Novartis Datathon 2025 - Step-by-Step Guide

Table of Contents

1. Project Overview
2. Installation & Setup
3. Configuration Files
4. Step 1: Data Loading
5. Step 2: Feature Engineering
6. Step 3: Validation & Splitting
7. Step 4: Model Training
8. Step 5: Evaluation
9. Step 6: Inference & Submission
10. Complete Pipeline Examples
11. Troubleshooting

---

1. Project Overview

What This Project Does

This project predicts **pharmaceutical brand volume erosion** after a drug loses patent protection (Loss of Exclusivity / LOE). When generics enter the market, branded drug sales typically decline - this project forecasts that decline over 24 months.

The Two Scenarios

Scenario	What You Know	What You Predict	Use Case
Scenario 1	Only pre-LOE history (months < 0)	Months 0-23	Long-range forecast
Scenario 2	Pre-LOE + first 6 months (0-5)	Months 6-23	Short-term update

Key Concept: Normalized Volume

Instead of predicting raw sales volume, we predict **normalized volume**:

```
y_norm = volume / avg_vol_12m
```

Where `avg_vol_12m` is the 12-month average BEFORE the drug lost exclusivity.

- `y_norm = 1.0` → Same volume as before LOE
- `y_norm = 0.5` → Half the volume (50% erosion)
- `y_norm = 0.1` → 90% erosion

To get actual volume: `volume = y_norm × avg_vol_12m`

Bucket Classification

Series (country + brand combinations) are classified:

- **Bucket 1** (high erosion): Average erosion ≤ 25% → Weighted 2× in scoring
- **Bucket 2** (low erosion): Average erosion > 25% → Weighted 1× in scoring

---

2. Installation & Setup

Step 2.1: Navigate to Project

```
cd "d:\Datathon\novartis_datathon_2025\novartis_datathon_2025-Arman"
```

Step 2.2: Create Virtual Environment

```
python -m venv .venv
.\.venv\Scripts\Activate.ps1
```

Step 2.3: Install Dependencies

```
pip install -r requirements.txt
```

Step 2.4: Verify Installation

```
pytest tests/ -v --tb=short
```

Step 2.5: Prepare Your Data

Place your data files in:

```
data/raw/TRAIN/
├── df_volume_train.csv      # Monthly sales volumes
├── df_generics_train.csv    # Number of generic competitors
└── df_medicine_info_train.csv  # Drug characteristics

data/raw/TEST/
├── df_volume_test.csv
├── df_generics_test.csv
└── df_medicine_info_test.csv
```

---

3. Configuration Files

All settings are in YAML files in `configs/`:

3.1 `configs/data.yaml` - Data Settings

Purpose: Define file paths, column names, and data schema.

```yaml
# Key sections:
paths:
  raw_dir: "data/raw"           # Where raw CSV files live
  processed_dir: "data/processed"  # Cached processed data

files:
  train:
    volume: "TRAIN/df_volume_train.csv"
    generics: "TRAIN/df_generics_train.csv"
    medicine_info: "TRAIN/df_medicine_info_train.csv"

columns:
  id_keys: ["country", "brand_name"]  # Unique identifier
  time_key: "months_postgx"           # Time index
  raw_target: "volume"                # What we predict
  model_target: "y_norm"              # Normalized target
```

3.2 `configs/features.yaml` - Feature Engineering

Purpose: Control which features to create.

```yaml
pre_entry:
  windows: [3, 6, 12]         # Rolling averages: 3, 6, 12 months
  compute_trend: true          # Linear slope before LOE
  compute_volatility: true     # Stability measure
  compute_seasonal: true       # Seasonal patterns

time:
  include_months_postgx: true  # Time since LOE
  include_decay: true          # Exponential decay features
  decay_alpha: 0.1             # Decay rate

generics:
  include_n_gxs: true          # Number of generic competitors
  include_future_n_gxs: true   # Expected future generics
```

3.3 `configs/run_defaults.yaml` - Training Settings

Purpose: Reproducibility, validation, sample weights.

```yaml
reproducibility:
  seed: 42                    # Random seed for reproducibility

validation:
  val_fraction: 0.2           # 20% for validation
  stratify_by: "bucket"       # Stratify by erosion bucket
  split_level: "series"       # CRITICAL: Split by series, not rows

sample_weights:
  scenario1:
    months_0_5: 3.0           # Early months weighted 3x
    months_6_11: 1.5          # Mid months weighted 1.5x
    months_12_23: 1.0         # Late months weighted 1x
```

3.4 `configs/model_cat.yaml` - Model Hyperparameters

Purpose: CatBoost-specific settings.

```yaml
model:
  type: catboost
  hyperparameters:
    iterations: 1000
    learning_rate: 0.05
    depth: 6
    l2_leaf_reg: 3.0
    early_stopping_rounds: 50
```

---

4. Step 1: Data Loading (`src/data.py`)

What It Does

1. **Loads** three CSV files (volume, generics, medicine_info)
2. **Validates** schema and checks for duplicates
3. **Joins** them into a unified panel dataset
4. **Computes** pre-entry statistics (avg_vol_12m, bucket)
5. **Caches** to Parquet for faster reloads

Input Files

File	Columns	Description
`df_volume_*.csv`	country, brand_name, month, months_postgx, volume	Monthly sales data
`df_generics_*.csv`	country, brand_name, months_postgx, n_gxs	Generic competitor count

File	Columns	Description
`df_medicine_info_*.csv`	country, brand_name, ther_area, hospital_rate, ...	Drug characteristics

Key Functions

```
# Load raw CSV files
raw_data = load_raw_data(data_config, split='train')
# Returns: {'volume': df, 'generics': df, 'medicine_info': df}

# Build unified panel
panel = prepare_base_panel(
    raw_data['volume'],
    raw_data['generics'],
    raw_data['medicine_info']
)
# Joins all three datasets on (country, brand_name)

# Compute pre-entry statistics
panel = compute_pre_entry_stats(panel)
# Adds: avg_vol_12m, bucket, y_norm (for training)

# Handle missing values
panel = handle_missing_values(panel, data_config)
# Fills NaN based on config strategies
```

Output

A **panel DataFrame** with one row per (country, brand_name, months_postgx):

country	brand_name	months_postgx	volume	n_gxs	avg_vol_12m	bucket	y_norm
USA	DrugA	-12	10000	0	9500	1	1.05
USA	DrugA	-11	9800	0	9500	1	1.03
...	...	...	...	...	...	...	...
USA	DrugA	23	2500	5	9500	1	0.26

How to Run (Python)

```
from src.data import load_raw_data, prepare_base_panel, compute_pre_entry_stats
from src.utils import load_config

# Load configs
data_config = load_config('configs/data.yaml')

# Load and process
raw_data = load_raw_data(data_config, split='train')
panel = prepare_base_panel(
```

```
        raw_data['volume'],
        raw_data['generics'],
        raw_data['medicine_info']
    )
    panel = compute_pre_entry_stats(panel)
    print(f"Panel shape: {panel.shape}")
```

How to Run (CLI)

```
python -m src.data --split train --scenario 1 --mode train --data-config
configs/data.yaml
```

---

5. Step 2: Feature Engineering (`src/features.py`)

What It Does

Creates ML features from the panel data while **preventing data leakage**.

Leakage Prevention (CRITICAL)

The model must not "see the future":

- **Scenario 1**: Only use data from months_postgx < 0
- **Scenario 2**: Only use data from months_postgx < 6

Feature Categories

5.1 Pre-Entry Features (from months < 0)

Feature	Description
`avg_vol_12m`	12-month average before LOE
`avg_vol_6m` , `avg_vol_3m`	Shorter window averages
`pre_entry_trend`	Linear slope before LOE
`pre_entry_volatility`	Standard deviation / mean
`log_avg_vol`	Log-transformed average
`seasonal_amplitude`	Seasonal pattern strength

5.2 Time Features

Feature	Description
`months_postgx`	Months since LOE (target variable)
`months_postgx_sq`	Squared for non-linear decay

Feature	Description
`is_early`	Binary: months 0-5
`is_mid`	Binary: months 6-11
`is_late`	Binary: months 12-23
`time_decay`	exp(-0.1 × months)
`month_sin`, `month_cos`	Cyclical month encoding

5.3 Competition Features

Feature	Description
`n_gxs`	Number of generics at current month
`has_generic`	Binary: n_gxs > 0
`multiple_generics`	Binary: n_gxs >= 2
`log_n_gxs`	Log-transformed generic count
`n_gxs_at_month_12`	Expected generics at month 12

5.4 Drug Features (Static)

Feature	Description
`hospital_rate`	% sold through hospitals
`is_biological`	Is it a biologic drug?
`ther_area_encoded`	Therapeutic area (label encoded)

5.5 Scenario 2 Early Erosion Features (months 0-5)

Feature	Description
`avg_vol_0_5`	Average volume in first 6 months
`erosion_0_5`	avg_vol_0_5 / avg_vol_12m
`trend_0_5`	Linear slope in first 6 months
`drop_month_0`	Initial drop at month 0

Key Function

```
from src.features import make_features

# Create features for Scenario 1
features_df = make_features(
    panel_df=panel,
    scenario=1,           # 1 or 2
```

```
    mode='train',          # 'train' or 'test'
    config=features_config
)
```

Input

Panel DataFrame from Step 1.

Output

Feature DataFrame with all engineered features + meta columns:

country	brand_name	months_postgx	avg_vol_12m	bucket	y_norm	feature_1	feature_2	...

How to Run

```
from src.features import make_features
from src.utils import load_config

features_config = load_config('configs/features.yaml')
features_df = make_features(panel, scenario=1, mode='train',
config=features_config)
print(f"Features shape: {features_df.shape}")
print(f"Feature columns: {[c for c in features_df.columns if c not in
META_COLS]}")
```

---

6. Step 3: Validation & Splitting (`src/validation.py`)

What It Does

Splits data into training and validation sets **at the series level**.

Why Series-Level Split? (CRITICAL)

If you split randomly by row, you might have:

- Month 5 of DrugA in **training**
- Month 6 of DrugA in **validation**

This is **data leakage** - the model learns from future months!

Correct approach: All months of a series go to either train OR validation.

Key Function

```
from src.validation import create_validation_split
```

```
train_df, val_df = create_validation_split(
    panel_df=features_df,
    val_fraction=0.2,          # 20% of SERIES for validation
    stratify_by='bucket',      # Balance bucket 1 and 2
    random_state=42
)
```

Input

Features DataFrame from Step 2.

Output

Two DataFrames:

- `train_df` : 80% of series (all months for each series)
- `val_df` : 20% of series (all months for each series)

Cross-Validation

For K-fold cross-validation:

```
from src.validation import get_fold_series

folds = get_fold_series(
    panel_df=features_df,
    n_folds=5,
    stratify_by='bucket'
)
# Returns list of (train_indices, val_indices) per fold
```

---

7. Step 4: Model Training (`src/train.py`)

What It Does

1. **Prepares** feature matrix (X) and target (y)
2. **Computes** sample weights aligned with competition metric
3. **Trains** model with early stopping
4. **Logs** metrics and saves artifacts

Available Models

Model	Command	Description
CatBoost	`--model catboost`	Best for tabular data, handles categoricals
LightGBM	`--model lightgbm`	Fast training, good accuracy
XGBoost	`--model xgboost`	Robust gradient boosting

Model	Command	Description
Linear	`--model linear`	Ridge/Lasso regression
Neural Net	`--model nn`	MLP neural network

Sample Weights (IMPORTANT)

Training uses weighted loss to match competition scoring:

```
# Scenario 1 weights:
# - Early months (0-5): 3.0 weight (50% of metric)
# - Mid months (6-11): 1.5 weight
# - Late months (12-23): 1.0 weight
# - Bucket 1: 2x additional weight
```

CLI Commands

Basic Training

```
# Train Scenario 1 model with CatBoost
python -m src.train --scenario 1 --model catboost

# Train Scenario 2 model
python -m src.train --scenario 2 --model catboost
```

With Cross-Validation

```
python -m src.train --scenario 1 --model catboost --cv --n-folds 5
```

With Hyperparameter Optimization

```
python -m src.train --scenario 1 --model catboost --hpo --hpo-trials 50
```

Full Pipeline (Both Scenarios)

```
python -m src.train --full-pipeline --model catboost
```

Python API

```
from src.train import run_experiment

model, results = run_experiment(
    scenario=1,
    model_type='catboost',
    model_config_path='configs/model_cat.yaml',
    run_config_path='configs/run_defaults.yaml',
    data_config_path='configs/data.yaml',
    features_config_path='configs/features.yaml'
)

print(f"Validation Metric: {results['val_metric']}")
```

Output Artifacts

After training, find artifacts in `artifacts/<run_id>/`:

```
artifacts/2025-11-29_catboost_scenario1/
├── model.bin           # Trained model
├── config_snapshot.json  # All configs used
├── metrics.json        # Training/validation metrics
├── feature_importance.csv  # Feature rankings
└── training_log.txt    # Detailed logs
```

---

8. Step 5: Evaluation (`src/evaluate.py`)

What It Does

Computes the **official competition metrics** to validate your model locally.

Metric 1 (Scenario 1)

Used for Phase 1A scoring:

```
Metric 1 = weighted average of:
  - 20% × Monthly absolute error (months 0-23)
  - 50% × Accumulated error (months 0-5)   [MOST IMPORTANT]
  - 20% × Accumulated error (months 6-11)
  - 10% × Accumulated error (months 12-23)

Plus: Bucket 1 weighted 2×, Bucket 2 weighted 1×
```

Metric 2 (Scenario 2)

Used for Phase 1B scoring:

```
Metric 2 = weighted average of:
  - 20% × Monthly absolute error (months 6-23)
  - 50% × Accumulated error (months 6-11)  [MOST IMPORTANT]
  - 30% × Accumulated error (months 12-23)
```

Key Functions

```python
from src.evaluate import compute_metric1, compute_metric2, create_aux_file

# Create auxiliary file (required for metric calculation)
aux_df = create_aux_file(panel_df)
# Contains: country, brand_name, avg_vol, bucket

# Compute Metric 1
score = compute_metric1(
    df_actual=actual_df,   # Actual volumes
    df_pred=pred_df,       # Predicted volumes
    df_aux=aux_df          # Auxiliary data
)
print(f"Metric 1 Score: {score:.4f}")  # Lower is better
```

Per-Bucket Analysis

```python
from src.evaluate import compute_bucket_metrics

bucket_scores = compute_bucket_metrics(actual_df, pred_df, aux_df, scenario=1)
print(f"Overall: {bucket_scores['overall']:.4f}")
print(f"Bucket 1: {bucket_scores['bucket1']:.4f}")
print(f"Bucket 2: {bucket_scores['bucket2']:.4f}")
```

---

9. Step 6: Inference & Submission (`src/inference.py`)

What It Does

1. **Loads** trained model
2. **Prepares** test features
3. **Generates** predictions (y_norm)
4. **Converts** to actual volumes
5. **Validates** submission format

6. **Saves** submission CSV

Submission Format

The competition expects this format:

```
country,brand_name,months_postgx,volume
USA,DrugA,0,9500.0
USA,DrugA,1,9200.0
...
USA,DrugA,23,2500.0
```

CLI Command

```
python -m src.inference \
    --model-s1 artifacts/catboost_s1/model.bin \
    --model-s2 artifacts/catboost_s2/model.bin \
    --output submissions/my_submission.csv
```

Python API

```
from src.inference import generate_submission, validate_submission

# Generate predictions
submission_df = generate_submission(
    model=model,
    test_panel=test_features,
    pre_entry_stats=pre_entry_df,
    scenario=1
)

# Validate format
is_valid, issues = validate_submission(submission_df, template_path)
if is_valid:
    submission_df.to_csv('submissions/submission.csv', index=False)
```

Output

`submissions/submission.csv` ready for upload.

---

10. Complete Pipeline Examples

Example 1: Quick Training (Single Model)

```
# Navigate to project
cd "d:\Datathon\novartis_datathon_2025\novartis_datathon_2025-Arman"

# Activate environment
.\.venv\Scripts\Activate.ps1

# Train Scenario 1
python -m src.train --scenario 1 --model catboost

# Train Scenario 2
python -m src.train --scenario 2 --model catboost

# Generate submission
python -m src.inference --model-s1 artifacts/latest_s1/model.bin --model-s2
artifacts/latest_s2/model.bin --output submissions/submission.csv
```

Example 2: Cross-Validation

```
# 5-fold CV for Scenario 1
python -m src.train --scenario 1 --model catboost --cv --n-folds 5

# Check results
cat artifacts/latest/cv_results.json
```

Example 3: Hyperparameter Optimization

```
# Run 50 trials of HPO
python -m src.train --scenario 1 --model catboost --hpo --hpo-trials 50

# Best params saved to artifacts/
```

Example 4: Full Pipeline (Python Script)

```
"""Complete pipeline from data to submission."""
from src.utils import load_config, set_seed
from src.data import load_raw_data, prepare_base_panel, compute_pre_entry_stats,
handle_missing_values
from src.features import make_features
from src.validation import create_validation_split
from src.train import train_scenario_model
from src.evaluate import compute_metric1, create_aux_file
from src.inference import generate_submission
```

```python
# Setup
set_seed(42)
data_config = load_config('configs/data.yaml')
features_config = load_config('configs/features.yaml')
model_config = load_config('configs/model_cat.yaml')

# Step 1: Load data
print("Loading data...")
raw_data = load_raw_data(data_config, split='train')
panel = prepare_base_panel(
    raw_data['volume'],
    raw_data['generics'],
    raw_data['medicine_info']
)
panel = compute_pre_entry_stats(panel)
panel = handle_missing_values(panel, data_config)

# Step 2: Create features
print("Engineering features...")
features_df = make_features(panel, scenario=1, mode='train',
config=features_config)

# Step 3: Split data
print("Splitting data...")
train_df, val_df = create_validation_split(features_df, val_fraction=0.2)

# Step 4: Train model
print("Training model...")
model, metrics = train_scenario_model(
    train_df=train_df,
    val_df=val_df,
    scenario=1,
    model_type='catboost',
    model_config=model_config
)

# Step 5: Evaluate
print(f"Validation Metric 1: {metrics['metric1']:.4f}")

# Step 6: Generate submission (for test data)
print("Generating submission...")
test_raw = load_raw_data(data_config, split='test')
test_panel = prepare_base_panel(
    test_raw['volume'],
    test_raw['generics'],
    test_raw['medicine_info']
)
test_features = make_features(test_panel, scenario=1, mode='test',
config=features_config)
submission = generate_submission(model, test_features, panel[['country',
'brand_name', 'avg_vol_12m']], scenario=1)
submission.to_csv('submissions/submission.csv', index=False)
print("Done! Submission saved.")
```

Example 5: Using Jupyter Notebook

Open `notebooks/colab/main.ipynb` in VS Code or Jupyter:

```
jupyter notebook notebooks/colab/main.ipynb
```

---

11. Troubleshooting

Common Issues

"ModuleNotFoundError: No module named 'src'"

Solution: Run from project root directory:

```
cd "d:\Datathon\novartis_datathon_2025\novartis_datathon_2025-Arman"
python -m src.train ...  # Use -m flag
```

"FileNotFoundError: data/raw/TRAIN/..."

Solution: Make sure your data files are in the correct location:

```
data/raw/TRAIN/df_volume_train.csv
data/raw/TRAIN/df_generics_train.csv
data/raw/TRAIN/df_medicine_info_train.csv
```

"CUDA out of memory" (GPU)

Solution: Reduce batch size or use CPU:

```
python -m src.train --scenario 1 --model catboost --device cpu
```

"KeyError: 'bucket'" in test mode

Solution: Bucket is only computed for training data. For test data, use mode='test':

```
features = make_features(panel, scenario=1, mode='test', config=cfg)
```

Getting Help

1. **Run tests**: `pytest tests/ -v`
2. **Check logs**: Look in `artifacts/<run_id>/training_log.txt`
3. **Verify configs**: Ensure all YAML files are valid

Summary: The Pipeline Flow

```
┌─────────────────────────────────────────────────────────────┐
│ 1. DATA LOADING (src/data.py)                               │
│    Input: CSV files (volume, generics, medicine_info)       │
│    Output: Unified panel DataFrame                          │
│    Command: python -m src.data --split train                │
└─────────────────────────────────────────────────────────────┘
                              ↓
┌─────────────────────────────────────────────────────────────┐
│ 2. FEATURE ENGINEERING (src/features.py)                    │
│    Input: Panel DataFrame                                   │
│    Output: Features DataFrame (with all ML features)        │
│    Key: Respects scenario cutoff (no future leakage)        │
└─────────────────────────────────────────────────────────────┘
                              ↓
┌─────────────────────────────────────────────────────────────┐
│ 3. VALIDATION SPLIT (src/validation.py)                     │
│    Input: Features DataFrame                                │
│    Output: train_df, val_df (series-level split)            │
│    Key: Same series never in both train and validation      │
└─────────────────────────────────────────────────────────────┘
                              ↓
┌─────────────────────────────────────────────────────────────┐
│ 4. MODEL TRAINING (src/train.py)                            │
│    Input: train_df, val_df                                  │
│    Output: Trained model + metrics                          │
│    Command: python -m src.train --scenario 1 --model catboost │
└─────────────────────────────────────────────────────────────┘
                              ↓
┌─────────────────────────────────────────────────────────────┐
│ 5. EVALUATION (src/evaluate.py)                             │
│    Input: Predictions, actuals                              │
│    Output: Metric 1 / Metric 2 scores                       │
│    Key: Matches official competition scoring                │
└─────────────────────────────────────────────────────────────┘
                              ↓
┌─────────────────────────────────────────────────────────────┐
│ 6. INFERENCE (src/inference.py)                             │
│    Input: Trained model, test features                      │
│    Output: submission.csv                                   │
│    Command: python -m src.inference --model-s1 ... --output ... │
└─────────────────────────────────────────────────────────────┘
```

Document created: November 29, 2025
For Novartis Datathon 2025 - Generic Erosion Forecasting