




Project Explanation - Novartis Datathon 2025

This document explains EXACTLY what each file does, step-by-step, in simple terms.

Based on the competition requirements from `datathon_explanation.md`.



Input/Output Summary Table

Step	Module	Input	Output	Goal
1	<code>config.py</code>	None	Constants, paths, params	Central configuration
2	<code>data_loader.py</code>	Raw CSVs	Merged DataFrame	Load & combine all data
3	<code>bucket_calculator.py</code>	Merged DataFrame	<code>aux_bucket_avgvol.csv</code>	Calculate pre-entry avg & buckets
4	<code>feature_engineering.py</code>	Merged DataFrame + aux	DataFrame + 40 features	Create ML-ready features
5	<code>models.py</code>	Features (X) + Target (y)	Trained model + predictions	Train prediction model
5b	<code>HybridPhysicsMLModel</code>	Features + avg_vol + months	Hybrid model + predictions	Physics + ML combined
5c	<code>ARIHOWModel</code> 	Time-series per brand	ARHOW predictions	SARIMAX + Holt-Winters ensemble
6	<code>evaluation.py</code>	Predictions + Actuals	PE Score	Measure prediction quality
7	<code>submission.py</code>	Test predictions	CSV + JSON summary	Format for competition
8	<code>pipeline.py</code>	All modules	End-to-end execution	Orchestrate workflow



The Big Picture: What Are We Doing?

The Business Problem (In Plain English)

Imagine you're Novartis, a big pharmaceutical company. You sell a drug called "BrandX" for \$100.

Then the patent expires.

Now other companies can make **generic copies** that do the same thing but cost only \$20.

What happens? Your sales **DROP** (this is called **generic erosion**).

The Question: How much will sales drop over the next 24 months?

Why Does This Matter?

Novartis needs to:

- Plan budgets
- Allocate resources
- Make business decisions

Your job: Build a model that predicts this sales decline accurately.

 The Two Scenarios

Scenario	What You Have	What You Predict
Scenario 1	Only BEFORE generic entry	Months 0-23 (all 24 months)
Scenario 2	Before + First 6 months actual	Months 6-23 (remaining 18 months)

Think of it like weather forecasting:

- **Scenario 1** = Predicting next week's weather with only historical data
- **Scenario 2** = Predicting next week's weather knowing today's weather

 The Bucket System (VERY IMPORTANT!)

Not all drugs decline the same way:

Bucket	Mean Erosion	What It Means	Scoring Weight
Bucket 1	≤ 0.25	SEVERE erosion (sales crash to <25% of original)	2× (double!)
Bucket 2	> 0.25	Moderate erosion	1× (normal)




Why This Matters:

If you predict Bucket 1 drugs wrong, your score gets **penalized TWICE as much!**

Strategy: Focus extra effort on predicting high-erosion drugs accurately.

 File-by-File Explanation

1 `src/config.py` - The Central Control Panel NEW

Description	
 INPUT	None (configuration file)
 OUTPUT	Python constants, paths, toggles, and parameters importable by other modules
 GOAL	Centralize ALL settings so you can change them in ONE place and run the entire pipeline

What it does: Stores ALL settings, toggles, and constants in ONE place. **This is the ONLY file you need to edit to customize the pipeline!**

Configuration Sections:

```
#
=====
# 1. RUN MODE & TOGGLES - Control what runs
#
=====
RUN_SCENARIO = 1           # 1, 2, or [1, 2] for both scenarios
TEST_MODE = True           # True = fast (50 brands), False = full training
TEST_MODE_BRANDS = 50      # Number of brands in test mode
SUBMISSION_MODEL = 'baseline' # Model for final submissions

# Pipeline step toggles
RUN_EDA = False            # Run EDA visualization
RUN_TRAINING = True        # Train models
RUN_SUBMISSION = False     # Generate submission files
RUN_VALIDATION = False     # Validate submissions

#
=====
# 2. MODEL TOGGLES - Enable/disable specific models
#
=====
MODELS_ENABLED = {
    'baseline_no_erosion': True,    # ~1 sec - Predicts avg_vol (no decline)
    'baseline_exp_decay': True,    # ~2 sec - Exponential decay (BEST!)
    'lightgbm': True,              # ~5 sec - LightGBM gradient boosting
    'xgboost': True,              # ~10 sec - XGBoost gradient boosting
    'hybrid_lightgbm': True,       # ~8 sec - Decay + LightGBM residual
    'hybrid_xgboost': True,       # ~12 sec - Decay + XGBoost residual
    'arihow': True,               # ~30 sec - SARIMAX + Holt-Winters
}




#
=====
# 3-13. Other sections: Paths, Constants, Model Parameters, etc.
#
=====
```

View current config:

```
python src/config.py
```

In simple terms: This is like a "control panel" where you adjust all the knobs in one place instead of hunting through every file.

2 src/data_loader.py - The Data Reader

Description	
 INPUT	3 raw CSV files: df_volume_[train/test].csv , df_generics_[train/test].csv , df_medicine_info.csv
 OUTPUT	Single merged DataFrame with all columns from all 3 files
 GOAL	Combine scattered data into ONE unified table for analysis

What it does: Loads the 3 CSV files and combines them.

The 3 datasets:

File	Contains	Key Columns
df_volume	Sales history	volume (units sold each month)
df_generics	Competition info	n_gxs (number of generic competitors)
df_medicine_info	Drug characteristics	ther_area , hospital_rate , etc.

Key functions:

```
# Load one dataset
volume_df = load_volume_data(train=True)




# Load all 3 datasets
volume, generics, medicine = load_all_data(train=True)

# MERGE them together into ONE table
merged = merge_datasets(volume, generics, medicine)
```

In simple terms: Like gathering ingredients from 3 different shelves and putting them on ONE cutting board.

Output: A single DataFrame with ALL information per (country, brand, month).

3 `src/bucket_calculator.py` - The Normalization Engine

Description	
 INPUT	Merged DataFrame with <code>volume</code> column and <code>months_pregx</code> (pre-entry months: -12 to -1)
 OUTPUT	<code>aux_bucket_avgvol.csv</code> file with columns: <code>country</code> , <code>brand_name</code> , <code>avg_vol</code> , <code>mean_erosion</code> , <code>bucket</code>
 GOAL	Calculate the baseline (avg_vol) and bucket assignment for scoring

What it does: Calculates key metrics for evaluation.

Step-by-step:

Step A: Calculate `Avg_j` (Pre-entry Average)

For each drug:
 `Avg_j` = average volume from month -12 to -1

Example:

- BrandX sold 1000, 1100, 900, ... units in the 12 months BEFORE generics arrived
- `Avg_j` = average of these = 1000 units

Why? This is the "baseline" to compare against. If you predict 500 units and `Avg_j` was 1000, that's 50% of baseline.

Step B: Calculate Normalized Volume

`Normalized Volume = Actual Volume / Avg_j`

Example:

- Month 3 actual: 400 units
- `Avg_j`: 1000 units
- Normalized: 400/1000 = 0.4 (40% of original sales)

Step C: Calculate Mean Erosion

Mean Erosion = average(Normalized Volume) over months 0-23

Example:

- BrandX normalized volumes: [0.8, 0.6, 0.4, 0.3, 0.2, ...]
- Mean erosion = average = 0.35




Step D: Assign Buckets

```
if Mean Erosion ≤ 0.25:
    Bucket = 1 (HIGH erosion - severe sales crash)
else:
    Bucket = 2 (LOWER erosion - moderate decline)
```

Output file: `aux_bucket_avgvol.csv` containing:

- `country` , `brand_name`
- `avg_vol` (the pre-entry average)
- `mean_erosion`
- `bucket` (1 or 2)

4 `src/feature_engineering.py` - The Feature Factory

Description	
 INPUT	Merged DataFrame + <code>aux_bucket_avgvol.csv</code> (for avg_vol baseline)
 OUTPUT	DataFrame with original columns + ~40 NEW feature columns
 GOAL	Transform raw data into ML-friendly patterns that capture erosion dynamics

What it does: Creates ~40 features for the ML model to learn from.

Feature Categories:

A. Lag Features (Past Values)

```
volume_lag_1  = volume from 1 month ago
volume_lag_3  = volume from 3 months ago
volume_lag_6  = volume from 6 months ago
volume_lag_12 = volume from 12 months ago
```

Why? Recent sales predict future sales.

B. Rolling Features (Trends)

```
rolling_mean_3  = average of last 3 months  
rolling_std_3   = volatility of last 3 months  
rolling_mean_6  = average of last 6 months  
rolling_mean_12 = average of last 12 months
```

Why? Captures momentum and stability.

C. Competition Features (Generics)

```
n_gxs           = number of generic competitors NOW  
n_gxs_cummax     = maximum competitors seen so far  
months_with_generics = how long generics have been in market
```

Why? More competitors = more erosion.

D. Pre-Entry Features (CRITICAL for Scenario 1)

```
avg_vol         = pre-entry average (the baseline)  
pre_entry_slope = was sales growing or declining BEFORE generics?  
pre_entry_volatility = how stable were sales before?
```




Why? In Scenario 1, this is ALL you have to predict with!

E. Time Features

```
months_postgx = months since generic entry (0, 1, 2, ...)  
month_sin/cos = capture seasonality  
is_early_postgx = is this in first 6 months? (binary)
```

Output: DataFrame with original data + ~40 new feature columns.

5 `src/models.py` - The Prediction Models

Description	
 INPUT	Feature matrix X (from feature_engineering), target vector y (actual volumes), avg_vol per brand
 OUTPUT	Trained model object + predictions array + saved model file (<code>.joblib</code>)
 GOAL	Learn patterns from historical data to predict future sales volumes

What it does: Implements different prediction strategies.

A. Baseline Models (Simple)

1. No Erosion Baseline:

```
prediction = avg_vol # (assume sales stay the same forever)
```

This is the WORST case - it ignores erosion completely.

2. Linear Decay:

```
prediction = avg_vol × (1 - 0.03 × month)
```

Sales drop by 3% each month in a straight line.

3. Exponential Decay: ☒ BEST PERFORMER

```
prediction = avg_vol × exp(-0.05 × month)
```

Sales drop quickly at first, then slow down (like real erosion!).

Why exponential works best: Generic erosion follows this pattern naturally:

- Month 0-3: Big drop (generics are new, doctors switch)
- Month 12+: Slower decline (loyal patients stay)

B. ML Models (Complex)

LightGBM / XGBoost:

- Use ALL 40 features
- Learn patterns from historical data

- Can capture non-linear relationships

```
model = GradientBoostingModel(model_type='lightgbm')
model.fit(X_train, y_train)
predictions = model.predict(X_test)
model.save("scenario1_lightgbm")
```

Current Result: Baseline exponential (PE=1.18) beats ML models (PE=2.84+)

Why? The decay pattern is so consistent that a simple formula works better than complex ML on this data.

C. Hybrid Model (Physics + ML) NEW

The Best of Both Worlds!

```
# Physics-based baseline
base_prediction = avg_vol × exp(-0.05 × month)

# ML learns the residuals (what physics misses)
residual = ML_model.predict(features)

# Combine them
final_prediction = base_prediction + residual
```

Why Hybrid?

- Physics captures the KNOWN decay pattern
- ML corrects systematic deviations
- Best of both worlds!

Class: `HybridPhysicsMLModel`

```
from models import HybridPhysicsMLModel

hybrid = HybridPhysicsMLModel(
    ml_model_type='lightgbm', # or 'xgboost'
    decay_rate=0.05
)

# Training requires avg_vol and months for physics baseline
hybrid.fit(X_train, y_train, avg_vol_train, months_train,
           X_val, y_val, avg_vol_val, months_val)

# Prediction also needs avg_vol and months
predictions = hybrid.predict(X_test, avg_vol_test, months_test)
```

```
# Save/Load
hybrid.save("scenario1_hybrid")
hybrid.load("scenario1_hybrid")
```

Current Results:

Model	Scenario 1 PE	Scenario 2 PE
Hybrid-Physics+LightGBM 🏆	1.08 ✓	29.60
Exponential Decay	1.18	1.10 ✓
XGBoost	2.84	3.39

Key Insight: Hybrid wins for Scenario 1 (no actuals), Baseline wins for Scenario 2 (has 0-5 actuals).

5c. ARHOW Model (SARIMAX + Holt-Winters Ensemble) NEW

Time-Series Based Forecasting!

The ARHOW (ARIMA + Holt-Winters) model uses a sophisticated ensemble approach:

```
# For each brand, fit two time-series models on pre-entry data:
1. SARIMAX - Captures autocorrelation and trend
2. Holt-Winters (ExponentialSmoothing) - Captures level and trend

# Learn optimal weights via Linear Regression:
y_hat = β0 × y_SARIMAX + β1 × y_HW

# The weights are learned from the last N observations
```

Why This Approach?

- SARIMAX captures autoregressive patterns in the data
- Holt-Winters captures exponential smoothing trends
- Learned weights (β₀, β₁) adapt to each brand's characteristics
- No manual weight tuning required

Class: `ARIHOWModel`

```
from models import ARIHOWModel

arhow = ARIHOWModel(
    arima_order=(1, 1, 1),      # ARIMA(p,d,q)
    seasonal_order=(0, 0, 0, 0), # SARIMA seasonal
```

```
hw_trend='add',           # Holt-Winters trend type
hw_seasonal=None,         # HW seasonality
hw_seasonal_periods=12,   # Seasonal period
weight_window=12          # Observations for weight learning
)

# Fit on ALL brands (not just training set)
arhow.fit(df) # Full dataframe with all brands




# Predict for test brands
predictions = arhow.predict(X_test)

# Inspect learned weights per brand
weights = arhow.get_brand_weights()
# Returns: {(country, brand): {'beta0': 0.6, 'beta1': 0.4, 'method':
'weights'}, ...}
```

Key Features:

- ☒ Fits on ALL brands' pre-entry data (not just training brands)
- ☒ Uses `get_forecast()` with RangeIndex for statsmodels compatibility
- ☒ Falls back to exponential decay if time-series fitting fails
- ☒ Suppresses statsmodels warnings for cleaner output

6 `src/evaluation.py` - The Scoring System

Description	
 INPUT	Predictions DataFrame, Actuals DataFrame, <code>aux_bucket_avgvol.csv</code> (for avg_vol & buckets)
 OUTPUT	PE score (float) - lower is better
 GOAL	Measure prediction accuracy using the EXACT same formula the competition uses

What it does: Calculates the official competition metric (PE = Prediction Error).

Scenario 1 PE Formula:

PE = 0.2 × (monthly errors normalized)

+ 0.5 × (error in months 0-5 sum) ← 50% WEIGHT!

+ 0.2 × (error in months 6-11 sum)

+ 0.1 × (error in months 12-23 sum)

In plain English:

- Getting months 0-5 right is HALF your score
- Monthly individual errors = 20%
- Later months matter less

Scenario 2 PE Formula:

$$\begin{aligned} \text{PE} &= 0.2 \times (\text{monthly errors normalized}) \\ &+ 0.5 \times (\text{error in months 6-11 sum}) \quad \leftarrow 50\% \text{ WEIGHT!} \\ &+ 0.3 \times (\text{error in months 12-23 sum}) \end{aligned}$$

Final Score Calculation:

$$\text{Final Score} = \frac{(2 \times \text{avg_PE_bucket1} + 1 \times \text{avg_PE_bucket2})}{\text{total_weighted_brands}}$$

The key insight: Bucket 1 errors count DOUBLE.




Example:

- Bucket 1 average PE: 0.5 (10 brands)
- Bucket 2 average PE: 0.3 (100 brands)
- Final = $(2 \times 0.5 \times 10 + 1 \times 0.3 \times 100) / (2 \times 10 + 1 \times 100) = (10 + 30) / 120 = 0.33$

7

src/submission.py

 - The Output Generator

Description	
 INPUT	Predictions DataFrame with columns: <code>country</code> , <code>brand_name</code> , <code>months_postgx</code> , <code>volume</code>
 OUTPUT	CSV file formatted for competition upload (e.g., <code>scenario1_baseline_final.csv</code>)
 GOAL	Format predictions into the EXACT structure required by the competition

What it does: Creates the CSV file you upload to the competition.

Required Format:

country	brand_name	months_postgx	volume
COUNTRY_001	BRAND_ABC	0	1234.56
COUNTRY_001	BRAND_ABC	1	1100.23

country	brand_name	months_postgx	volume
...
COUNTRY_001	BRAND_ABC	23	456.78

Validation Checks:

- ☑ All required columns present
- ☑ No missing values
- ☑ No negative volumes
- ☑ Correct months for scenario (0-23 or 6-23)
- ☑ Every brand has all required months
- ☑ Total rows = brands × months

Example output:

- Scenario 1: 340 brands × 24 months = 8,160 rows
- Scenario 2: 340 brands × 18 months = 6,120 rows

8 src/pipeline.py - The Orchestrator

Description	
📁 INPUT	Command-line arguments: <code>--scenario</code> (1 or 2), <code>--model</code> (baseline/lightgbm/xgboost)
📁 OUTPUT	Trained model files, submission CSVs, performance reports
🎯 GOAL	Run the ENTIRE workflow from raw data to final submission in ONE command

What it does: Runs EVERYTHING in the correct order.

The Pipeline Steps:




```
STEP 1: Load Data
  ↓
STEP 2: Create Auxiliary File (avg_vol, buckets)
  ↓
STEP 3: Feature Engineering (create 40 features)
  ↓
STEP 4: Split Train/Validation
  ↓
STEP 5: Prepare X (features) and y (target)
  ↓
STEP 6: Train Model
  ↓
STEP 7: Evaluate on Validation Set
```

↓
STEP 8: Generate Submission File

Usage:

```
python src/pipeline.py --scenario 1 --model lightgbm
```

9 src/eda_analysis.py - The Data Explorer

Description	
 INPUT	Merged DataFrame from data_loader
 OUTPUT	Dictionary of statistics, DataFrames with aggregated insights
 GOAL	Understand data patterns BEFORE modeling to inform feature engineering

What it does: Analyzes and understands the data BEFORE modeling.

Key analyses:

```
# Data quality
- Missing values per column
- Duplicate records
- Negative volumes




# Distribution analysis
- Volume distribution (heavily right-skewed)
- Brands per country
- Brands per therapeutic area

# Erosion analysis
- Average erosion curve over 24 months
- Erosion by bucket
- Impact of competition on erosion

# Bucket analysis
- How many Bucket 1 vs Bucket 2?
- Characteristics of high-erosion drugs
```

 Scripts Explained

scripts/run_demo.py

Description	
 INPUT	Raw data files
 OUTPUT	Console output showing pipeline works
 GOAL	Quick sanity check that all code works

Purpose: Quick test to make sure everything works.




```
python scripts/run_demo.py
```

What it does:

1. Loads small sample of data
2. Creates features
3. Trains baseline model
4. Evaluates predictions
5. Generates sample submission

Use when: You want to quickly verify the code works.

```
scripts/train_models.py
```

Description	
 INPUT	Raw data files, <code>--scenario</code> argument (1 or 2)
 OUTPUT	Model files in <code>models/</code> , comparison CSV in <code>reports/</code>
 GOAL	Train ALL models and find the best one for each scenario

Purpose: Train and compare ALL models.

```
python scripts/train_models.py --scenario 1
python scripts/train_models.py --scenario 2
```




What it does:

1. Trains No Erosion baseline
2. Trains Exponential Decay baseline (tunes λ)
3. Trains LightGBM
4. Trains XGBoost
5. Compares all models
6. Saves best models to `models/`

7. Saves comparison to `reports/model_comparison_scenarioX.csv`

`scripts/generate_final_submissions.py`

Description

 INPUT	Test data, trained models, <code>--model</code> argument
 OUTPUT	<code>submissions/scenario1*_final.csv</code> , <code>submissions/scenario2*_final.csv</code>
 GOAL	Create the final CSV files ready for competition upload

Purpose: Create the final competition submission files.




```
python scripts/generate_final_submissions.py --model baseline
```

Output:

- `submissions/scenario1_baseline_final.csv`
- `submissions/scenario2_baseline_final.csv`

`scripts/validate_submissions.py`

Description

 INPUT	Submission CSV files in <code>submissions/</code> folder
 OUTPUT	Console output with pass/fail for each check
 GOAL	Verify submission files meet ALL competition requirements BEFORE upload

Purpose: Check submissions BEFORE uploading.

```
python scripts/validate_submissions.py
```

Checks:

- ☒ Correct column names
- ☒ No missing values
- ☒ No negative volumes
- ☒ Correct months per scenario
- ☒ All brands present
- ☒ Correct total row count

Notebooks Explained

Notebook	Input	Output	Goal
01_eda_visualization.ipynb	Raw data	Visualizations + insights	Understand data before modeling
02_feature_exploration.ipynb	Feature DataFrame	Feature correlation heatmaps, importance charts	Validate feature engineering

03_model_results.ipynb	Model comparison CSVs	Performance charts	Compare models, analyze predictions
------------------------	-----------------------	--------------------	-------------------------------------

Notebook	Purpose
01_eda_visualization.ipynb	See data distributions, erosion curves, bucket breakdown
02_feature_exploration.ipynb	Visualize features, correlations, importances
03_model_results.ipynb	Compare model performance, analyze submissions

These are for **VISUALIZATION** only - all logic is in `src/` files.

EDA Data Export NEW

The EDA notebook (`01_eda_visualization.py`) now exports both JSON and CSV files for all figures:

```
reports/eda_data/
├─ fig01_volume_distribution.json      # JSON summary
├─ fig01_volume_distribution.csv      # Full data (histogram bin edges, counts)
├─ fig02_erosion_curves.json
├─ fig02_erosion_curves.csv          # Erosion curves per bucket
├─ fig03_bucket_distribution.json
├─ fig03_bucket_distribution.csv      # Bucket counts by country
├─ fig04_generic_impact.json
├─ fig04_generic_impact.csv          # Volume by generics count
├─ fig05_therapeutic_area.json
├─ fig05_therapeutic_area.csv        # Metrics by therapeutic area
├─ fig06_monthly_patterns.json
├─ fig06_monthly_patterns.csv        # Monthly volume trends
├─ fig07_correlation_matrix.json
├─ fig07_correlation_matrix.csv      # Feature correlations
├─ fig08_country_analysis.json
└─ fig08_country_analysis.csv        # Country-level analysis
```

CSV Data Contents:

- Raw data used to generate each figure

- Can be used for custom visualizations or further analysis
- Complements JSON summary files

Complete Workflow

Option 1: Config-Based (Recommended) 

Just edit `src/config.py` toggles and run ONE command:

```
# 1. Edit config.py to set your preferences:
#   RUN_SCENARIO = 1           # or 2, or [1, 2]
#   TEST_MODE = True          # False for full training
#   MODELS_ENABLED = {...}    # Enable/disable models
#   RUN_TRAINING = True
#   RUN_SUBMISSION = True

# 2. Run the master pipeline
python scripts/run_pipeline.py
```

Option 2: Individual Scripts (CLI Override)

```
# Uses config.py settings by default, CLI can override
python scripts/train_models.py           # Uses config settings
python scripts/train_models.py --scenario 2 # Override scenario
python scripts/train_models.py --full    # Override to full mode
python scripts/generate_final_submissions.py # Uses SUBMISSION_MODEL from config
```

Option 3: Step-by-Step Manual

MANUAL WORKFLOW

1. SETUP
`pip install -r requirements.txt`
2. VIEW CONFIG
`python src/config.py`
3. EXPLORE DATA (optional)
`python notebooks/01_eda_visualization.py`

4. TRAIN MODELS
python scripts/train_models.py
5. CHECK RESULTS
Look at reports/model_comparison_scenario1.csv
6. GENERATE SUBMISSIONS
python scripts/generate_final_submissions.py
7. VALIDATE
python scripts/validate_submissions.py
8. SUBMIT
Upload submissions/*.csv to competition

 Current Results (Updated November 2025) 

Latest Model Comparison (Test Mode)

Scenario 1: (Predict months 0-23 with only pre-entry data)

Model	Final PE Score
Baseline-ExpDecay(0.020) ⚡	0.1483 ✓
Hybrid-Physics+LightGBM	0.2294
Baseline-NoErosion	0.2470
Hybrid-Physics+XGBoost	0.2716
ARHOW-SARIMAX+HW	0.2995
LightGBM	0.3432
XGBoost	0.6738

Scenario 2: (Predict months 6-23 with actual months 0-5)

Model	Final PE Score
Baseline-ExpDecay(0.020) ⚡	0.1580 ✓
Hybrid-Physics+LightGBM	0.2086
Hybrid-Physics+XGBoost	0.3258
Baseline-NoErosion	0.3289
LightGBM	0.3316
ARHOW-SARIMAX+HW	0.4062
XGBoost	0.6445

Key Findings:

- 1. **Exponential Decay Baseline wins both scenarios** - Simple physics-based approach outperforms complex ML
- 2. **Hybrid models are strong second place** - Physics + ML correction works well
- 3. **ARHOW model** - Time-series approach shows promise but needs tuning
- 4. **Pure ML models struggle** - XGBoost/LightGBM alone don't capture erosion patterns well

Recommendations:

- **Scenario 1:** Use Exponential Decay (PE=0.1483) or Hybrid-LightGBM (PE=0.2294)
- **Scenario 2:** Use Exponential Decay (PE=0.1580) or Hybrid-LightGBM (PE=0.2086)

 Output Files with Timestamps

All training runs and submissions now save with timestamps and JSON summaries.

Training Output Files:

```
reports/
├─ model_comparison_scenario1_20251128_210310.csv      ← Timestamped CSV
├─ model_comparison_scenario1.csv                      ← Latest (easy access)
├─ run_summary_scenario1_20251128_210310.json         ← Full JSON summary
├─ model_comparison_scenario2_20251128_210433.csv
├─ model_comparison_scenario2.csv
└─ run_summary_scenario2_20251128_210433.json
```

EDA Data Files: 

```
reports/eda_data/
├─ fig01_volume_distribution.json + .csv
├─ fig02_erosion_curves.json + .csv
├─ fig03_bucket_distribution.json + .csv
├─ fig04_generic_impact.json + .csv
├─ fig05_therapeutic_area.json + .csv
├─ fig06_monthly_patterns.json + .csv
├─ fig07_correlation_matrix.json + .csv
└─ fig08_country_analysis.json + .csv
```

Submission Output Files:

```
submissions/
├─ scenario1_baseline_20251128_185734.csv      ← Timestamped CSV
```

└─ scenario1_baseline_20251128_185734.json	← Summary JSON
└─ scenario1_baseline_final.csv	← Latest (easy access)

JSON Summary Contents:

Training Run Summary (`run_summary_*.json`):

```
{
  "run_info": {
    "run_id": "20251128_185725",
    "timestamp": "20251128_185725",
    "scenario": 1,
    "date": "2025-11-28",
    "time": "18:57:25"
  },
  "best_model": {
    "name": "Hybrid-Physics+LightGBM",
    "final_score": 1.0758
  },
  "all_results": [...],
  "config": {
    "paths": {...},
    "constants": {...},
    "metric_weights": {...},
    "model_params": {...}
  },
  "data_info": {
    "train_rows": 75024,
    "val_rows": 18720,
    "n_features": 40,
    "feature_cols": [...]
  },
  "feature_importance": [...]
}
```

Submission Summary (`scenario*_*.json`):

```
{
  "submission_info": {
    "scenario": 1,
    "model_type": "baseline",
    "timestamp": "20251128_185734",
    "date": "2025-11-28",
    "time": "18:57:34"
  },
  "data_stats": {
    "n_brands": 340,

```

```
    "n_rows": 8160,
    "months_predicted": [0, 1, ..., 23]
  },
  "volume_predictions": {
    "min": 25.84,
    "max": 126466938.0,
    "mean": 1909240.94,
    "median": 110694.60,
    "std": 7035711.05
  },
  "model_config": {
    "decay_rate": 0.05,
    "model_type": "baseline"
  },
  "full_config": {...}
}
```

🔑 Key Takeaways

- 1. **Bucket 1 is CRITICAL** - Double weighted, focus on high-erosion drugs
- 2. **Early months matter most** - 50% of score from first 6 months
- 3. **Simple models can win** - Exponential decay captures the physics
- 4. **Normalize everything** - All errors divided by pre-entry average
- 5. **Validate before submit** - One wrong format = rejected submission

📄 Quick Reference

NEW Config-Based Commands (Recommended)

Task	Command	Config Toggle
Show current config	<code>python src/config.py</code>	-
Run full pipeline	<code>python scripts/run_pipeline.py</code>	All <code>RUN_*</code> toggles
Train models only	<code>python scripts/train_models.py</code>	<code>RUN_SCENARIO</code> , <code>TEST_MODE</code> , <code>MODELS_ENABLED</code>
Generate submissions	<code>python scripts/generate_final_submissions.py</code>	<code>RUN_SCENARIO</code> , <code>SUBMISSION_MODEL</code>
Run EDA	<code>python notebooks/01_eda_visualization.py</code>	-
Validate submissions	<code>python scripts/validate_submissions.py</code>	-

CLI Override Commands (Legacy)

Task	Command
Train specific scenario	<code>python scripts/train_models.py --scenario 1</code>
Train in test mode	<code>python scripts/train_models.py --test</code>
Force full mode	<code>python scripts/train_models.py --no-test</code>

Available Model Types:

- `baseline_no_erosion` - No generic erosion (naive baseline)
- `baseline_exp_decay` - Exponential decay (best for both scenarios) ⌚
- `hybrid_lightgbm` - Physics + LightGBM hybrid (strong second place)
- `hybrid_xgboost` - Physics + XGBoost hybrid
- `arihow` - SARIMAX + Holt-Winters time-series ensemble NEW
- `lightgbm` - LightGBM gradient boosting
- `xgboost` - XGBoost gradient boosting

 Complete Pipeline: From Zero to Submission (Step-by-Step) NEW

This section provides a **complete, copy-paste ready** guide to run the entire pipeline from scratch.

Prerequisites

```
# 1. Navigate to project directory
cd D:\Datathon\novartis_datathon_2025\Main_project

# 2. Create virtual environment (first time only)
python -m venv saeed_venv

# 3. Activate virtual environment
.\saeed_venv\Scripts\Activate.ps1

# 4. Install dependencies (first time only)
pip install -r requirements.txt
```

Step 1: Verify Data Files

Ensure raw data files exist in `data/raw/` :

```
data/raw/
├── df_volume_train.csv
├── df_volume_test.csv
├── df_generics_train.csv
├── df_generics_test.csv
└── df_medicine_info.csv
```

Step 2: Configure Pipeline in `src/config.py`

Edit `src/config.py` to set your desired options:

```
# =====
# SECTION 1: RUN MODE SETTINGS (Most frequently changed)
# =====
RUN_SCENARIO = 1          # 1, 2, or [1, 2] for both
TEST_MODE = True          # True = fast (50 brands), False = full
TEST_MODE_BRANDS = 50     # Number of brands in test mode

# =====
# SECTION 2: MODEL ENABLE/DISABLE TOGGLES
# =====
MODELS_ENABLED = {
    'baseline_no_erosion': True,    # No erosion baseline
    'baseline_exp_decay': True,    # Exponential decay baseline ⌚
    'lightgbm': True,              # LightGBM
    'xgboost': True,              # XGBoost
    'hybrid_lightgbm': True,       # Physics + LightGBM
    'hybrid_xgboost': True,       # Physics + XGBoost
    'arihow': True,               # Time-series ensemble
}

# =====
# SECTION 3: PIPELINE STEP TOGGLES (for run_pipeline.py)
# =====
RUN_EDA = False           # Run EDA visualization
RUN_TRAINING = True       # Run model training
RUN_SUBMISSION = True     # Generate submissions
RUN_VALIDATION = True     # Validate submissions
```

Step 3: Verify Configuration

```
# Check your current configuration
python src/config.py
```

Step 4: Run Complete Pipeline

Option A: Master Pipeline Script (Recommended)

```
# Runs all enabled steps based on config
python scripts/run_pipeline.py
```

Option B: Individual Steps

```
# Step 4a: Run EDA (optional)
python notebooks/01_eda_visualization.py

# Step 4b: Train models
python scripts/train_models.py

# Step 4c: Generate submissions
python scripts/generate_final_submissions.py

# Step 4d: Validate submissions
python scripts/validate_submissions.py
```

Step 5: Review Results

```
# Check model comparison results
type reports\model_comparison_scenario1.csv
```

Or open `notebooks/03_model_results.ipynb` for visualizations.

Quick Reference Commands (Copy-Paste Ready)

```
# =====
# COMPLETE PIPELINE - CONFIG-BASED (RECOMMENDED)
# =====

# Activate environment
cd D:\Datathon\novartis_datathon_2025\Main_project
.\saeed_venv\Scripts\Activate.ps1

# 1. Check/edit config first
python src/config.py                # View current settings

# 2. Run master pipeline (uses config toggles)
python scripts/run_pipeline.py      # Runs EDA→Train→Submit→Validate

# Done! Files ready in submissions/ folder

# =====
# INDIVIDUAL STEPS (When you need more control)
```

```
# =====

# Run EDA only
python notebooks/01_eda_visualization.py

# Train models only (uses config.RUN_SCENARIO, TEST_MODE, MODELS_ENABLED)
python scripts/train_models.py

# Generate submissions only (uses config.RUN_SCENARIO, SUBMISSION_MODEL)
python scripts/generate_final_submissions.py

# Validate submissions only
python scripts/validate_submissions.py

# =====
# CLI OVERRIDE (For quick tests)
# =====

# Override scenario (ignores config.RUN_SCENARIO)
python scripts/train_models.py --scenario 2

# Override test mode
python scripts/train_models.py --test      # Force test mode
python scripts/train_models.py --no-test   # Force full mode
```

Common Configuration Presets

Edit `src/config.py` with these presets:

```
# === QUICK TEST (Before full run) ===
RUN_SCENARIO = 1
TEST_MODE = True
MODELS_ENABLED = {
    'baseline_no_erosion': False,
    'baseline_exp_decay': True, # Test best model only
    'lightgbm': False,
    'xgboost': False,
    'hybrid_lightgbm': False,
    'hybrid_xgboost': False,
    'arihow': False,
}
RUN_TRAINING = True
RUN_SUBMISSION = False
RUN_VALIDATION = False

# === FULL PRODUCTION RUN ===
RUN_SCENARIO = [1, 2] # Both scenarios
TEST_MODE = False     # All brands
MODELS_ENABLED = {all_models: True} # Enable all
RUN_EDA = True
```

```

RUN_TRAINING = True
RUN_SUBMISSION = True
RUN_VALIDATION = True

# === SUBMISSION GENERATION ONLY (Models already trained) ===
RUN_SCENARIO = [1, 2]
RUN_TRAINING = False
RUN_SUBMISSION = True
RUN_VALIDATION = True

```

🔗 Expected Final Output Structure

```

Main_project/
├── models/
│   ├── scenario1_lightgbm.joblib          # Latest model (always
│   └── updated)
│       ├── scenario1_lightgbm_20251128_220910.joblib  # Timestamped backup
│       ├── scenario1_xgboost.joblib
│       ├── scenario1_xgboost_20251128_220911.joblib
│       ├── scenario1_hybrid_hybrid.joblib
│       ├── scenario1_arihow_arihow.joblib
│       ├── scenario2_lightgbm.joblib
│       ├── scenario2_xgboost.joblib
│       ├── scenario2_hybrid_hybrid.joblib
│       └── scenario2_arihow_arihow.joblib
├── reports/
│   ├── model_comparison_scenario1.csv
│   ├── model_comparison_scenario1_20251128_220915.csv  # Timestamped
│   ├── model_comparison_scenario2.csv
│   ├── run_summary_scenario1_*.json
│   ├── run_summary_scenario2_*.json
│   └── eda_data/
│       ├── fig01-08_*.json
│       └── fig01-08_*.csv
└── submissions/
    ├── scenario1_baseline_final.csv  ← UPLOAD THIS
    ├── scenario2_baseline_final.csv  ← UPLOAD THIS
    └── scenario*_*.json

```

🔗 Recent Updates (November 2025) NEW

1. Config-Based Pipeline System NEW 💧

- **Centralized configuration** in `src/config.py` - All settings in one place
- **13 well-documented sections** covering run modes, model params, paths, etc.
- **Model enable/disable toggles** - Select which models to train via `MODELS_ENABLED` dict
- **Pipeline step toggles** - `RUN_EDA` , `RUN_TRAINING` , `RUN_SUBMISSION` , `RUN_VALIDATION`

- **Master orchestrator script** - `scripts/run_pipeline.py` runs complete pipeline based on config
- **CLI override still available** - Use `--scenario` , `--test` flags for quick overrides

2. Timestamped Model Saving NEW

- Models now save with timestamps: `scenario1_lightgbm_20251128_220910.joblib`
- Also saves a "latest" copy: `scenario1_lightgbm.joblib` (always updated)
- Reports also timestamped: `model_comparison_scenario1_20251128_220915.csv`
- Prevents accidental overwrites during experiments

3. ARHOW Model Improvements

- **Upgraded from basic ARIMA to SARIMAX** - Better handling of seasonal patterns
- **Added Holt-Winters (ExponentialSmoothing)** - Captures level and trend
- **Learned weights via Linear Regression** - Optimal combination: $y_{\text{hat}} = \beta_0 \times \text{ARIMA} + \beta_1 \times \text{HW}$
- **Fixed training on ALL brands** - Previously only trained on training brands
- **Added RangeIndex for statsmodels compatibility** - Prevents datetime index errors

4. Warning Suppression

- Added `warnings.catch_warnings()` context managers to suppress statsmodels convergence warnings
- Cleaner console output during training

5. EDA Data Export

- All EDA figures now export accompanying CSV files with raw data
- Enables custom visualizations and further analysis
- 9 JSON + 9 CSV files in `reports/eda_data/`

6. Pandas FutureWarning Fix

- Updated `feature_engineering.py` to use `include_groups=False` in groupby operations
- Prevents deprecation warnings in newer pandas versions

Good luck with the competition! 🍀