# Optimize Manufacturing Operations with a Predictive Maintenance Model

## 1. Data Exploration & Validation Strategy

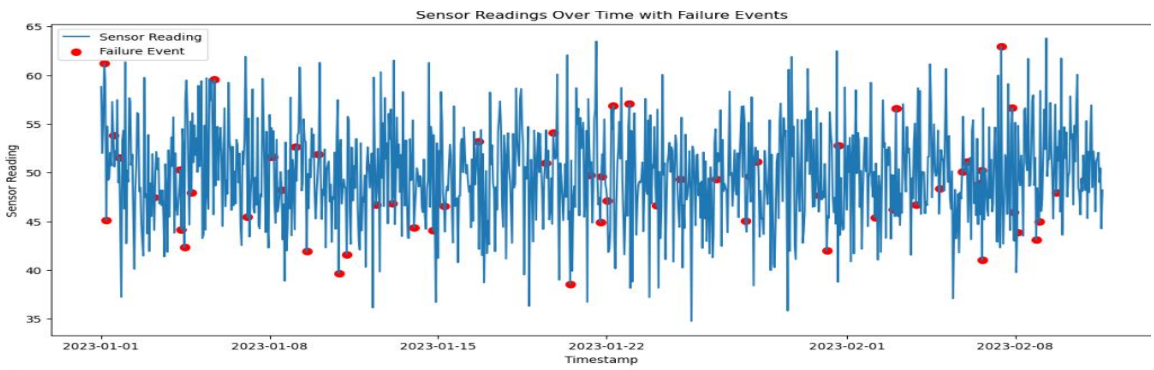
- Plot all sensor readings (temperature, vibration, pressure, current, etc.) over time and overlay known failure events.
    - This helps visually identify patterns, anomalies, and pre-failure trends.
- Check for missing timestamps, duplicates, sudden jumps, or sensor drift.
- Study the temporal behavior of machines (daily patterns, shift-wise variations, seasonal effects).
- Use a time-aware validation strategy — traditional k-fold must not be used.
    - Apply TimeSeriesSplit, rolling-forward validation, or expanding-window validation to simulate real-world model deployment.
- Ensure that validation data always occurs later in time than training data to prevent data leakage.

```
print(os.listdir("predictive_maintenance_Dataset"))
```

```
['ai4i2020.csv']
```

```
import pandas as pd

df = pd.read_csv("predictive_maintenance_Dataset/ai4i2020.csv")

df.head()
```

| | UDI | Product ID | Type | Air temperature [K] | Process temperature [K] | Rotational speed [rpm] | Torque [Nm] | Tool wear [min] | Machine failure | TWF | HDF | PWF | OSF | RNF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | M14860 | M | 298.1 | 308.6 | 1551 | 42.8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | L47181 | L | 298.2 | 308.7 | 1408 | 46.3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 3 | L47182 | L | 298.1 | 308.5 | 1498 | 49.4 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 4 | L47183 | L | 298.2 | 308.6 | 1433 | 39.5 | 7 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 5 | L47184 | L | 298.2 | 308.7 | 1408 | 40.0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 |




```
Fold 1
Training set: 170 samples
Test set: 166 samples

Fold 2
Training set: 336 samples
Test set: 166 samples

Fold 3
Training set: 502 samples
Test set: 166 samples

Fold 4
Training set: 668 samples
Test set: 166 samples

Fold 5
Training set: 834 samples
Test set: 166 samples
```

## 2. Feature Engineering

- Create features that capture trends, volatility, and time-based patterns in signals.
- Strong features commonly include:
  - Rolling statistics:
    - Rolling mean, rolling std deviation
    - Rolling min/max
    - Rolling median
    - Useful windows: 1 hr, 3 hr, 6 hr, 12 hr
  - Exponential moving averages (EMA) to smooth noisy sensor readings.
  - Gradient / Rate-of-change:
    - First derivative, second derivative
    - Helps detect increasing vibration/temperature before failure
  - Operational features:
    - Time since last maintenance
    - Machine age
    - Shift number (morning/evening/night)
- Capture consistent lookback windows so the model sees fixed-size feature sets for each timestamp.

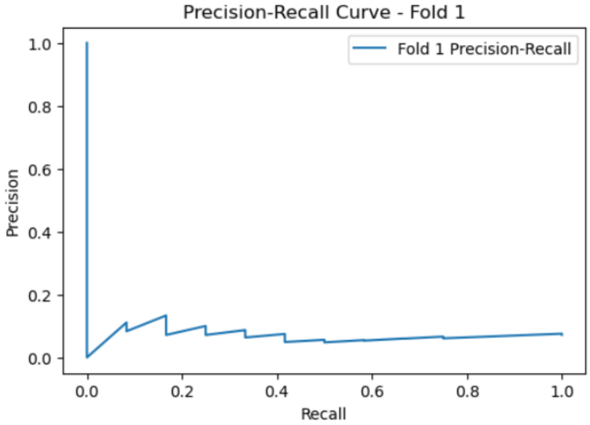
[49]:
```python
df['rolling_std_3'] = df['sensor_reading'].rolling(window=3).std().bfill()
df['ema_5'] = df['sensor_reading'].ewm(span=5, adjust=False).mean()
df['gradient'] = np.gradient(df['sensor_reading'])
df['time_since_maintenance'] = np.random.randint(1, 100, size=len(df))
df['operational_age'] = (df['timestamp'] - df['timestamp'].min()).dt.total_seconds() / 3600

print(df.head(10))
```

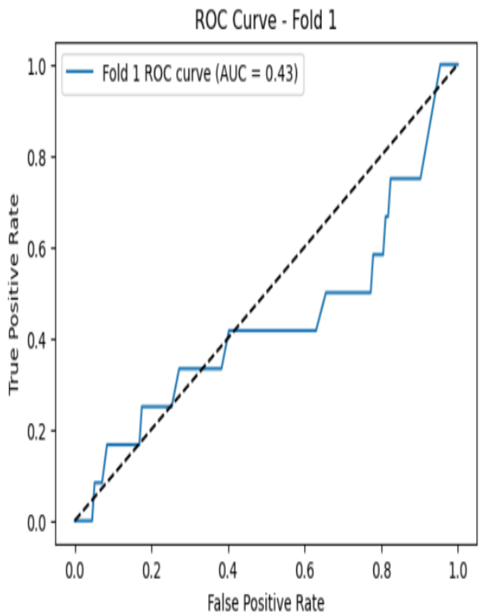
```
            timestamp  sensor_reading  failure_event  rolling_std_3  \
0 2023-01-01 00:00:00       58.820262              0       3.422770
1 2023-01-01 01:00:00       52.000786              0       3.422770
2 2023-01-01 02:00:00       54.893690              0       3.422770
3 2023-01-01 03:00:00       61.204466              1       4.706423
4 2023-01-01 04:00:00       59.337790              0       3.241923
5 2023-01-01 05:00:00       45.113611              1       8.800827
6 2023-01-01 06:00:00       54.750442              0       7.259931
7 2023-01-01 07:00:00       49.243214              0       4.834799
8 2023-01-01 08:00:00       49.483906              0       3.112445
9 2023-01-01 09:00:00       52.052993              0       1.557401

       ema_5  gradient  time_since_maintenance  operational_age
0  58.820262 -6.819476                      91              0.0
1  56.547103 -1.963286                      73              1.0
2  55.995965  4.601840                      79              2.0
3  57.732132  2.222050                      36              3.0
4  58.267352 -8.045428                      77              4.0
5  53.882771 -2.293674                      82              5.0
6  54.171995  2.064802                      88              6.0
7  52.529068 -2.633268                      82              7.0
8  51.514014  1.404889                      97              8.0
9  51.693673  0.618156                      50              9.0
```

## 3. Modeling for Imbalanced Data

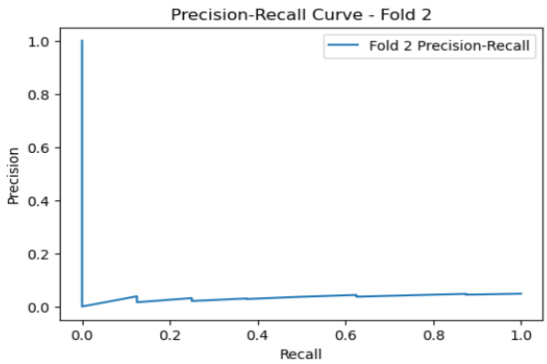
- Failure data is naturally imbalanced; only a small percentage of timestamps correspond to failures.
- Apply class imbalance handling:
  - Use scale_pos_weight in XGBoost/LightGBM
  - Or class_weight='balanced' in RandomForest or Logistic Regression
- Optionally test oversampling methods like SMOTE — only inside training folds, never across time boundaries.
- Primary evaluation metrics should be:
  - Precision-Recall Curve
  - F1-score
  - PR-AUC (Precision–Recall AUC)
- Avoid accuracy as it gives misleading results for imbalanced problems.
- Tune decision thresholds for your operational needs (high recall for safety, high precision for reducing false alarms).

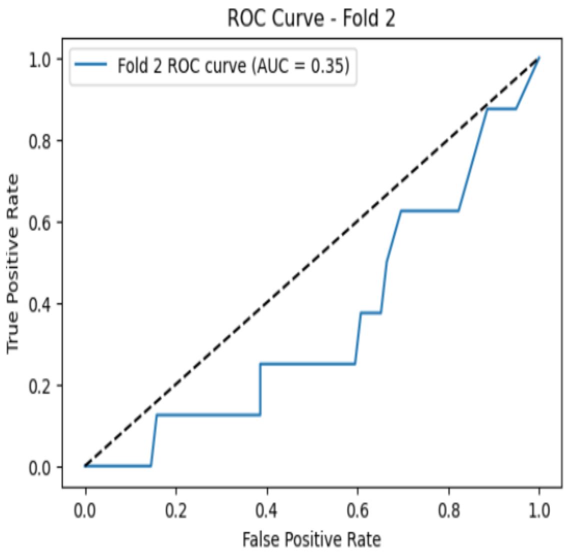
```
Fold 1 Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.23      0.36       154
           1       0.05      0.50      0.09        12

    accuracy                           0.25       166
   macro avg       0.45      0.36      0.22       166
weighted avg       0.80      0.25      0.34       166

Fold 1 Average Precision (AP): 0.0808
Fold 1 F1-score: 0.0876
```



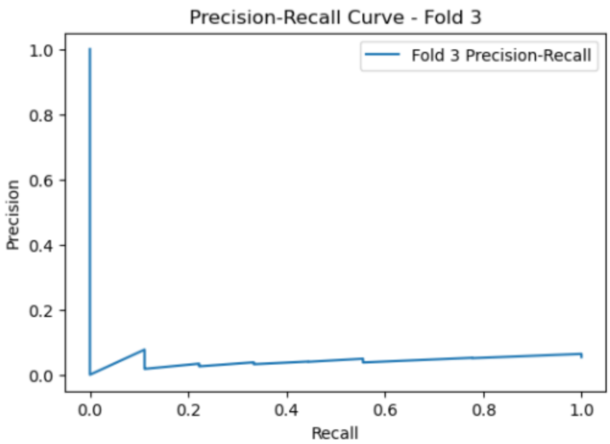



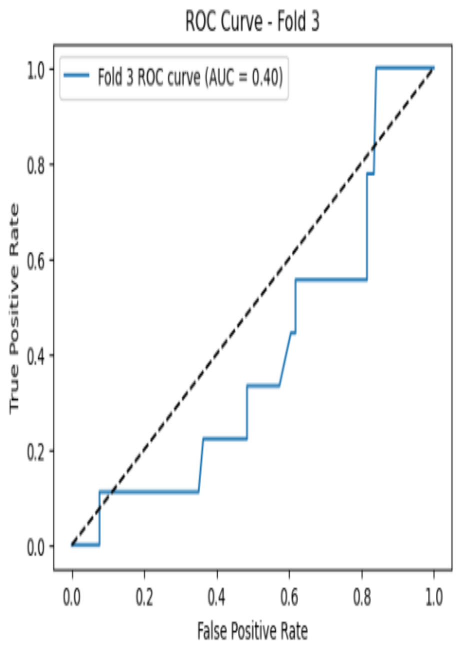
```
Fold 2 Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.27      0.42       158
           1       0.04      0.62      0.08         8

    accuracy                           0.29       166
   macro avg       0.49      0.45      0.25       166
weighted avg       0.89      0.29      0.41       166

Fold 2 Average Precision (AP): 0.0405
Fold 2 F1-score: 0.0781
```



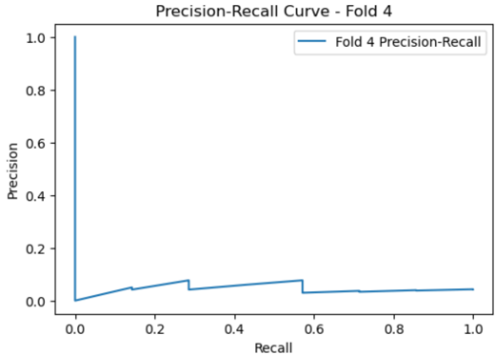



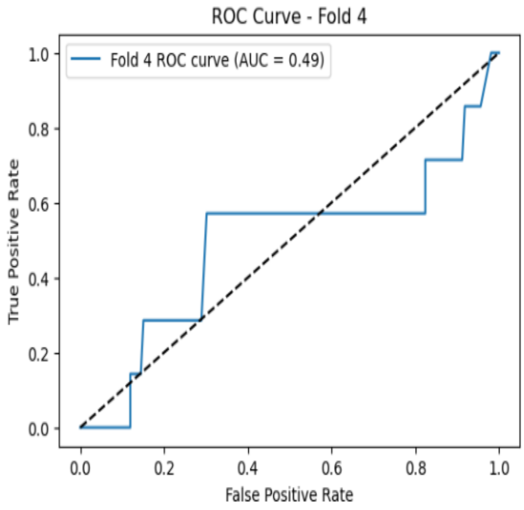

```
Fold 3 Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.12      0.22       157
           1       0.06      1.00      0.12         9

    accuracy                           0.17       166
   macro avg       0.53      0.56      0.17       166
weighted avg       0.95      0.17      0.21       166

Fold 3 Average Precision (AP): 0.0522
Fold 3 F1-score: 0.1154
```



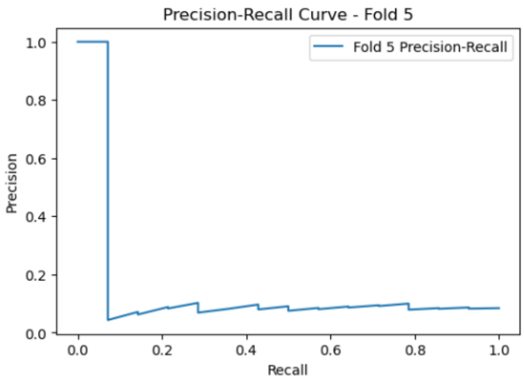



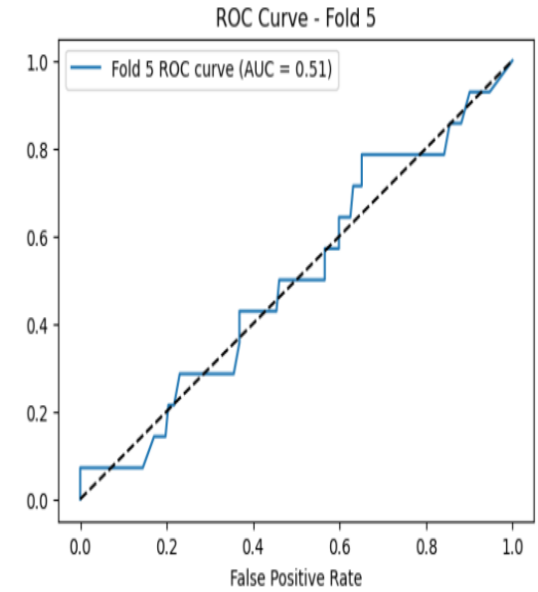
```
Fold 4 Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.13      0.22       159
           1       0.03      0.71      0.07         7

    accuracy                           0.15       166
   macro avg       0.47      0.42      0.14       166
weighted avg       0.87      0.15      0.21       166

Fold 4 Average Precision (AP): 0.0547
Fold 4 F1-score: 0.0662
```

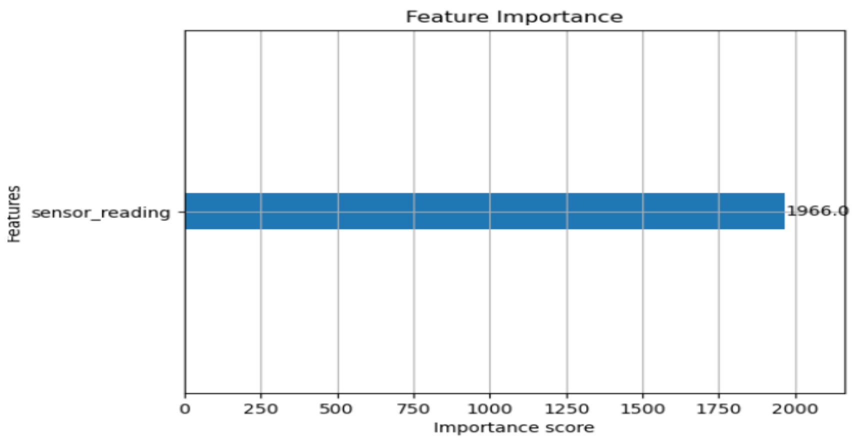



```
Fold 5 Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.21      0.34       152
           1       0.08      0.79      0.15        14

    accuracy                           0.26       166
   macro avg       0.50      0.50      0.25       166
weighted avg       0.84      0.26      0.33       166

Fold 5 Average Precision (AP): 0.1541
Fold 5 F1-score: 0.1517
```



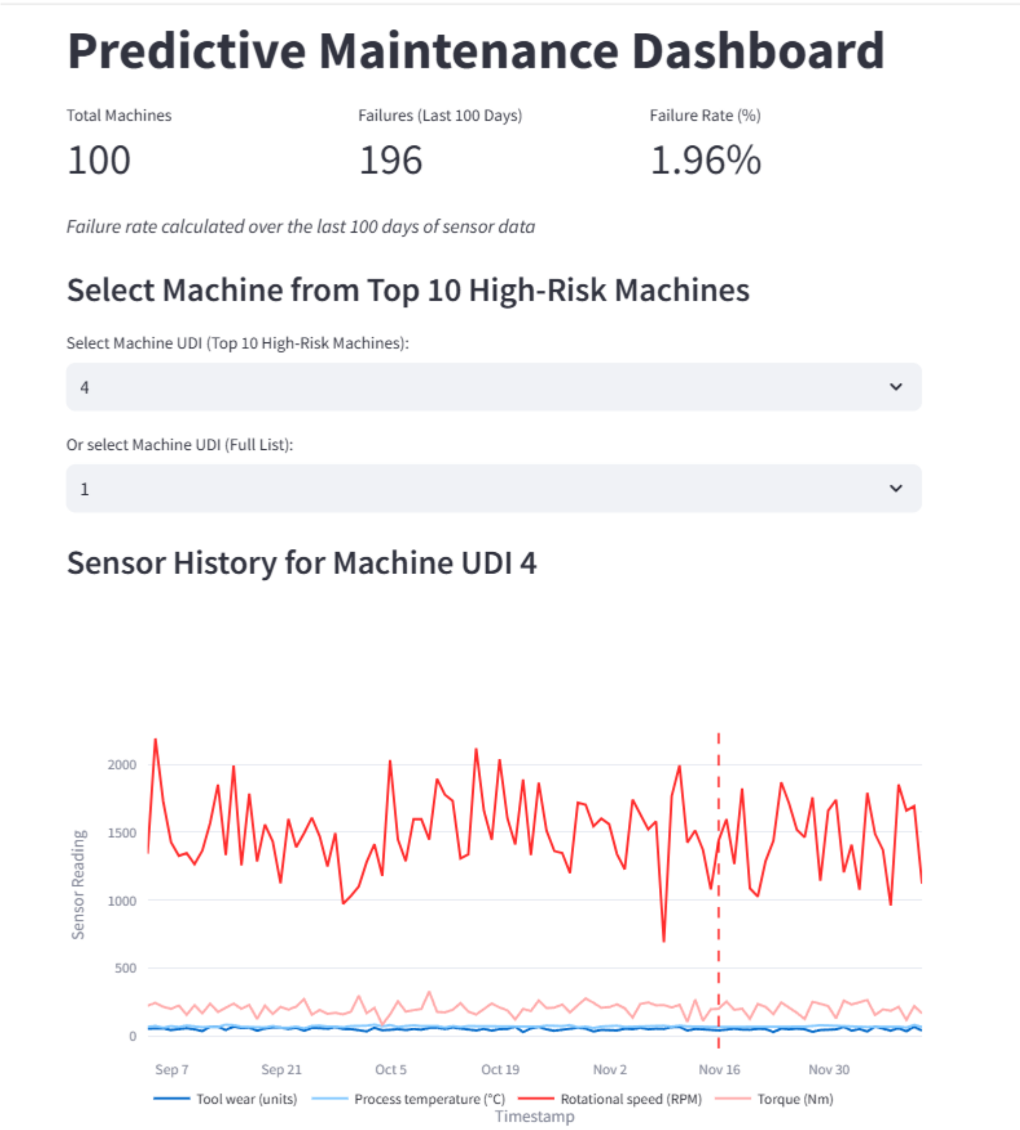



```python
tscv = TimeSeriesSplit(n_splits=5)

print("TimeSeriesSplit Ready!")
```

```
Dataset Loaded Successfully!
   UDI Product ID Type  Air temperature [K]  Process temperature [K]  \
0    1     M14860    M                298.1                    308.6
1    2     L47181    L                298.2                    308.7
2    3     L47182    L                298.1                    308.5
3    4     L47183    L                298.2                    308.6
4    5     L47184    L                298.2                    308.7

   Rotational speed [rpm]  Torque [Nm]  Tool wear [min]  Machine failure  TWF  \
0                    1551         42.8                0                0    0
1                    1408         46.3                3                0    0
2                    1498         49.4                5                0    0
3                    1433         39.5                7                0    0
4                    1408         40.0                9                0    0

   HDF  PWF  OSF  RNF
0    0    0    0    0
1    0    0    0    0
2    0    0    0    0
3    0    0    0    0
4    0    0    0    0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   UDI                      10000 non-null  int64
 1   Product ID               10000 non-null  object
 2   Type                     10000 non-null  object
 3   Air temperature [K]      10000 non-null  float64
 4   Process temperature [K]  10000 non-null  float64
 5   Rotational speed [rpm]   10000 non-null  int64
 6   Torque [Nm]              10000 non-null  float64
 7   Tool wear [min]          10000 non-null  int64
 8   Machine failure          10000 non-null  int64
 9   TWF                      10000 non-null  int64
 10  HDF                      10000 non-null  int64
 11  PWF                      10000 non-null  int64
 12  OSF                      10000 non-null  int64
 13  RNF                      10000 non-null  int64
dtypes: float64(3), int64(9), object(2)
memory usage: 1.1+ MB
None
Preprocessing Complete! Encoded + Scaled shape: (10000, 10014)
Positive: 339 Negative: 9661 scale_pos_weight: 28.49852507374631
TimeSeriesSplit Ready!
```

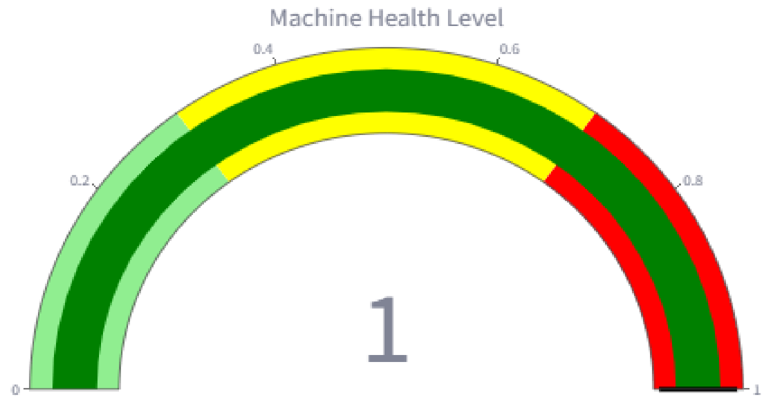
## 4. Dashboard Development

- Design the dashboard with real end-users in mind — typically maintenance engineers.
- Main dashboard should include:
  - Total machine count, failure count, failure rate
  - High-risk machine list
  - Machine selector (dropdown)
- For each selected machine, show:
  - Historical sensor readings with failure markers
  - Predicted risk score (numeric or gauge indicator)
  - SHAP explanation for contributing factors
- Add interactive elements:
  - Zoomable sensor charts
  - Hover tooltips
  - Dynamic risk trend plots
- Ensure the UI is responsive and guides the user quickly to "at-risk" assets.

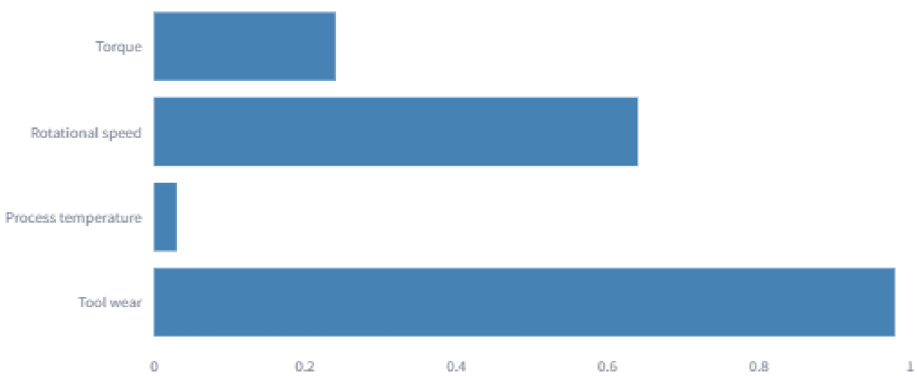


**Failure Status: No Failure** **Risk Score: 1.00 (High Risk)**

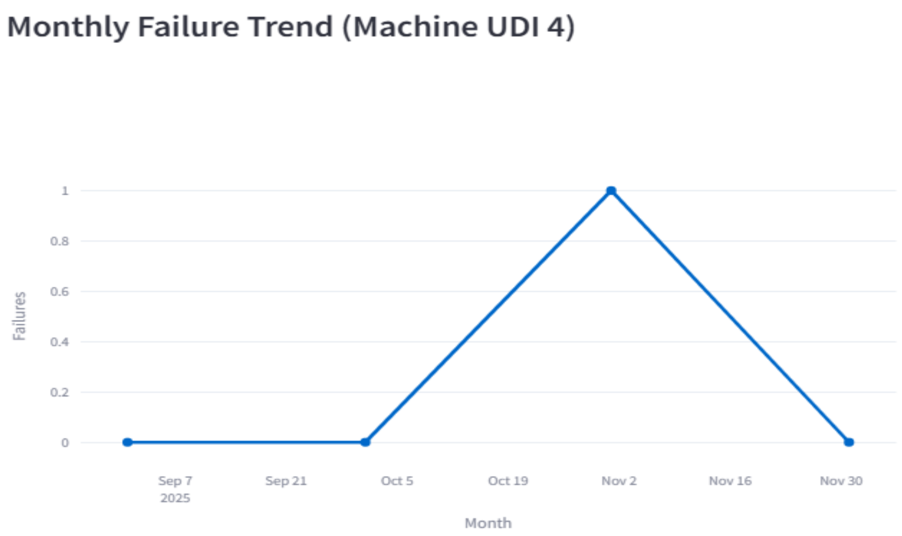
### Machine Risk Gauge

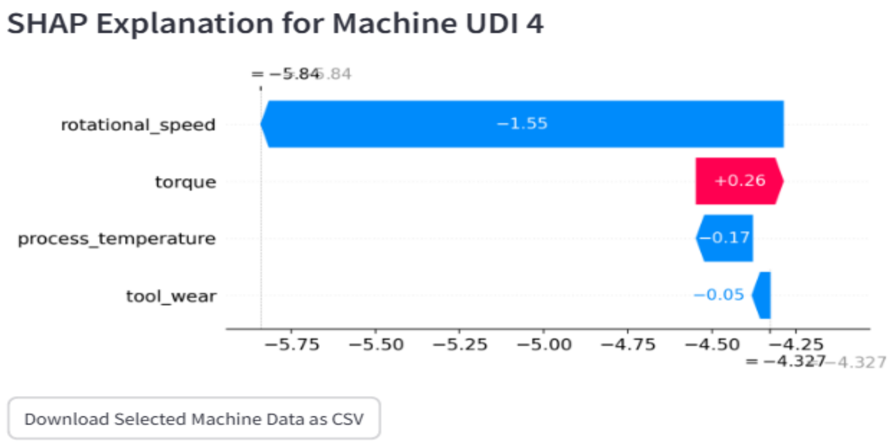



### Important Factors Influencing Risk




| Tool wear | Process temperature | Rotational speed | Torque |
|---|---|---|---|
| 0.98 | 0.03 | 0.64 | 0.24 |




## 5. Project Structure & Reproducibility

- Organize the project clearly to ensure repeatability:
    - Separate directories for raw data, processed data, notebooks, scripts, and dashboards.
- Maintain a requirements.txt file to fix library versions.
- Use consistent random seeds for reproducibility in all notebooks and model scripts.
- Maintain a clear folder for experiment logs, evaluation metrics, and model artifacts.
- Export the final trained model and ensure the dashboard loads it correctly each time.

```
altair==5.5.0
annotated-doc==0.0.4
anyio==4.12.0
blinker==1.9.0
cachetools==5.5.2
cloudpickle==3.1.2
contourpy==1.3.1
cycler==0.12.1
fastapi==0.123.0
fonttools==4.55.3
gitdb==4.0.12
GitPython==3.1.44
h11==0.16.0
Jinja2==3.1.6
joblib==1.4.2
jsonpatch==1.32
jsonpointer==2.1
jsonschema==4.19.0
kiwisolver==1.4.7
llvmlite==0.46.0
MarkupSafe==3.0.2
matplotlib==3.10.0
narwhals==1.31.0
nbformat==5.8.0
numba==0.63.1
numpy==2.2.1
pandas==2.2.3
plotly==6.5.0
protobuf==5.29.4
PyAudio==0.2.14
pycryptodome==3.23.0
pydeck==0.9.1
Pygments==2.16.1
pyotp==2.9.0
pyparsing==3.2.0
PyQt5==5.15.10
PyQtWebEngine==5.15.6
pywin32==305.1
PyYAML==6.0
scikit-learn==1.6.0
scipy==1.14.1
setuptools==75.1.0
shap==0.50.0
slicer==0.0.8
SpeechRecognition==3.14.
starlette==0.50.0
streamlit==1.43.2
tenacity==9.0.0
threadpoolctl==3.5.0
toml==0.10.2
tornado==6.4.2
traitlets==5.10.1
tzdata==2024.2
ujson==5.8.0
uvicorn==0.38.0
watchdog==6.0.0
wheel==0.44.0
xgboost==3.1.2
zstandard==0.19.0
```

## Random Seed Initialization

-To ensure that all experiments in the project produce consistent and repeatable results, random seeds are initialized at the beginning of the workflow.
-Machine learning pipelines rely on randomness in several stages:
-Train/validation splitting
-Model weight initialization
-Feature shuffling
-Sampling
-Any NumPy or Python-based random operations
-If the random seed is not fixed, you will get different results every time, which makes debugging and comparing models impossible.

## Code Used for Random Seed Initialization

```
import random
import numpy as np

SEED = 42
random.seed(SEED)
np.random.seed(SEED)
```