1. Rolling statistics/quantities

Past-window aggregates (mean/std/min/max/percentiles) over several horizons (e.g., 7/30/90 days).

Captures level & variability trends without using the label; quantiles are robust to rare spikes and class imbalance.

On segment-time series (e.g., Acc_*_RMS, Twist, Speed) compute 7/30/90-day stats **excluding the current timestamp** to avoid leakage.

Pros

- Simple, fast, strong baseline
- Quantiles resist outliers (common in rare-event data)

Cons

- Window choice is manual
- May dilute short bursts if windows are too long

Sample :

```
WITH base AS (
  SELECT
    BaseCode AS seg,
    CAST(RecordingDate AS timestamp) AS ts,
    Acc_Vert_RMS, Acc_Lat_RMS, Twist, Speed
  FROM predictive_maintenance.wagondata
)
SELECT
  seg, ts,
  -- 7-day rolling mean/std (past only)
  avg(Acc_Vert_RMS) OVER w7  AS accv_mean_7d,
  stddev_pop(Acc_Vert_RMS) OVER w7 AS accv_std_7d,
  -- 30-day 95th percentile
  percentile_cont(0.95) WITHIN GROUP (ORDER BY Acc_Vert_RMS) OVER w30 AS
accv_p95_30d,
  avg(Twist) OVER w30 AS twist_mean_30d
FROM base
WINDOW
  w7  AS (PARTITION BY seg ORDER BY ts
          RANGE BETWEEN INTERVAL 7 DAYS  PRECEDING AND INTERVAL 1 SECOND
PRECEDING),
  w30 AS (PARTITION BY seg ORDER BY ts
```

```
         RANGE BETWEEN INTERVAL 30 DAYS PRECEDING AND INTERVAL 1 SECOND
PRECEDING);
```

2. Lags & differences

Lagged values, first differences, percent changes, linear trend (windowed slope).

Rare failures often follow **acceleration of degradation**; deltas & slopes reveal momentum.

Per segment, build lag_1/7/30, diff_1, and 30-day slope.

Pros

- Highlights deterioration speed
- Low risk of leakage if you only use past lags

Cons

- Sensitive to missingness/irregular sampling

Sample :

```
from pyspark.sql import functions as F, Window as W

w = W.partitionBy("seg").orderBy("ts")

df2 = df \
  .withColumn("accv_lag1", F.lag("Acc_Vert_RMS", 1).over(w)) \
  .withColumn("accv_diff1", F.col("Acc_Vert_RMS") - F.col("accv_lag1")) \
  .withColumn("t_unix", F.col("ts").cast("long")) \
  # rolling slope over last 30 records (replace with time-based in SQL if needed)
  .withColumn("accv_slope_30",
     F.regr_slope(F.col("Acc_Vert_RMS"), F.col("t_unix")).over(w.rowsBetween(-30, -1)))
# NOTE: use SQL time windows when you need "last 30 days" instead of "last 30 rows".
```

3. Recency-weighted features (EWMA / EWMVar)

Exponentially-weighted moving averages/variances, so the recent past dominates.

In rare-event prediction, very recent anomalies matter more than old ones.

Build EWMA/EWMVar for key sensors ($\alpha \in [0.05, 0.3]$) **per segment**.

Pros

- Emphasizes fresh signals
- One parameter (α) to tune

Cons

- Requires custom logic in Spark (no built-in EWM in SQL)

Sample :

```
import pandas as pd
from pyspark.sql import functions as F, Window as W
from pyspark.sql.types import StructType, StructField, TimestampType,
StringType, DoubleType

@F.pandas_udf("seg string, ts timestamp, accv_ewma double, accv_ewmvar
double")
def ewm_udf(pdf: pd.DataFrame) -> pd.DataFrame:
    # Input is sorted per partition (seg)
    pdf = pdf.sort_values("ts")
    x = pdf["Acc_Vert_RMS"].astype(float)
    # Exponentially weighted mean/var
    ewma = x.ewm(alpha=0.2, adjust=False).mean()
    ewmvar = x.ewm(alpha=0.2, adjust=False).var(bias=False)
    return pd.DataFrame({"seg": pdf["seg"], "ts": pdf["ts"],
                         "accv_ewma": ewma, "accv_ewmvar": ewmvar})

df_ew = df.groupBy("seg").applyInPandas(ewm_udf)
# NOTE: This keeps causality since it scans forward in time per segment.
```

4. Eventization for rare patterns: counts & time-since-last

Convert continuous signals into **event flags** (e.g., above segment-level P95/P99) and engineer:

- event count / share in last 7/30/90 days
- Time-since-last-event

For imbalanced targets, “how often and how recently things went bad” is highly predictive.
Define flags for Acc_*_RMS, Twist, LFI = $Speed^2 \times |Curvature|$, etc.

Pros

- Directly addresses rarity and recency
- Robust to scale/unit issues

Cons

- Needs good thresholds (use robust, per-segment quantiles)

Sample :

```
WITH b AS (
  SELECT
    BaseCode AS seg,
    CAST(RecordingDate AS date) AS d,
    Acc_Vert_RMS,
    -- Event if value exceeds per-segment 95th percentile (computed
historically)
    CASE WHEN Acc_Vert_RMS >
         PERCENTILE(Acc_Vert_RMS, 0.95) OVER (PARTITION BY BaseCode)
    THEN 1 ELSE 0 END AS ev
  FROM predictive_maintenance.wagondata
),
daily AS (
  SELECT seg, d, SUM(ev) AS ev_cnt, COUNT(*) AS n
  FROM b GROUP BY 1,2
)
SELECT
  seg, d,
  -- Past-30-day event count (exclude today)
  SUM(ev_cnt) OVER (PARTITION BY seg ORDER BY d
    RANGE BETWEEN INTERVAL 30 DAYS PRECEDING AND INTERVAL 1 DAY
PRECEDING) AS ev_cnt_30d,
  -- Time since last event in days
  DATEDIFF(d, MAX(CASE WHEN ev_cnt>0 THEN d END)
            OVER (PARTITION BY seg ORDER BY d
                  RANGE BETWEEN INTERVAL 365 DAYS PRECEDING AND INTERVAL
1 DAY PRECEDING))
    AS days_since_last_ev
FROM daily;
```

5. Exposure & missingness features

Track **how much data** exists in a window: number of observations, share of missing, sensor uptime.

Imbalanced datasets often have **uneven coverage**; low exposure can confound the model (false “safe” sections).

Apply:
For every window, add obs_count, %missing, %imputed.

Pros

- Improves reliability and calibration
- Helps the model “know what it doesn’t know”

Cons

- Correlated with operating schedules (interpret carefully)

Sample :

```
from pyspark.sql import functions as F, Window as W
w = W.partitionBy("seg").orderBy("ts").rowsBetween(-1000, -1)  # last
~1000 rows as a proxy window

df_exp = df \
  .withColumn("obs_cnt_win", F.count(F.lit(1)).over(w)) \
  .withColumn("miss_accv_win",
              F.avg(F.when(F.col("Acc_Vert_RMS").isNull(),
1).otherwise(0)).over(w)) \
  .withColumn("miss_twist_win",
              F.avg(F.when(F.col("Twist").isNull(),
1).otherwise(0)).over(w))
# NOTE: Replace row-based windows with time-based windows in SQL when
required.
```

6. Fourier seasonal terms

Encode seasonality with sine/cosine of day-of-year / week-of-year; month dummies if needed.

Temperature cycles (thermal stress) can correlate with breaks; cyclical encoding avoids artificial jumps at year boundaries.

Add doy_sin, doy_cos, woy_sin, woy_cos.

Pros

- Cheap signal; works even without explicit weather data

- Smooth, differentiable (good for linear/NN models)

Cons

- Coarse proxy; add real weather later if available

Sample :

```
SELECT
  seg, ts,
  SIN(2*PI() * dayofyear(ts)/365.0) AS doy_sin,
  COS(2*PI() * dayofyear(ts)/365.0) AS doy_cos,
  SIN(2*PI() * weekofyear(ts)/52.0) AS woy_sin,
  COS(2*PI() * weekofyear(ts)/52.0) AS woy_cos
FROM your_time_series;
```

7. Target encoding (leakage-safe)

For each segment (or segment×line), compute the **historical event rate up to t-1** and use it as a prior/propensity feature.

Imbalanced labels benefit from a **smoothed prior**; expanding mean respects time and reduces variance on rare positives.

Join `trainingcontext` (labels) and compute expanding `avg(target)` **excluding current**; use Bayesian smoothing if data is sparse.

Pros

- Strong priority for rare events
- Encodes persistent "hotness" of a segment

Cons

- Must guard against leakage strictly
- Sparse segments need smoothing

Sample :

```
WITH y AS (
  SELECT BaseCode AS seg, CAST(r_date AS timestamp) AS ts, target
  FROM predictive_maintenance.trainingcontext
),
feat AS (
```

```
  SELECT
    seg, ts,
    -- Expanding mean up to t-1
    AVG(target) OVER (
      PARTITION BY seg ORDER BY ts
      ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING
    ) AS seg_prior_rate
  FROM y
)
SELECT * FROM feat;
-- NOTE: For smoothing: seg_prior_smoothed = (sum(y) + a) / (count + a +
b)
```

Reference

Box, GEP 2016, *Time series analysis : forecasting and control*, Fifth edition., Wiley, Hoboken, New Jersey.

Bergmeir, C & Benítez, JM 2012, 'On the use of cross-validation for time series predictor evaluation', *Information Sciences*, vol. 191, pp. 192–213.

Coles, S 2001, *An Introduction to Statistical Modeling of Extreme Values*, 1st ed. 2001., Springer London, London.

Che, Z, Purushotham, S, Cho, K, Sontag, D & Liu, Y 2018, 'Recurrent Neural Networks for Multivariate Time Series with Missing Values', *Scientific Reports*, 6085, vol. 8, no. 1.

Jones, LA, Champ, CW & Rigdon, SE 2001, 'The Performance of Exponentially Weighted Moving Average Charts With Estimated Parameters', *Technometrics*, vol. 43, no. 2, pp. 156–167.