---

# 📝 Notebook Summary: Step 1 – Data Cleaning & Preparation

This notebook performs the foundational data preprocessing required to enable predictive modeling for smart farming machinery.

---

## 📌 Objectives

- Load high-frequency synthetic sensor data
- Parse timestamps and standardize columns
- Validate data integrity (missing values, shape)
- Explore key categorical variables (e.g., operating mode)
- Save a cleaned dataset for further processing

---

## 🔍 Section-by-Section Overview

### 1. **Import & Setup**

- Loaded essential libraries ( `pandas` , `numpy` , `matplotlib` , `seaborn` )
- Used `Pathlib` for OS-agnostic file management
- Defined `RAW_DATA_PATH` and `PROCESSED_DATA_PATH` for I/O separation

### 2. **Data Loading**

- Loaded the raw CSV file and explicitly renamed all columns to meaningful headers
- Ensures clarity and consistent downstream processing

### 3. **Initial Inspection**

- Displayed `df.head()` and dataset shape to verify successful loading
- Confirmed that the structure matches expectations

### 4. **Timestamp Parsing**

- Converted the `timestamp` column to `datetime` format using `pd.to_datetime`
- Enables time-series operations, rolling features, and sorting

### 5. **Missing Values**

- Checked for nulls across all columns using `df.isnull().sum()`
- No missing data was found, so no imputation was necessary

### 6. **Summary Statistics**

- `df.describe()` helped detect outliers and confirmed valid sensor ranges
- Highlights important distributions (e.g., motor current range, torque variability)

### 7. **Operating Mode Check**

- Explored unique categories in `operating_mode`
- Ensures correct categorization before encoding in later stages

### 8. **Sorting and Saving**

- Sorted data by `machine_id` and `timestamp` to preserve time-series continuity
- Exported cleaned data to `data/processed/agri_sensor_data_cleaned.csv`

---

## 📈 Recommendations

- Add a few exploratory time-series plots (e.g., `torque`, `vibration_level` over time)
- Log the number of unique timestamps per machine to detect duplicates
- Summarize row counts per operating mode or machine

---

## 🔗 Link to Next Notebook

The cleaned dataset is now ready for `02_feature_engineering.ipynb`, where we will:

- Generate rolling features (mean, std, etc.)
- One-hot encode operational modes
- Prepare the final feature matrix `X` for modeling

---

✅ **Status**: Data successfully cleaned and exported. Ready for feature engineering.

In [1]:
```
# 📦 Import Required Libraries
# These libraries are used for data handling, visualization, and path management.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
```

In [2]:
```
# 📂 Define File Paths
# Setting up paths to load raw data and later save processed outputs.

RAW_DATA_PATH = Path("../data/raw/agri_synthetic_sensor_data.csv")
PROCESSED_DATA_PATH = Path("../data/processed/agri_sensor_data_cleaned.csv")
```

In [3]:
```
# 🚛 Load Raw Sensor Data
# The dataset includes sensor logs from agricultural machinery.
# Column headers are defined explicitly to replace default unnamed headers.

column_names = ['timestamp', 'machine_id', 'vibration_level', 'motor_current',
                'ambient_temp', 'motor_temp', 'torque', 'rpm', 'operating_mode',
                'failure_label', 'remaining_minutes']

df = pd.read_csv(RAW_DATA_PATH)
df.columns = column_names
```

In [4]:
```
# 🔍 Quick Dataset Preview
# Check the structure and initial rows to verify import success.

print("✅ Raw dataset loaded. Shape:", df.shape)
print(df.head())
```

```
✅ Raw dataset loaded. Shape: (32400, 11)
            timestamp  machine_id  vibration_level  motor_current  \
0  2024-06-01 08:00:00           1         0.553126      53.947485
1  2024-06-01 08:00:01           1         0.332621      52.479691
2  2024-06-01 08:00:02           1         0.309272      22.995413
3  2024-06-01 08:00:03           1         0.320388      47.664084
4  2024-06-01 08:00:04           1         0.544413      19.224424

   ambient_temp  motor_temp      torque          rpm operating_mode  \
0     23.379140   45.343130  120.754989  2105.975012          idle
1     26.513821   46.432882  242.454425  2350.921787     hydraulic
2     21.509210   44.396522  116.864833  2296.847784      traction
3     25.270313   52.550659  251.358248  1621.819025      traction
4     25.955789   53.209899  110.452437  1969.956101          idle

   failure_label  remaining_minutes
0              0                559
1              0                854
2              0                681
3              0                968
4              0                677
```

In [5]:
```
# ⏱️ Parse Timestamps
# Convert 'timestamp' column from string to datetime format for time-based operations.

df['timestamp'] = pd.to_datetime(df['timestamp'])
```

In [6]:
```
# 🧼 Check for Missing Values
# Ensure data completeness before further processing.

missing = df.isnull().sum()
```

```
print("Missing values per column:")
print(missing[missing > 0])
```

```
Missing values per column:
Series([], dtype: int64)
```

In [7]:
```
# 📊 Summary Statistics
# Get statistical overview of the dataset: mean, std, min, max for each numerical column.

print(df.describe())
```

```
         machine_id  vibration_level  motor_current  ambient_temp  \
count  32400.000000     32400.000000   32400.000000  32400.000000
mean       2.000000         0.349868      40.143722     24.002885
std        0.816509         0.120284      14.904560      2.996791
min        1.000000         0.000000       5.000000     12.002003
25%        1.000000         0.267829      30.019147     21.991123
50%        2.000000         0.350388      40.085331     24.014464
75%        3.000000         0.431970      50.162418     26.004566
max        3.000000         0.799446      97.446733     36.606078

         motor_temp        torque           rpm  failure_label  \
count  32400.000000  32400.000000  32400.000000   32400.000000
mean      44.011202    220.081366   1848.814807       0.001852
std        4.969155     59.342229    250.051818       0.042994
min       24.391863     80.000000    849.100487       0.000000
25%       40.683464    179.503812   1681.250720       0.000000
50%       44.010620    219.592498   1851.612368       0.000000
75%       47.368328    260.134702   2017.884557       0.000000
max       64.962635    450.000000   2795.546962       1.000000

       remaining_minutes
count       32400.000000
mean          749.479691
std           261.689933
min             1.000000
25%           525.000000
50%           751.000000
75%           976.000000
max          1199.000000
```

In [8]:
```
# 🚜 Unique Operating Modes
# Explore which operational states are logged in the data.

print("Operating Modes:", df['operating_mode'].unique())
```

```
Operating Modes: ['idle' 'hydraulic' 'traction' 'pto']
```

In [9]:
```
# 🗂️ Sort and Save
# Sort the data by machine and time to prepare for time-series modeling.

df = df.sort_values(by=['machine_id', 'timestamp'])
df.to_csv(PROCESSED_DATA_PATH, index=False)
print(f"💾 Cleaned dataset saved to: {PROCESSED_DATA_PATH}")
```

```
💾 Cleaned dataset saved to: ..\data\processed\agri_sensor_data_cleaned.csv
```