Time Series Anomaly Detection — Summary

Problem understanding and approach

This project aims to detect anomalies in time-series sensor data from bearings. The operational goal is early identification of abnormal behavior that could indicate mechanical faults, enabling preventive maintenance and reduced downtime. The dataset contains multivariate sensor readings (accelerometer/vibration channels and possibly temperature or other signals) collected over time. The project follows a standard pipeline: data loading and cleaning, feature engineering, model training (unsupervised anomaly detection), evaluation, and reporting.

Key constraints and goals: - Work with a historical dataset where anomalies are rare and labels may be limited. - Prefer unsupervised or semi-supervised approaches that can learn normal behavior and flag deviations. - Provide interpretable outputs (anomaly scores and visualizations) that can be acted upon by maintenance teams.

Approach summary: - Data ingestion: consistent parsing and timestamp handling to build time-indexed series. - Feature extraction: generate statistical and domain-inspired features over sliding windows to summarize local behavior. - Models: compare an Isolation Forest (tree-based unsupervised method) against a sequence reconstruction approach (LSTM autoencoder) to capture both instantaneous outliers and temporal-context anomalies. - Evaluation: use available labeled points or surrogate metrics (ROC, confusion matrix on held-out labeled tests) and inspect model outputs visually (time-series anomaly overlays, ROC/PR curves, confusion matrices).

Feature engineering rationale

Rationale: - Raw sensor streams are high-frequency and noisy; models benefit from compact, informative features computed over short windows. - Combining time-domain statistics (mean, std, skewness, kurtosis), peak-related measures (max/min, RMS), and frequency-domain summaries (dominant frequency, spectral energy) captures both amplitude and periodicity changes that signal bearing wear or faults.

Common features used in the codebase: - Windowed statistics: mean, median, standard deviation, variance, min, max, interquartile range. - Higher-order moments: skewness, kurtosis to detect distribution shifts. - Energy measures: RMS and total spectral energy from short-time FFTs. - Signal peaks and envelope metrics: peak-to-peak amplitude, crest factor. - Rolling correlations or cross-channel features to detect synchronization changes between sensors.

Why these help: - Early faults often show as subtle shifts in variance, occasional spikes, or emergent periodic signatures; windowed statistics and spectral features capture these. - Aggregating over windows reduces model sensitivity to isolated sensor noise while retaining event-level anomalies.

Implementation notes from repository: - Feature extraction is performed in features/feature_engineering.py. The script computes a set of windowed statistics and persists transformed data for model training. Visual checks (feature distributions and correlation matrix) are available in visualizations/feature_distributions.png and visualizations/correlation_matrix.png to guide feature selection.

Model selection and comparison

Models implemented: - Isolation Forest (models/isolation_forest_model.py): fast, tree-based unsupervised detector that isolates anomalies via random partitioning. Strengths: scalable, interpretable anomaly scores, few hyperparameters. Weaknesses: limited temporal modeling; sensitive to feature scaling. - LSTM Autoencoder (models/lstm_autoencoder.py): sequence model that learns to reconstruct normal sequences; high reconstruction error indicates anomalies. Strengths: captures temporal dependencies and gradual degradation patterns.

Weaknesses: requires more data, careful training, and is more computationally intensive.

Comparison criteria: - Detection capability: Isolation Forest detects pointwise or feature-space outliers; LSTM AE better captures contextual/temporal anomalies. - Interpretability: Isolation Forest provides per-feature contribution insights via tree paths; autoencoder outputs reconstruction error per time-step
which can be visualized but is less directly attributable to a single feature. - Operational factors: Isolation Forest trains faster and is easier to deploy on edge devices; LSTM AE may be used centrally or in cloud for richer temporal modeling.

Evaluation results (repo artifacts): - Saved results in results/ show multiple metrics JSON files (e.g., metrics_20251210_063814.json) and prediction arrays. Visual results include ROC curves and confusion matrices for both models (visualizations/iso_forest_roc.png, visualizations/lstm_ae_roc.png, visualizations/iso_forest_cm.png, visualizations/lstm_ae_cm.png). - From these artifacts, the models appear to have been evaluated across several runs and thresholds; choose the best operating point based on the desired trade-off between precision and recall for your maintenance policy.

Practical recommendation: - Use Isolation Forest for a lightweight baseline and quick alerts. Add LSTM Autoencoder for a second-stage filter or for scenarios where sequence context matters (e.g., slow-developing faults).

Key findings and business insights

Technical findings: - Feature engineering matters: models trained on aggregated window statistics and spectral features show improved ROC and more stable predictions versus using raw sensor channels directly (see visualizations/feature_distributions.png). - Complementary models improve coverage: Isolation Forest catches sharp, high-magnitude anomalies; LSTM AE detects sequences with elevated reconstruction error before catastrophic spikes. - Threshold selection is business-critical: the ROC curves demonstrate a trade-off—tuning for high recall will increase false positives and maintenance cost; tuning for high precision risks missing early warnings.

Business insights: - Early detection reduces unplanned downtime: flagging anomalies even a few cycles earlier enables scheduled maintenance and part replacement planning. - Tunable risk profiles: create alarm tiers (informational, investigate, immediate action) based on anomaly score thresholds or consensus between models to balance operational cost and safety. - Root cause triage: augment anomaly alerts with top contributing features and recent trends (e.g., rising variance or emergent spectral peaks) to prioritize investigations.

Operational suggestions: - Deploy the Isolation Forest as a fast, continuous monitor on device or near-edge. Periodically (daily/weekly) run an LSTM AE centrally for aggregated sequence analysis. - Implement alert escalation rules: require persistent anomaly scores over multiple windows before triggering high-priority maintenance tickets.

Limitations and future improvements

Limitations: - Label scarcity: supervised evaluation is limited if few labeled fault events are available; metrics may be optimistic if labels are noisy. - Domain shift: models trained on historical data may degrade if operating conditions (speed, load, sensor placement) change. - Interpretability: while feature-based methods are fairly transparent, deep sequence models need additional tooling (SHAP, LIME for time-series, or attention mechanisms) for per-decision explainability.

Short-term improvements: - Improve labeling: collect and curate more labeled failure events, or use semi-supervised methods and human-in-the-loop verification to expand positive examples. - Threshold tuning and calibration: implement an operational calibration routine that adapts thresholds based on recent false positive/negative rates. - Ensemble strategy: combine p-value style calibration from Isolation Forest with LSTM reconstruction error via a calibrated meta-scoring function to reduce false alarms.

Long-term directions: - Domain adaptation: add transfer learning or online updating to handle new machines or changed operating regimes without full retraining. - Explainability: integrate attribution tools to show which features or time-steps contributed most to a given anomaly score. - Real-time pipeline: build a streaming ingestion and scoring pipeline (Kafka/Redis + lightweight scoring service) with windowed feature computation for low-latency detection.

Artifacts and where to look in this repo

- Data loading and preprocessor: `data/load_data.py`, `data/preprocessor.py`
- Feature engineering: `features/feature_engineering.py`
- Models: `models/isolation_forest_model.py`, `models/lstm_autoencoder.py`
- Evaluation: `evaluation/evaluator.py`, `results/` (metrics and predictions)
- Visualizations: files in `visualizations/` such as `iso_forest_roc.png`, `lstm_ae_roc.png`, `feature_distributions.png`, and anomaly overlays.

Next steps and recommendations

1. Validate thresholds with operations: run an A/B test or pilot with the current Isolation Forest alarms and collect feedback to refine thresholds.

2. Add model monitoring: track concept drift metrics and alert when model performance degrades.

3. Expand labeled dataset and retrain LSTM AE with more representative sequences.

4. Add per-alert explanations (top features) to help technicians triage faster.