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

	Description
📥 **INPUT**	None (configuration file)
📤 **OUTPUT**	Python constants, paths, and parameters importable by other modules
🎯 **GOAL**	Centralize ALL settings so you can change them in ONE place

What it does: Stores ALL settings and constants in ONE place.

What's inside:

```
# WHERE to find files
PROJECT_ROOT = Path(__file__).parent.parent
DATA_RAW = PROJECT_ROOT / "data" / "raw"
MODELS_DIR = PROJECT_ROOT / "models"

# COMPETITION RULES
PRE_ENTRY_MONTHS = 12      # Use 12 months before generic entry
POST_ENTRY_MONTHS = 24     # Predict 24 months after
BUCKET_1_THRESHOLD = 0.25  # Below this = Bucket 1 (high erosion)

# SCORING WEIGHTS for Scenario 1
S1_SUM_0_5_WEIGHT = 0.5    # First 6 months = 50% of score!
S1_SUM_6_11_WEIGHT = 0.2   # Months 6-11 = 20%
S1_SUM_12_23_WEIGHT = 0.1  # Months 12-23 = 10%

# MODEL SETTINGS
LGBM_PARAMS = {...}  # LightGBM hyperparameters
XGB_PARAMS = {...}   # XGBoost hyperparameters
```

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 `volume` column and `months_pregx` (pre-entry months: -12 to -1)
📤 **OUTPUT**	`aux_bucket_avgvol.csv` file with columns: `country`, `brand_name`, `avg_vol`, `mean_erosion`, `bucket`
🎯 **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 + `aux_bucket_avgvol.csv` (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 (`.joblib`)
🎯 **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) 🆕

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)

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 (β_0, β_1) 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

	Description
📥 **INPUT**	Predictions DataFrame, Actuals DataFrame, `aux_bucket_avgvol.csv` (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:

```
PE = 0.2 × (monthly errors normalized)
   + 0.5 × (error in months 6-11 sum)     ← 50% WEIGHT!
   + 0.3 × (error in months 12-23 sum)
```

Final Score Calculation:

```
Final Score = (2 × avg_PE_bucket1 + 1 × avg_PE_bucket2) /
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 = (2×0.5×10 + 1×0.3×100) / (2×10 + 1×100) = (10 + 30) / 120 = 0.33

---

7 `src/submission.py` - The Output Generator

	Description
INPUT	Predictions DataFrame with columns: `country`, `brand_name`, `months_postgx`, `volume`
OUTPUT	CSV file formatted for competition upload (e.g., `scenario1_baseline_final.csv`)
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_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: `--scenario` (1 or 2), `--model` (baseline/lightgbm/xgboost)

	Description
📤 **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, `--scenario` argument (1 or 2)
📤 **OUTPUT**	Model files in `models/`, comparison CSV in `reports/`
🎯 **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, `--model` argument
📤 **OUTPUT**	`submissions/scenario1_*_final.csv`, `submissions/scenario2_*_final.csv`
🎯 **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 `submissions/` 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 🆕

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

```
┌─────────────────────────────────────────────────────────────┐
│                    YOUR WORKFLOW                            │
└─────────────────────────────────────────────────────────────┘

1. SETUP
   pip install -r requirements.txt

2. QUICK TEST
   python scripts/run_demo.py

3. EXPLORE DATA (optional)
   Open notebooks/01_eda_visualization.ipynb

4. TRAIN MODELS
   python scripts/train_models.py --scenario 1
   python scripts/train_models.py --scenario 2

5. CHECK RESULTS
   Look at reports/model_comparison_scenario1.csv
   Open notebooks/03_model_results.ipynb
```

```
6. GENERATE SUBMISSIONS
   python scripts/generate_final_submissions.py --model baseline

7. VALIDATE
   python scripts/validate_submissions.py

8. SUBMIT
   Upload submissions/*.csv to competition
```

Current Results (Updated November 2025) NEW

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

Task	Command
Test everything	`python scripts/run_demo.py`
Train models	`python scripts/train_models.py --scenario 1`
Generate submission (baseline)	`python scripts/generate_final_submissions.py --model baseline`
Generate submission (hybrid)	`python scripts/generate_final_submissions.py --model hybrid`
Validate submission	`python scripts/validate_submissions.py`
Run full pipeline	`python src/pipeline.py --scenario 1 --model lightgbm`

Available Model Types:

- `baseline` - Exponential decay (best for both scenarios)
- `hybrid` - Physics + ML hybrid (strong second place)
- `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) 🆕

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: Run EDA (Optional but Recommended)

```
# Generate EDA visualizations and data exports
python notebooks/01_eda_visualization.py
```

Output:

- `reports/eda_data/fig01-08_*.json` - Summary statistics
- `reports/eda_data/fig01-08_*.csv` - Raw data for each figure
- `reports/figures/fig01-08_*.png` - Visualization images

Step 3: Quick Demo Test

```
# Verify everything works with a quick test
python scripts/run_demo.py
```

Step 4: Train Models (Test Mode - Fast)

```
# Train Scenario 1 models (test mode - subset of data)
python scripts/train_models.py --scenario 1 --test

# Train Scenario 2 models (test mode - subset of data)
python scripts/train_models.py --scenario 2 --test
```

Output:

- `reports/model_comparison_scenario1.csv`
- `reports/model_comparison_scenario2.csv`
- `reports/run_summary_scenario*.json`
- `models/scenario*_*.joblib`

Step 5: Train Models (Full Mode - Production)

```
# Train Scenario 1 models (FULL - all data)
python scripts/train_models.py --scenario 1

# Train Scenario 2 models (FULL - all data)
python scripts/train_models.py --scenario 2
```

Step 6: Review Results

```
# Check model comparison results
type reports\model_comparison_scenario1.csv
type reports\model_comparison_scenario2.csv
```

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

Step 7: Generate Final Submissions

```
# Generate submissions using best model (baseline exponential decay)
python scripts/generate_final_submissions.py --model baseline
```

```
# OR generate submissions using hybrid model
python scripts/generate_final_submissions.py --model hybrid
```

Output:

- `submissions/scenario1_baseline_final.csv`
- `submissions/scenario2_baseline_final.csv`
- `submissions/scenario*_*.json` - Summary files

Step 8: Validate Submissions

```
# Validate submission files before upload
python scripts/validate_submissions.py
```

Checks performed:

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

Step 9: Upload to Competition

Upload the following files to the competition platform:

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

📋 Quick Reference Commands (Copy-Paste Ready)

```
# ============================================
# COMPLETE PIPELINE - RUN ALL STEPS
# ============================================

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

# Run EDA
python notebooks/01_eda_visualization.py

# Train models (full mode)
```

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

# Generate submissions
python scripts/generate_final_submissions.py --model baseline

# Validate
python scripts/validate_submissions.py

# Done! Files ready in submissions/ folder
```

🎯 Expected Final Output Structure

```
Main_project/
├── models/
│   ├── scenario1_lightgbm.joblib
│   ├── scenario1_xgboost.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_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) 🆕

1. 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_hat = β₀ × ARIMA + β₁ × HW`
- **Fixed training on ALL brands** - Previously only trained on training brands
- **Added RangeIndex for statsmodels compatibility** - Prevents datetime index errors

2. Warning Suppression

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

3. 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/`

4. 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! ⏳