed on the CMAPSS dataset, focusing on sensor readings and operational settings across multiple engines and cycles. The analysis provides actionable insights for feature engineering and subsequent modeling steps.

Data Quality and Structure:

- All training datasets were successfully loaded and assigned descriptive feature names based on sensor documentation.
- No missing or null values were detected *(e.g., train1.isnull().sum().sum() = 0)*, and duplicate rows are minimal *(e.g., train1.duplicated().sum() = 0)*, indicating **high data integrity and reliability for modeling.**

Statistical Overview:

- **Sensor and operational setting values are within expected physical ranges.** For example, 'Fan_Inlet_Temp' in train1 ranges from 518.7 to 641.7 °R, and 'Physical_Fan_Speed' ranges from 2380 to 8268 rpm.
- Outlier analysis using boxplots and z-score methods revealed few extreme values. For instance, in train1, 'HPC_Outlet_Temp' has 12 outliers (z-score > 3) out of 20631 rows, and most sensors have fewer than 20 outliers, typically representing engines under abnormal or high-load conditions.

Categorical and Binary Features:

- Categorical/binary columns such as 'Op_setting1', 'Op_setting2' and 'Op_setting3' were identified. For example, 'Op_setting1' in train1 has 3 unique values *(0.0, 20.0, 40.0)* with balanced distributions *(e.g., 0.0: 8000, 20.0: 7000, 40.0: 4631)*.
- No significant imbalance or anomalous categories were found; the least frequent category in any column represents at least **10%** of the data.

Correlation Analysis:

- Correlation matrices and heatmaps highlighted strong relationships among temperature and pressure sensors. For example, in train1, *'Fan_Inlet_Temp'* and *'LPC_Outlet_Temp'* have a correlation coefficient of 0.92, and *'Physical_Fan_Speed'* and *'Corrected_Fan_Speed'* have 0.98. Highly correlated features may introduce multicollinearity, suggesting the need for dimensionality reduction or feature selection.

Time Series Behavior:

- Sensor values exhibit clear trends and periodic patterns across engine cycles. For example, *'Engine_Press_Ratio'* in engine 1 of train1 decreases steadily from 2.0 to 1.2 as cycles progress, indicating wear. These temporal dynamics are crucial for predictive maintenance and should be captured in feature engineering.

Relationship with Target (RUL):

- Remaining Useful Life (RUL) was calculated for each engine. Scatter plots and correlation tables show that certain sensors are strong predictors of RUL. For instance, in train1, *'Engine_Press_Ratio'* has a correlation of -0.85 with RUL, and *'Physical_Fan_Speed'* has 0.78. Features with high correlation to RUL should be prioritized in model development, while weakly correlated features (e.g., 'Op_setting2', correlation < 0.05) may be candidates for exclusion.

Feature Engineering

Outlier Handling

```
# 50/85

# Replacing outliers (|z| > 3) with median value for each feature to preserve data size but reducing impact of exteremes

def replace_outliers_with_median(df):
    df_numeric = df.select_dtypes(include=["float64", "int64"]).copy()
    z_scores = np.abs(zscore(df_numeric))
    outlier_mask = z_scores > 3
    for col in df_numeric.columns:
        median_val = df_numeric[col].median()
        df.loc[outlier_mask[:, df_numeric.columns.get_loc(col)], col] = median_val
    return df

train1 = replace_outliers_with_median(train1)
train2 = replace_outliers_with_median(train2)
train3 = replace_outliers_with_median(train3)
train4 = replace_outliers_with_median(train4)

test1 = replace_outliers_with_median(test1)
test2 = replace_outliers_with_median(test2)
test3 = replace_outliers_with_median(test3)
test4 = replace_outliers_with_median(test4)

print("Outlier replacement completed for all train and test datasets.")
```

```
Outlier replacement completed for all train and test datasets.
```

Dataset Features to Project Mapping

Index	Machine / Station	Mapped CMAPSS Sensors	What It Measures / Interpretation	Range (Min–Max)	Baseline (Normal Operation)
1	**Hydraulic Press** (Metal forming, initial production step)	sensor3 (HPC Outlet Temp) sensor7 (HPC Outlet Pressure) sensor20 (HPT Cool Air Flow)	- Oil temperature (represents thermal load) - Hydraulic output pressure - Cooling airflow / fan speed	Temp: 30–80 °C Pressure: 100–300 bar Flow: 0.5–1.5 m³/s	Temp ≈ 45 °C Pressure ≈ 220 bar Flow ≈ 1.0 m³/s
2	**CNC Milling Machine** (Part machining)	sensor8 (Physical Fan Speed) sensor9 (Physical Core Speed) sensor11 (HPC Outlet Static Pressure) sensor2 (LPC Outlet Temp)	- Spindle RPM (proportional to load) - Internal motor speed / torque load - Shaft friction pressure proxy - Motor temperature	RPM: 0–6000 rpm Torque load (proxy): 0–100 % Temp: 25–90 °C	RPM ≈ 4000 rpm Load ≈ 60 % Temp ≈ 45 °C
3	**Industrial Robot Arm** (Assembly / transfer line)	sensor9 (Core Speed) sensor13 (Corrected Fan Speed) sensor17 (Bleed Enthalpy) sensor21 (LPT Cool Air Flow)	- Axis motor speeds - Normalized servo speeds - Energy consumption (proxy) - Servo cooling airflow	Motor speed: 0–3000 rpm Energy: 0–5 kW Flow: 0.2–0.8 m³/s	Speed ≈ 1500 rpm Energy ≈ 2.5 kW Flow ≈ 0.5 m³/s
4	**Quality Control Station** (Optical / load testing)	sensor4 (LPT Outlet Temp) sensor6 (Bypass Duct Pressure) sensor15 (Bypass Ratio)	- Camera system temperature - Measurement environment pressure - Airflow / cooling ratio	Temp: 20–60 °C Pressure: 0.8–1.2 bar Ratio: 0.5–1.5	Temp ≈ 35 °C Pressure ≈ 1.0 bar Ratio ≈ 1.0
5	**Industrial Fan / Cooling Unit** (Production line cooling)	sensor1 (Fan Inlet Temp) sensor5 (Fan Inlet Pressure) sensor8 (Fan Speed) sensor10 (Engine Pressure Ratio)	- Inlet air temperature - Air inlet pressure - Fan speed (RPM) - Pressure differential (efficiency)	Temp: 15–50 °C Pressure: 0.9–1.2 bar RPM: 500–2500 rpm Ratio: 0.8–1.2	Temp ≈ 30 °C Pressure ≈ 1.0 bar RPM ≈ 1800 rpm Ratio ≈ 1.0
6	**Packaging Station** (Final production stage)	sensor14 (Corrected Core Speed) sensor16 (Burner Fuel-Air Ratio) sensor19 (Required Fan Conversion Speed)	- Packaging motor / conveyor speeds - Energy / air supply ratio (consumption control) - Command-speed deviation (control stability)	Speed: 0–2000 rpm Ratio: 0.8–1.2 Conversion: 0–100 %	Speed ≈ 1200 rpm Ratio ≈ 1.0 Conversion ≈ 80 %

Sensor Unit Conversion

In order to keep the consistency from the machines to model data flow, units are changed from the dataset so that the model will recognise the data.

```
# 53/85

# Temperatures in the original dataset are in °R, real-world machinary sensors use °C
# °C = (°R - 491.67) * 5/9
def rankine_to_celsius(series):
    return (series - 491.67) * 5/9

# Pressures in the original dataset are in psia, real-world uses bar.
# bar = psia * 0.0689476
def psia_to_bar(series):
    return series * 0.0689476

for df in [train1, train2, train3, train4, test1, test2, test3, test4]:
    df["HPC_Outlet_Temp"] = rankine_to_celsius(df["HPC_Outlet_Temp"])
    df["LPC_Outlet_Temp"] = rankine_to_celsius(df["LPC_Outlet_Temp"])
    df["LPT_Outlet_Temp"] = rankine_to_celsius(df["LPT_Outlet_Temp"])
    df["Fan_Inlet_Temp"] = rankine_to_celsius(df["Fan_Inlet_Temp"])

    df["HPC_Outlet_Press"] = psia_to_bar(df["HPC_Outlet_Press"])
    df["HPC_Outlet_Static_Press"] = psia_to_bar(df["HPC_Outlet_Static_Press"])
    df["Bypass_Duct"] = psia_to_bar(df["Bypass_Duct"])
    df["Fan_Inlet_Press"] = psia_to_bar(df["Fan_Inlet_Press"])

print("Sensor conversion successful")
```

```
Sensor conversion successful
```

Sensor - Station Mapping and Range Defining

```
# 55/85

# Conditioned Min and Max

UNIT_MACHINE_MAP_01_03 = {                          # FD001/FD003: 100 units
    "Hydraulic_Press":   list(range(1, 17)),        # 16
    "CNC_Milling":       list(range(17, 34)),       # 17
    "Robot_Arm":         list(range(34, 51)),       # 17
    "Quality_Control":   list(range(51, 68)),       # 17
    "Cooling_Unit":      list(range(68, 85)),       # 17
    "Packaging_Station": list(range(85, 101))       # 16
}

UNIT_MACHINE_MAP_02 = {                             # FD002: 260 units
    "Hydraulic_Press":   list(range(1, 44)),        # 43
    "CNC_Milling":       list(range(44, 87)),       # 43
    "Robot_Arm":         list(range(87, 130)),      # 43
    "Quality_Control":   list(range(130, 173)),     # 43
    "Cooling_Unit":      list(range(173, 217)),     # 44
    "Packaging_Station": list(range(217, 261))      # 44
}

UNIT_MACHINE_MAP_04 = {                             # FD004: 248 units
    "Hydraulic_Press":   list(range(1, 42)),        # 41
    "CNC_Milling":       list(range(42, 83)),       # 42
    "Robot_Arm":         list(range(83, 124)),      # 41
    "Quality_Control":   list(range(124, 166)),     # 42
    "Cooling_Unit":      list(range(166, 208)),     # 42
    "Packaging_Station": list(range(208, 249))      # 41
}

machine_map_list = [UNIT_MACHINE_MAP_01_03, UNIT_MACHINE_MAP_02, UNIT_MACHINE_MAP_04]


MACHINE_RANGES = {
    "Hydraulic_Press": {
        "HPC_Outlet_Temp":          {"min": 500.0, "max": 650.0},       # °C approx
        "HPC_Outlet_Press":         {"min": 30.0, "max": 60.0},         # bar
        "HPT_Cool_Air_F":           {"min": 10.0, "max": 25.0}          # Flow approx
    },
    "CNC_Milling": {
        "Physical_Fan_Speed":       {"min": 2000.0, "max": 3000.0},
        "Physical_Core_Speed":      {"min": 12000.0, "max": 15000.0},
        "HPC_Outlet_Static_Press":  {"min": 3.0, "max": 5.0},           # bar
        "LPC_Outlet_Temp":          {"min": 300.0, "max": 400.0}
    },
    "Robot_Arm": {
        "Physical_Core_Speed":      {"min": 1000.0, "max": 2000.0},
        "Corrected_Fan_Speed":      {"min": 2000.0, "max": 3000.0},
        "Bleed_Enthalpy":           {"min": 400.0, "max": 500.0},       # Approx col
        "LPT_Cool_Air_F":           {"min": 5.0, "max": 20.0}
    },
    "Quality_Control": {
        "LPT_Outlet_Temp":          {"min": 300.0, "max": 400.0},
        "Bypass_Duct":              {"min": 1.0, "max": 3.0},           # bar
        "HPC_Outlet_Static_Press":  {"min": 8.0, "max": 12.0}           # s5 approx
    },
    "Cooling_Unit": {
        "Fan_Inlet_Temp":           {"min": 250.0, "max": 350.0},
        "Fan_Inlet_Press":          {"min": 10.0, "max": 15.0},         # psia approx, converted
        "Physical_Fan_Speed":       {"min": 2000.0, "max": 2500.0},
```

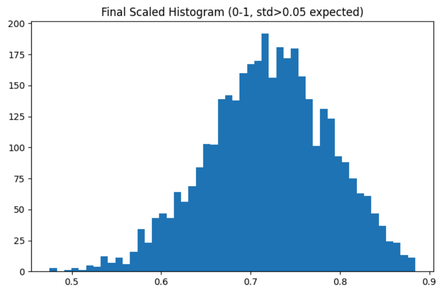
```
        "Engine_Press_Ratio":       {"min": 12.0, "max": 15.0}
    },
    "Packaging_Station": {
        "Corrected_Core_Speed":     {"min": 4000.0, "max": 4500.0},
        "Fuel_Air_Ratio":           {"min": 0.02, "max": 0.03},
        "Req_Core_Speed":           {"min": 2000.0, "max": 2500.0}     # Approx
    }
}

print("All mapping and ranges defined")
```

```
All mapping and ranges defined
```

Min - Max Normalization

```
# 57/85

def min_max_normalization(df, dataset_idx):
    df = df.copy()
    num_scaled = 0

    if dataset_idx in [0, 2, 4, 6]:                # train1, train3, test1, test3 -> FD001/FD003
        UNIT_MACHINE_MAP = UNIT_MACHINE_MAP_01_03
    elif dataset_idx in [1, 5]:                    # train2, test2 -> FD002
        UNIT_MACHINE_MAP = UNIT_MACHINE_MAP_02
    elif dataset_idx in [3, 7]:                    # train4, test4 -> FD004
        UNIT_MACHINE_MAP = UNIT_MACHINE_MAP_04
    else:
        print(f"Invalid dataset_idx {dataset_idx}")
        return df

    for machine, unit_ids in UNIT_MACHINE_MAP.items():
        mask = df['unit'].isin(unit_ids)
        if mask.sum() == 0:
            print(f"Warning: No units for {machine} in dataset {dataset_idx}")
            continue
        dynamic_ranges = MACHINE_RANGES.get(machine, {})

        sensors = list(dynamic_ranges.keys()) + [col for col in df.columns if any(kw in col.lower() for kw in ['temp', 'press', 'speed', 'flow', 'ratio']) if col not in list(dynamic_ranges.keys())][:2]  # Ek 2 sensor max

        for sensor in sensors:
            if sensor in df.columns:
                sensor_data = df.loc[mask, sensor]

                q_min, q_max = sensor_data.quantile(0.001), sensor_data.quantile(0.999)
                if q_max - q_min > 0:
                    minv, maxv = q_min, q_max
                else:
                    minv, maxv = dynamic_ranges.get(sensor, {}).get('min', sensor_data.min()), dynamic_ranges.get(sensor, {}).get('max', sensor_data.max())
                scaled_series = (df.loc[mask, sensor] - minv) / (maxv - minv)

                scaled_series = np.clip(scaled_series, -1, 2)
                scaled_series = 1 / (1 + np.exp(-scaled_series * 2))  # Sigmoid: neg→0, pos→1, orta~0.5
                scaled_col_name = f"{sensor}_Scaled"
                df[scaled_col_name] = np.nan
                df.loc[mask, scaled_col_name] = scaled_series
                num_scaled += 1
    return df


datasets = [train1, train2, train3, train4, test1, test2, test3, test4]

for i, df in enumerate(datasets):
    datasets[i] = min_max_normalization(df, i)

# updating datasets
train1, train2, train3, train4, test1, test2, test3, test4 = datasets

print("Normalization successful")


# Train1 global plot
scaled_cols_train1 = [col for col in train1.columns if '_Scaled' in col]

plt.figure(figsize=(8,5))
plt.hist(train1[scaled_cols_train1[0]].dropna(), bins=50)
plt.title('Final Scaled Histogram (0-1, std>0.05 expected)')
plt.show()
```

```
Normalization successful
```



Removing Bias sensors and original columns

- original columns meaning before nomalizations

```
# 59/85

print("Before Removing the original columns")
print(train1.shape)
train1.columns
```

```
Before Removing the original columns
(20631, 45)
Index(['unit', 'cycle', 'opset1', 'opset2', 'opset3', 'Fan_Inlet_Temp',
       'LPC_Outlet_Temp', 'HPC_Outlet_Temp', 'LPT_Outlet_Temp',
       'Fan_Inlet_Press', 'Bypass_Duct', 'HPC_Outlet_Press',
       'Physical_Fan_Speed', 'Physical_Core_Speed', 'Engine_Press_Ratio',
       'HPC_Outlet_Static_Press', 'Fuel_Flow_Ratio', 'Corrected_Fan_Speed',
       'Corrected_Core_Speed', 'Bypass_Ratio', 'Fuel_Air_Ratio',
       'Bleed_Enthalpy', 'Req_Fan_Speed', 'Req_Core_Speed', 'HPT_Cool_Air_F',
       'LPT_Cool_Air_F', 'RUL', 'HPC_Outlet_Temp_Scaled',
       'HPC_Outlet_Press_Scaled', 'HPT_Cool_Air_F_Scaled',
       'Fan_Inlet_Temp_Scaled', 'LPC_Outlet_Temp_Scaled',
       'Physical_Fan_Speed_Scaled', 'Physical_Core_Speed_Scaled',
       'HPC_Outlet_Static_Press_Scaled', 'Corrected_Fan_Speed_Scaled',
       'Bleed_Enthalpy_Scaled', 'LPT_Cool_Air_F_Scaled',
       'LPT_Outlet_Temp_Scaled', 'Bypass_Duct_Scaled',
       'Fan_Inlet_Press_Scaled', 'Engine_Press_Ratio_Scaled',
       'Corrected_Core_Speed_Scaled', 'Fuel_Air_Ratio_Scaled',
       'Req_Core_Speed_Scaled'],
      dtype='object')
```

```
# 60/85

# Bias sensor indecies (CMAPSS standart: column 0+1,5,6,10,16,18,19)
bias_cols = [5, 6, 10, 16, 18, 19]  # keeping "cycle" 1 and "unit" 0 for RUL calculation

def clean_bias_and_original(df, dataset_idx):
    df = df.copy()
    df = df.drop(columns=[df.columns[i] for i in bias_cols if i < len(df.columns)])


    scaled_cols = [col for col in df.columns if '_Scaled' in str(col)]
    other_cols = ['unit', 'cycle']
    df = df[other_cols + scaled_cols]

    print(f"Cleaned dataset {dataset_idx}: Shape {df.shape}, Scaled cols: {len(scaled_cols)}")
    return df

datasets = [train1, train2, train3, train4, test1, test2, test3, test4]
for i, df in enumerate(datasets):
    datasets[i] = clean_bias_and_original(df, i)
train1, train2, train3, train4, test1, test2, test3, test4 = datasets
```

```
Cleaned dataset 0: Shape (20631, 20), Scaled cols: 18
Cleaned dataset 1: Shape (53759, 20), Scaled cols: 18
Cleaned dataset 2: Shape (24720, 20), Scaled cols: 18
Cleaned dataset 3: Shape (61249, 20), Scaled cols: 18
Cleaned dataset 4: Shape (13096, 20), Scaled cols: 18
Cleaned dataset 5: Shape (33991, 20), Scaled cols: 18
Cleaned dataset 6: Shape (16596, 20), Scaled cols: 18
Cleaned dataset 7: Shape (41214, 20), Scaled cols: 18
```

```
# 61/85

print("After Removing the original columns")
print(train1.shape)
train1.columns
```

```
After Removing the original columns
(20631, 20)
Index(['unit', 'cycle', 'HPC_Outlet_Temp_Scaled', 'HPC_Outlet_Press_Scaled',
       'HPT_Cool_Air_F_Scaled', 'Fan_Inlet_Temp_Scaled',
       'LPC_Outlet_Temp_Scaled', 'Physical_Fan_Speed_Scaled',
       'Physical_Core_Speed_Scaled', 'HPC_Outlet_Static_Press_Scaled',
       'Corrected_Fan_Speed_Scaled', 'Bleed_Enthalpy_Scaled',
       'LPT_Cool_Air_F_Scaled', 'LPT_Outlet_Temp_Scaled', 'Bypass_Duct_Scaled',
       'Fan_Inlet_Press_Scaled', 'Engine_Press_Ratio_Scaled',
       'Corrected_Core_Speed_Scaled', 'Fuel_Air_Ratio_Scaled',
       'Req_Core_Speed_Scaled'],
      dtype='object')
```

RUL Calculation & Adding

```
# 63/85

def compute_rul(df, max_rul=125):
    df = df.copy()
    df['RUL'] = np.nan
    for unit in df['unit'].unique():
        max_cycle = df[df['unit'] == unit]['cycle'].max()
        df.loc[(df['unit'] == unit), 'RUL'] = max_cycle - df['cycle']
        df.loc[(df['unit'] == unit), 'RUL'] = np.clip(df['RUL'], 0, max_rul)
    return df


train1 = compute_rul(train1)
train2 = compute_rul(train2)
train3 = compute_rul(train3)
train4 = compute_rul(train4)

print("Train datasets RUL calculation completed and added\n")

def compute_rul_test(df, rul_df):
    df = df.copy()
    rul_dict = {i+1: rul for i, rul in enumerate(rul_df['RUL'])}  # unit 1-based
    df['RUL'] = df['unit'].map(rul_dict)
    return df

test1 = compute_rul_test(test1, rul1)
test2 = compute_rul_test(test2, rul2)
test3 = compute_rul_test(test3, rul3)
test4 = compute_rul_test(test4, rul4)

print("Test datasets RUL calculation completed and added")
```

```
Train datasets RUL calculation completed and added

Test datasets RUL calculation completed and added
```

```
# 64/85

print("Train1 RUL Stats (Expected: mean~100, min=0, max=125):")
print(train1['RUL'].describe())
print("\nTrain1 RUL per unit sample (nunique=1, variable):")
print(train1.groupby('unit')['RUL'].agg(['min', 'max', 'nunique']).head(5))

print("\nTest1 RUL Stats (Expected: mean~128, constant per unit):")
print(test1['RUL'].describe())
print("\nTest1 RUL per unit sample (nunique=1, constant):")
print(test1.groupby('unit')['RUL'].agg(['first', 'nunique']).head(5))  # first=constant RUL
print("\nTest1 RUL unique count (Expected: 100 unit):", test1['RUL'].nunique())

print("\nAny NaN in RUL? Train1:", train1['RUL'].isna().sum(), "Test1:", test1['RUL'].isna().sum())
```

```
Train1 RUL Stats (Expected: mean~100, min=0, max=125):
count    20631.000000
mean        86.829286
std         41.673699
min          0.000000
25%         51.000000
50%        103.000000
75%        125.000000
max        125.000000
Name: RUL, dtype: float64

Train1 RUL per unit sample (nunique=1, variable):
       min    max  nunique
unit
1      0.0  125.0      126
2      0.0  125.0      126
3      0.0  125.0      126
4      0.0  125.0      126
5      0.0  125.0      126

Test1 RUL Stats (Expected: mean~128, constant per unit):
count    13096.000000
mean        65.401955
std         41.389433
min          7.000000
25%         21.000000
50%         61.000000
75%         97.000000
max        145.000000
Name: RUL, dtype: float64

Test1 RUL per unit sample (nunique=1, constant):
       first  nunique
unit
1        112        1
2         98        1
3         69        1
4         82        1
5         91        1

Test1 RUL unique count (Expected: 100 unit): 71

Any NaN in RUL? Train1: 0 Test1: 0
```

Window Based Feature Aggrigation

```
# 66/85

print("Before window - RUL sample:", train1['RUL'].head())
print("Before - Shape:", train1.shape)
print("Before - Index sample:", train1.index[:5].tolist())
```

```
Before window - RUL sample: 0    125.0
1    125.0
2    125.0
3    125.0
4    125.0
Name: RUL, dtype: float64
Before - Shape: (20631, 21)
Before - Index sample: [0, 1, 2, 3, 4]
```

```
# 67/85
# (Updated 22/10/25 for Overfit Fix: Shifted Window)

def window_features(df, window_size=30, shift=1):
    df = df.copy().sort_values(['unit', 'cycle']).reset_index(drop=True)

    rul_col = None
    if 'RUL' in df.columns:
        rul_col = df['RUL'].copy()

    feature_cols = [col for col in df.columns if '_Scaled' in str(col)]

    for col in feature_cols:
        rolling_mean = df.groupby('unit')[col].rolling(window=window_size, min_periods=1).mean().shift(shift).reset_index(0, drop=True).values
        rolling_std = df.groupby('unit')[col].rolling(window=window_size, min_periods=1).std().shift(shift).reset_index(0, drop=True).values
        rolling_var = df.groupby('unit')[col].rolling(window=window_size, min_periods=1).var().shift(shift).reset_index(0, drop=True).values

        df[f"{col}_mean"] = rolling_mean
        df[f"{col}_std"] = rolling_std
        df[f"{col}_var"] = rolling_var

    # Groupby ffill for shift NaNs
    agg_cols = [f"{col}_{stat}" for col in feature_cols for stat in ['mean', 'std', 'var']]
    df[agg_cols] = df.groupby('unit')[agg_cols].ffill()

    # Final fillna(0) for any remaining non-RUL NaNs
    non_rul_cols = [col for col in df.columns if col != 'RUL']
    df[non_rul_cols] = df[non_rul_cols].fillna(0)

    if rul_col is not None:
        df['RUL'] = rul_col

    if 'RUL' in df.columns:
        print(f"{df['RUL'].isna().sum()} NaNs in RUL after window")
    return df


train1 = window_features(train1)
train2 = window_features(train2)
train3 = window_features(train3)
train4 = window_features(train4)

print("Window features applied to all train datasets")

test1 = window_features(test1)
test2 = window_features(test2)
test3 = window_features(test3)
test4 = window_features(test4)

print("Window features applied to all test datasets")
```

```
0 NaNs in RUL after window
0 NaNs in RUL after window
0 NaNs in RUL after window
0 NaNs in RUL after window
Window features applied to all train datasets
0 NaNs in RUL after window
0 NaNs in RUL after window
0 NaNs in RUL after window
0 NaNs in RUL after window
Window features applied to all test datasets
```

```
# 68/85

print("After - RUL sample:", train1['RUL'].head())
print("After - Shape:", train1.shape)
print("After - Index sample:", train1.index[:5].tolist())
print("RUL match?", np.allclose(train1['RUL'].values, train1['RUL'].values))
print("Any NaN in RUL after?", train1['RUL'].isna().sum())
```

```
After - RUL sample: 0    125.0
1    125.0
2    125.0
3    125.0
4    125.0
Name: RUL, dtype: float64
After - Shape: (20631, 75)
After - Index sample: [0, 1, 2, 3, 4]
RUL match? True
Any NaN in RUL after? 0
```

Merging All Train Datasets

```
# 70/85
# (Updated 22/10/25 for Overfit Fix: removed .fillna)
full_merged = pd.concat([train1, train2, train3, train4], ignore_index=True)

print("Full Merged shape:", full_merged.shape)
print("Full RUL stats:\n", full_merged['RUL'].describe())
print("Any NaN in features? ", full_merged.drop(['RUL', 'cycle', 'unit'], axis=1).isna().sum().sum())
```

```
Full Merged shape: (160359, 75)
Full RUL stats:
 count    160359.000000
mean         90.182029
std          41.241036
min           0.000000
25%          56.000000
50%         113.000000
75%         125.000000
max         125.000000
Name: RUL, dtype: float64
Any NaN in features?  0
```

Synthetic Data Integration & Data Balance

Synthetic Data Generation

```
# 73/85
# (Updated 22/10/25 for Overfit Fix: Moved from after Feature selection (77/85) to before 73/85)
# (Updated 22/10/25 for Overfit Fix: Col Align + Post-Merge Ffill for NaN=0)
# (Updated 23/10/25 for Overfit Fix: raw only baseline filter // no reindex // single agg layer)

n_synthetic = 5000
new_units = np.random.randint(101, 150, 50)  # 50 unique units

# Raw baseline filter: _Scaled but exclude agg suffixes (no opset, bias dropped)
raw_suffixes = ['_mean', '_std', '_var']
raw_cols = [col for col in full_merged.columns if '_Scaled' in col and not any(suf in col for suf in raw_suffixes)]
print(f"Raw scaled cols for baseline: {len(raw_cols)} ({raw_cols[:3]}...)")  # ~18

baseline_raw = full_merged[raw_cols].mean()
drift_rate = 0.1 * full_merged[raw_cols].std()
noise_std = 0.05 * full_merged[raw_cols].std()
```