# Pynomaly Process Guide

## Document Information

**Title:** Pynomaly Process Guide

**Generated:** 2025-06-24 16:37:41

**Version:** 1.0

**Project:** Pynomaly - State-of-the-Art Anomaly Detection Platform

# Pynomaly Process Guide¶

## Overview¶

Pynomaly is a state-of-the-art Python anomaly detection package that provides a unified interface for multiple anomaly detection libraries. This comprehensive guide covers the complete process of using Pynomaly for anomaly detection, from initial setup through production deployment.

## Table of Contents¶

1. Installation and Setup
2. Data Preparation
3. Algorithm Selection
4. Model Training
5. Anomaly Detection
6. Results Analysis
7. Model Evaluation
8. Production Deployment
9. Monitoring and Maintenance

## Installation and Setup¶

### Prerequisites¶

- Python 3.11 or higher
- Minimum 4GB RAM (8GB+ recommended for large datasets)
- GPU support optional but recommended for deep learning models

## Basic Installation¶

```
# Using pip
pip install pynomaly

# Using Poetry (recommended for development)
poetry add pynomaly

# From source
git clone https://github.com/your-org/pynomaly.git
cd pynomaly
poetry install
```

## Environment Configuration¶

```
# Create configuration directory
mkdir ~/.pynomaly

# Set up environment variables
export PYNOMALY_CONFIG_PATH=~/.pynomaly
export PYNOMALY_STORAGE_PATH=~/.pynomaly/storage
export PYNOMALY_LOG_LEVEL=INFO
```

### Verify Installation¶

```
# Check CLI installation
pynomaly --version

# Run basic health check
pynomaly status

# Test installation
python -c "import pynomaly; print('Installation successful')"
```

# Data Preparation¶

## Supported Data Formats¶

Pynomaly supports multiple data formats:

- **CSV**: Comma-separated values
- **Parquet**: Columnar storage format
- **Arrow**: In-memory columnar format
- **JSON**: JavaScript Object Notation
- **Excel**: Microsoft Excel files
- **Database**: SQL databases via SQLAlchemy

## Data Loading Process¶

### 1. Load Data from File¶

```
# CLI approach
pynomaly dataset load data.csv --name production_data --format csv

# Python SDK approach
from pynomaly import PynomlalyClient
```

```
client = PynomlalyClient()
dataset = client.load_dataset(
    file_path="data.csv",
    name="production_data",
    format="csv"
)
```

### 2. Data Validation¶

The system automatically validates: - Data types and schema - Missing values - Outliers and anomalies - Feature distributions - Data quality metrics

### 3. Data Preprocessing¶

Available preprocessing options: - **Normalization**: Min-max, Z-score, robust scaling - **Encoding**: One-hot, label, target encoding - **Feature Engineering**: Polynomial features, interactions - **Missing Value Handling**: Imputation, deletion, flagging

```
# Configure preprocessing pipeline
preprocessing_config = {
    "scaling": "standard",
    "encoding": "onehot",
    "missing_strategy": "median",
    "feature_selection": "variance_threshold"
}

preprocessed_dataset = client.preprocess_dataset(
    dataset_id=dataset.id,
    config=preprocessing_config
)
```

# Algorithm Selection¶

## Available Algorithm Categories¶

### 1. Statistical Methods¶

- **Isolation Forest**: Tree-based anomaly detection
- **Local Outlier Factor (LOF)**: Density-based detection
- **One-Class SVM**: Support vector machine approach
- **Elliptic Envelope**: Gaussian distribution assumption

### 2. Machine Learning Methods¶

- **AutoEncoder**: Neural network-based reconstruction
- **LSTM**: Long Short-Term Memory for time series
- **Random Forest**: Ensemble tree method
- **Gradient Boosting**: Boosted tree ensemble

### 3. Deep Learning Methods¶

- **Variational AutoEncoder (VAE)**: Probabilistic encoding
- **Generative Adversarial Networks (GAN)**: Adversarial training
- **Transformer**: Attention-based models
- **Convolutional Neural Networks (CNN)**: For image/spatial data

### 4. Specialized Methods¶

- **Graph Neural Networks**: For graph-structured data
- **Time Series Specific**: ARIMA, Prophet, seasonal decomposition
- **Text Analysis**: TF-IDF, word embeddings, transformers

## Algorithm Selection Process¶

### 1. Automated Selection (Recommended)¶

```
# Use autonomous mode for automatic selection
pynomaly auto detect --dataset production_data --target-metric f1
```

### 2. Manual Selection¶

```
# Create detector with specific algorithm
detector = client.create_detector(
    name="production_detector",
    algorithm="IsolationForest",
    parameters={
        "n_estimators": 100,
        "contamination": 0.1,
        "random_state": 42
    }
)
```

### 3. Ensemble Methods¶

```
# Create ensemble detector
ensemble_detector = client.create_ensemble_detector(
    algorithms=["IsolationForest", "LOF", "OneClassSVM"],
    voting_strategy="soft",
    weights=[0.4, 0.3, 0.3]
)
```

# Model Training¶

## Training Process Overview¶

1. **Data Splitting**: Train/validation/test splits
2. **Hyperparameter Tuning**: Grid search, random search, Bayesian optimization
3. **Cross-Validation**: Time-series aware splitting
4. **Model Validation**: Performance metrics evaluation

## Training Execution¶

### CLI Training¶

```
# Basic training
pynomaly detect train --detector production_detector --dataset production_data

# Advanced training with hyperparameter tuning
pynomaly detect train \
    --detector production_detector \
    --dataset production_data \
    --tune-hyperparameters \
    --cv-folds 5 \
    --optimization-metric f1
```

### Python SDK Training¶

```
# Configure training parameters
training_config = {
    "validation_split": 0.2,
    "cross_validation": {
        "folds": 5,
        "strategy": "time_series"
    },
    "hyperparameter_tuning": {
        "method": "bayesian",
```

```
        "n_trials": 100,
        "optimization_metric": "f1_score"
    }
}

# Execute training
training_result = client.train_detector(
    detector_id=detector.id,
    dataset_id=dataset.id,
    config=training_config
)
```

### Training Monitoring¶

The system provides real-time training monitoring: - Loss curves and metrics - Hyperparameter optimization progress - Resource utilization (CPU, GPU, memory) - Estimated completion time

## Anomaly Detection¶

### Detection Modes¶

#### 1. Batch Detection¶

Process entire datasets at once:

```
# CLI batch detection
pynomaly detect run \
    --detector production_detector \
    --dataset test_data \
    --output results.json
```

```
# SDK batch detection
results = client.detect_anomalies(
```

```
    detector_id=detector.id,
    dataset_id=test_dataset.id,
    threshold=0.5
)
```

### 2. Streaming Detection¶

Real-time anomaly detection:

```
# Set up streaming detection
stream_config = {
    "input_source": "kafka://localhost:9092/data_topic",
    "output_sink": "kafka://localhost:9092/anomaly_topic",
    "batch_size": 100,
    "max_latency_ms": 500
}

streaming_job = client.create_streaming_job(
    detector_id=detector.id,
    config=stream_config
)
```

### 3. API-based Detection¶

REST API for integration:

```
# Single prediction
curl -X POST "http://localhost:8000/api/v1/detect" \
  -H "Content-Type: application/json" \
  -d '{"detector_id": "det_123", "features": [1.2, 3.4, 5.6]}'
```

### Detection Parameters¶

Key parameters for detection: - **Threshold**: Anomaly score threshold (0.0-1.0) - **Contamination Rate**: Expected proportion of anomalies - **Confidence Level**: Statistical confidence for predictions - **Batch Size**: Number of samples to process at once

## Results Analysis¶

### Detection Results Structure¶

```
{
  "detection_id": "det_20240624_001",
  "timestamp": "2024-06-24T10:30:00Z",
  "detector_info": {
    "id": "production_detector",
    "algorithm": "IsolationForest",
    "version": "1.0.0"
  },
  "dataset_info": {
    "id": "test_data",
    "size": 10000,
    "features": 15
  },
  "results": {
    "total_samples": 10000,
    "anomalies_detected": 150,
    "anomaly_rate": 0.015,
    "processing_time_ms": 1250
  },
  "anomalies": [
    {
      "sample_id": "sample_123",
      "anomaly_score": 0.85,
      "confidence": 0.92,
      "features": {...},
      "explanation": {...}
    }
  ],
  "performance_metrics": {
    "precision": 0.89,
    "recall": 0.76,
```

```
    "f1_score": 0.82,
    "auc_roc": 0.91
  }
}
```

## Visualization and Reporting¶

### 1. Generate Visualizations¶

```
# Create visualization reports
pynomaly detect visualize \
    --results results.json \
    --output-dir ./reports \
    --format html
```

### 2. Export to Business Intelligence Tools¶

```
# Export to Excel
pynomaly export excel results.json anomaly_report.xlsx

# Export to Power BI
pynomaly export powerbi results.json --connection-string "..."

# Export to Tableau
pynomaly export tableau results.json --format tde
```

# Model Evaluation¶

## Evaluation Metrics¶

### Classification Metrics¶

- **Precision**: True positives / (True positives + False positives)
- **Recall**: True positives / (True positives + False negatives)
- **F1-Score**: Harmonic mean of precision and recall
- **AUC-ROC**: Area under the receiver operating characteristic curve

### Anomaly-Specific Metrics¶

- **Anomaly Score Distribution**: Distribution of anomaly scores
- **Threshold Sensitivity**: Performance across different thresholds
- **False Positive Rate**: Rate of incorrectly flagged normal samples
- **Detection Latency**: Time to detect anomalies

## Model Comparison¶

```
# Compare multiple models
comparison_result = client.compare_detectors(
    detector_ids=["det_1", "det_2", "det_3"],
    test_dataset_id="test_data",
    metrics=["precision", "recall", "f1_score", "processing_time"]
)
```

## Performance Benchmarking¶

```
# Run performance benchmarks
pynomaly benchmark \
    --detectors production_detector \
    --datasets test_data \
```

```
    --metrics accuracy,speed,memory \
    --output benchmark_report.json
```

# Production Deployment¶

## Deployment Options¶

### 1. Docker Deployment¶

```
# Build Docker image
docker build -t pynomaly-app .

# Run container
docker run -p 8000:8000 \
    -e PYNOMALY_CONFIG_PATH=/app/config \
    -v $(pwd)/config:/app/config \
    pynomaly-app
```

### 2. Kubernetes Deployment¶

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pynomaly-api
spec:
  replicas: 3
  selector:
    matchLabels:
      app: pynomaly-api
  template:
    metadata:
      labels:
        app: pynomaly-api
    spec:
      containers:
```

```
      - name: api
        image: pynomaly:latest
        ports:
        - containerPort: 8000
        env:
        - name: PYNOMALY_CONFIG_PATH
          value: "/app/config"
        resources:
          requests:
            memory: "2Gi"
            cpu: "1"
          limits:
            memory: "4Gi"
            cpu: "2"
```

### 3. Cloud Deployment¶

- **AWS**: ECS, Lambda, SageMaker
- **Azure**: Container Instances, Functions, Machine Learning
- **GCP**: Cloud Run, Cloud Functions, AI Platform

## Production Configuration¶

```
# production.yml
app:
  name: "Pynomaly Production"
  version: "1.0.0"
  debug: false

api:
  host: "0.0.0.0"
  port: 8000
  workers: 4
  max_request_size: "10MB"

security:
  auth_enabled: true
  api_key_required: true
  rate_limiting:
    requests_per_minute: 1000
    burst_size: 100
```

```
performance:
  connection_pool_size: 20
  query_timeout: 30
  cache_ttl: 3600

monitoring:
  telemetry_enabled: true
  metrics_endpoint: "/metrics"
  health_check_endpoint: "/health"
```

# Monitoring and Maintenance¶

## Health Monitoring¶

```
# Check system health
pynomaly status --detailed

# Monitor specific components
pynomaly monitor --component detector --id production_detector
```

## Performance Monitoring¶

Key metrics to monitor: - **Detection Latency**: Time to process requests - **Throughput**: Requests processed per second - **Resource Utilization**: CPU, memory, GPU usage - **Error Rates**: Failed requests and exceptions - **Model Performance**: Accuracy, drift detection

## Automated Maintenance¶

### 1. Model Retraining¶

```
# Set up automated retraining
retraining_schedule = {
```

```
    "frequency": "weekly",
    "trigger_conditions": {
        "performance_degradation": 0.05,
        "data_drift_threshold": 0.1
    },
    "notification_channels": ["email", "slack"]
}

client.schedule_retraining(
    detector_id=detector.id,
    schedule=retraining_schedule
)
```

### 2. Data Quality Monitoring¶

```
# Monitor data quality
quality_checks = {
    "missing_values_threshold": 0.05,
    "outlier_threshold": 0.1,
    "schema_validation": True,
    "drift_detection": True
}

client.setup_data_monitoring(
    dataset_id=dataset.id,
    checks=quality_checks
)
```

## Troubleshooting¶

### Common Issues and Solutions¶

1. **High Memory Usage**
2. Reduce batch size
3. Enable memory-efficient processing
4. Use streaming mode for large datasets
5. **Slow Detection Performance**

6. Enable GPU acceleration
7. Optimize hyperparameters
8. Use model compression techniques
9. **Poor Detection Accuracy**
10. Retrain with more recent data
11. Adjust contamination rate
12. Try different algorithms or ensembles
13. **API Timeout Errors**
14. Increase timeout settings
15. Implement request queuing
16. Scale horizontally with load balancing

# Best Practices¶

## Data Preparation¶

- Always validate data quality before training
- Use appropriate preprocessing for your data type
- Maintain consistent feature engineering across train/test

## Model Selection¶

- Start with simple algorithms before complex ones
- Use cross-validation for robust evaluation
- Consider ensemble methods for improved performance

## Production Deployment¶

- Implement comprehensive monitoring
- Use staging environments for testing
- Maintain model versioning and rollback capabilities
- Set up automated alerts for system failures

### Security Considerations¶

- Encrypt sensitive data at rest and in transit
- Implement proper authentication and authorization
- Regular security audits and vulnerability assessments
- Comply with data protection regulations (GDPR, CCPA)

## Conclusion¶

This process guide provides a comprehensive overview of using Pynomaly for anomaly detection. The system's modular architecture and extensive feature set enable organizations to implement robust anomaly detection solutions that scale from prototype to production.

For specific implementation details, refer to the technical documentation and API references. For support, consult the troubleshooting guide or contact the development team.