Documentation

Edit New page Jump to bottom

Mehdi-kbz edited this page now · 5 revisions

User Documentation

Overview

This guide explains how to use the RTDIP Forecasting Components for analyzing and forecasting Shell TagMeasurements data.

Prerequisites

- Dev container built and running (see Build & Deploy Documentation)
- Micromamba environment activated: `micromamba activate rtdip-sdk`
- Raw Shell data available (CSV format)

Quick Start

End-to-End Pipeline

The simplest way to run the complete pipeline:

```
# Run full pipeline (preprocessing + training)
python pipeline_shell_data.py
```

This script orchestrates:

1. **Preprocessing**: Load raw data and apply RTDIP transformations
2. **Training**: Train AutoGluon forecasting models

Each step can be skipped if already completed (checkpointing).

Pipeline Options

Skip preprocessing if already done:

```
python pipeline_shell_data.py --skip-preprocessing
```

Skip training if already done:

```
python pipeline_shell_data.py --skip-training
```

Custom paths:

```
python pipeline_shell_data.py --raw-data ShellData.csv --preprocessed-dat o
```

Run on sample data:

```
python pipeline_shell_data.py --sample 0.1
```

Individual Scripts

If you need more control, run scripts separately in `amos_team_resources/shell/` :

1. Explore the data:

```
cd shell/exploration
python explore_data.py
```

2. Preprocess data:

```
cd shell/preprocessing
python preprocess_shell_data.py
```

3. Train models:

```
cd shell/training
python train_comparison.py
```

4. Generate visualizations:

```
cd shell/visualisation
python forecasting_visualization.py
```

All outputs are automatically saved to `output_images/`

Basic Workflow

If building custom analysis:

1. **Load Data** - Read parquet files from Azure Storage
2. **Filter & Clean** - Keep only "Good" status readings, remove nulls
3. **Select Tags** - Choose specific sensor tags to forecast
4. **Train Model** - Use ARIMA/SARIMA models with train/test split (80/20)
5. **Generate Forecasts** - Predict future values with confidence intervals
6. **Evaluate** - Calculate MAE, RMSE metrics on test data

Model Options

ARIMA - Standard time series forecasting for single tags

SARIMA - Seasonal ARIMA for data with patterns (e.g., 24-hour cycles)

Auto ARIMA - Automatically selects best model parameters

Batch Processing - Process multiple tags simultaneously

Output Files

Results saved to `output_images/` :

- Forecast plots (actual vs predicted values)
- Feature importance charts
- Performance metrics (MAE, RMSE)
- Model comparison reports

Common Issues

Memory errors:

- Increase Spark driver memory: `export SPARK_DRIVER_MEMORY=8g`

Slow processing:

- Use data partitioning for large datasets

Model convergence:

- Use Auto ARIMA for automatic parameter selection

Testing

Run test suite:

```
pytest tests/ -v
```

Design Documentation

Architecture

Pipeline architecture processing Shell TagMeasurements rows:

```
Delta Storage → Sources → Monitoring → Data Manipulation → Transformer →
Forecasting
```

Project Structure

```
Deliverables/sprint-{01-05}/    # Sprint deliverables
amos_team_resources/            # Team docs (eia, opsd, scada)
...
shell/
  ├── exploration/              # Data exploration scripts
  ├── preprocessing/            # Cleaning & transformation
  ├── training/                 # Model training per dataset
  └── visualisation/            # Analysis & reporting
tests/                          # Test suite
```

Technology Stack

- **Storage**: Delta Lake (Parquet format)
- **Processing**: PySpark 3.3-3.5
- **Quality:** Great Expectations
- **Forecasting**: statsmodels, pmdarima
- **ML**: PySpark MLlib
- **Testing**: pytest

Components

1. **Delta Storage**: ACID-compliant storage with schema `TagName`, `EventTime`, `Status`, `Value`
2. **Sources**: Streams from Delta/CSV, bridges PySpark ↔ pandas
3. **Monitoring**: Validates Status field, filters bad readings
4. **Data Manipulation**: Handles missing values, outliers, resampling
5. **Transformer**: Creates lag features, rolling stats, time features
6. **Forecasting**: ARIMA/SARIMA models with automated tuning

Design Patterns

```python
# Pipeline
class PipelineComponent:
    def process(self, data): pass

# Strategy (Models)
class BaseForecaster:
    def fit(self, train_data): pass
    def predict(self, horizon): pass
    def evaluate(self, test_data): pass
```

Data Schemas

Input: `TagName`, `EventTime`, `Status`, `Value`

Output: `TagName`, `ForecastTime`, `PredictedValue`, `LowerBound`, `UpperBound`, `ModelType`

Key Implementation

```python
# Feature engineering
class TimeSeriesTransformer:
    def transform(self, df):
        window = Window.partitionBy('TagName').orderBy('EventTime')
        df = df.withColumn('lag_7', F.lag('Value', 7).over(window))
        df = df.withColumn('rolling_mean_7', F.avg('Value').over(window.rowsBe
        return df

# Forecasting
class ARIMAForecaster(BaseForecaster):
    def fit(self, train_data):
        if isinstance(train_data, pyspark.sql.DataFrame):
            train_data = train_data.toPandas().set_index('EventTime')
        self.model = ARIMA(train_data, order=self.order).fit()

    def predict(self, horizon):
        return self.model.forecast(steps=horizon)
```

Scaling

- **Local:** 1M rows, PySpark local mode, pandas-ARIMA
- **Production:** 214M+ rows, Spark cluster, distributed MLlib + pandas-ARIMA
- **Strategy**: Partition by TagName, pre-compute features

Testing

- Unit tests: transformers, forecasters, metrics
- Integration tests: end-to-end pipeline
- Backtesting on historical splits
- Performance benchmarks (MAE, RMSE)

Reference: Sprint 02 architecture documentation

Build and Deploy Documentation

Overview

This guide covers building the RTDIP Forecasting Components development environment and running the data pipeline. The process uses VS Code Dev Containers with micromamba for dependency management.

Prerequisites

- Git
- Visual Studio Code
- Docker Desktop
- VS Code Dev Containers extension
- Azure Storage credentials (for data fetching)

Build Process

1. Clone Repository

```
git clone https://github.com/amosproj/amos2025ws03-rtdip-timeseries-forec n
cd amos2025ws03-rtdip-timeseries-forecasting
code .
```

2. Build Dev Container

The project uses a custom dev container configuration (updated in Sprint 5 due to dependency changes).

Steps:

1. VS Code will detect `.devcontainer` configuration
2. Click "Reopen in Container" when prompted
3. Container build takes 5-10 minutes
4. If extensions fail to load, reload the VS Code window

Manual rebuild:

- Open Command Palette (`Ctrl+Shift+P` / `Cmd+Shift+P`)
- Run: `Dev Containers: Rebuild Container`

3. Activate Micromamba Environment

Once inside the dev container:

```
micromamba activate rtdip-sdk
```

Running the Pipeline

Stage 1: Data Fetching from Azure

Setup credentials:

Create `.env` file in project root:

```
AZURE_STORAGE_ACCOUNT=<your_account>
AZURE_STORAGE_KEY=<your_key>
AZURE_CONTAINER_NAME=<container_name>
```

Run data fetch:

```
python3 amos_team_resssources/shell/preprocessing/preprocess_shell_data.p
```

This downloads all parquet files from Azure Storage and aggregates them into a master training dataset (several GB).

Run Training:

```
python3 amos_team_resources/shell/training/train_comparison.py
```

Run Visualization:

```
jupyter notebook amos_team_resources/shell/visualisation/forecasting_visu a
```

All visualizations are automatically saved to `output_images/`

Testing

Run Full Test Suite

From repository root:

```
pytest -v
```

Expected results:

- ~786 tests pass
- 2 tests skipped (EIA API key not provided, dependency issue)
- Runtime: ~15-20 minutes (one test takes ~10 minutes alone)

Run Specific Tests

```
pytest tests/forecasting/ -v
```

CI/CD Pipeline

Current Status: CI/CD workflows are outdated due to Sprint 5 dependency changes. Pipeline will be updated in Sprint 6.

Troubleshooting

VS Code extensions not loading:

- Reload window: `Ctrl+Shift+P` → "Developer: Reload Window"

Credential errors in logs:

- The data fetch script has been patched to prevent credential leakage
- Ensure `.env` file is in `.gitignore`

Memory issues:

- Full pipeline requires a lot of RAM
- Consider running on cloud VM or limiting dataset size for testing

Deployment

[TO BE ADDED: Production deployment steps will be documented after CI/CD pipeline is finalized in Sprint 6]

+ Add a custom footer

Pages 13

Find a page…

+ Add a custom sidebar

Clone this wiki locally

```
https://github.com/amosproj/amos2025ws03-rtdip-timeseries-forecasting.wiki.git
```