**Error Detection System Documentation**

**1. Project Structure**

Oliver/

├── data\_loading.py # Data import and preprocessing

├── error\_detection/

│ ├── synthetic\_errors.py # Error injection system

│ ├── error\_detection.py # ML detection algorithms

│ ├── imputation.py # Value imputation

│ ├── validation.py # Performance evaluation

│ ├── utils.py # Utility functions

│ └── config.py # Configuration parameters

└── plot\_code/

└── error\_plots.py # Visualization functions

**2. Module Responsibilities & Key Functions**

**2.1 Data Loading (data\_loading.py)**

**Primary Purpose**: Handle raw data import and initial preprocessing

1. load\_vst\_file(): Loads raw VST data with encoding handling
2. prepare\_data\_for\_error\_detection(): Prepares clean data for error injection
3. Ensures 15-minute intervals
4. Creates consistent DateTime index
5. Validates data quality

**2.2 Synthetic Error Generation (synthetic\_errors.py)**

**Primary Purpose**: Create training data with known errors

SyntheticErrorGenerator class:

* Manages error injection process
* Prevents error overlap
* Tracks ground truth

Key methods:

* inject\_spike\_errors(): Adds spike anomalies
* inject\_gap\_errors(): Creates missing data periods
* inject\_flatline\_errors(): Inserts constant value segments
* \_create\_ground\_truth(): Generates truth labels

**2.3 Configuration (config.py)**

**Primary Purpose**: Central configuration management

SYNTHETIC\_ERROR\_PARAMS = {

'spike': {

'frequency': 0.01,

'magnitude\_range': (2, 5)

},

'gap': {

'frequency': 0.01,

'duration\_range': (1, 24)

},

'flatline': {

'frequency': 0.01,

'duration\_range': (2, 48)

}

}

**2.4 Validation (validation.py)**

**Primary Purpose**: Split data and validate results

 create\_train\_test\_split(): Splits data into training/testing sets

 validate\_error\_injection(): Verifies error injection quality

 calculate\_metrics(): Computes performance metrics

**2.5 Visualization (error\_plots.py)**

**Primary Purpose**: Create analysis plots

 plot\_detected\_errors(): Visualize detected errors

 plot\_error\_distribution(): Show error patterns

 plot\_imputation\_results(): Compare original vs imputed

**3. Workflow Steps**

**Step 1: Data Preparation**

# Load and prepare data

clean\_data = prepare\_data\_for\_error\_detection(raw\_data\_path)

train\_data, test\_data = create\_train\_test\_split(clean\_data)

**Output: Clean DataFrame with consistent intervals**

**Step 2: Synthetic Error Generation**

# Generate synthetic errors

error\_generator = SyntheticErrorGenerator()

modified\_data, ground\_truth = error\_generator.inject\_all\_errors(train\_data)

**Output:**

 Modified data with known errors

 Ground truth labels

**Step 3: Error Detection**

# Detect errors using ML models

detector = error\_detection.SpikeDetector()

error\_flags = detector.detect(modified\_data)

**Output: DataFrame with error flags and confidence scores**

**Step 4: Validation**

# Validate results

validation\_results = validate\_error\_injection(

train\_data,

modified\_data,

ground\_truth

)

**Output**: Dictionary of validation metrics

**4. Next Steps for Implementation**

 **Data Analysis**

 Analyze real data to determine:

 Error frequencies

 Error durations

 Typical magnitudes

 Seasonal patterns

 **ML Model Selection**

 Choose two unsupervised methods

 Potential options:

 Isolation Forest

 Local Outlier Factor

 One-class SVM

 Autoencoders

 DBSCAN

 **Configuration Tuning**

 Update config.py with real-world parameters

 Set appropriate thresholds

 Define validation metrics

 **Validation Strategy**

 Define success criteria

 Set performance benchmarks

 Create evaluation metrics