Feature Engineering Comparison: Main_project vs novartis_datathon_2025-Arman

Executive Summary

This document provides a detailed comparison of feature engineering approaches between two implementations for the Novartis Datathon 2025. The **Main_project** takes a procedural, straightforward approach while **novartis_datathon_2025-Arman** implements a more modular, configuration-driven architecture with advanced features.

---

1. Architecture & Design Philosophy

Main_project

Aspect	Implementation
Code Size	~1,671 lines
Configuration	Python constants embedded in code
Pattern	Procedural functions
Leakage Prevention	Manual checks throughout code

Pros:

- ✅ Easier to understand and debug for newcomers
- ✅ All logic in one place - no need to trace through multiple files
- ✅ Quick to modify for experimentation
- ✅ Lower overhead for small-scale changes

Cons:

- ❌ Harder to maintain as codebase grows
- ❌ Configuration changes require code modifications
- ❌ No formal validation framework for feature correctness
- ❌ Risk of copy-paste errors when duplicating feature logic

novartis_datathon_2025-Arman

Aspect	Implementation
Code Size	~2,735 lines (63% larger)
Configuration	YAML config files
Pattern	Modular with Protocol pattern
Leakage Prevention	Schema-enforced with `FORBIDDEN_FEATURES` list

Pros:

- ☑ Highly modular and reusable components
- ☑ Configuration-driven - change behavior without code changes
- ☑ Strict leakage prevention with automated validation
- ☑ Better suited for production deployment
- ☑ Built-in feature selection and ablation tools

Cons:

- ❌ Steeper learning curve
- ❌ More files to navigate and understand
- ❌ Overhead may be unnecessary for quick experiments
- ❌ YAML configs can become complex

---

2. Pre-Entry Features

Main_project Implementation

```
# Features created:
- avg_vol                     # Pre-entry average from avg_j_df
- pre_entry_slope             # Linear trend before entry
- pre_entry_volatility        # Std dev of pre-entry volumes
- pre_entry_growth_rate       # Growth rate in pre-entry period
- pre_entry_min               # Minimum volume before entry
- pre_entry_max               # Maximum volume before entry
- pre_entry_last_volume       # Last volume at month -1
```

Pros:

- ☑ Simple and interpretable features
- ☑ Covers basic statistical properties

Cons:

- ❌ Single window approach (no multi-scale analysis)
- ❌ No log transforms for skewed distributions
- ❌ Missing ratio-based features

novartis_datathon_2025-Arman Implementation

```
# Features created:
- avg_vol_12m, avg_vol_6m, avg_vol_3m    # Multiple window averages
- pre_entry_trend                         # Linear slope with R²
- pre_entry_trend_norm                    # Normalized trend
- pre_entry_volatility                    # Std / avg_vol_12m
```

```
- pre_entry_max_ratio                     # max / avg_vol_12m
- pre_entry_min_ratio                     # min / avg_vol_12m
- pre_entry_range_ratio                   # (max - min) / avg_vol_12m
- volume_growth_rate                      # Overall growth rate
- vol_ratio_6m_12m                        # 6m avg / 12m avg
- vol_ratio_3m_12m                        # 3m avg / 12m avg
- vol_ratio_3m_6m                         # 3m avg / 6m avg
- log_avg_vol, log_avg_vol_6m, log_avg_vol_3m  # Log transforms

# UNIQUE - Seasonal Features:
- seasonal_amplitude                      # Max deviation from mean by month-
of-year
- seasonal_peak_month                     # Which month has highest volume
- seasonal_trough_month                   # Which month has lowest volume
- seasonal_peak_trough_ratio              # Peak-to-trough ratio
- seasonal_q1_effect through seasonal_q4_effect  # Quarter-wise deviations
```

Pros:

- ☑ Multi-scale analysis captures different temporal patterns
- ☑ Ratio features are scale-invariant
- ☑ Log transforms handle skewed distributions
- ☑ Seasonal features capture cyclical patterns unique to pharma
- ☑ Normalized features improve model convergence

Cons:

- ❌ More features increase risk of overfitting
- ❌ Seasonal features require sufficient pre-entry data
- ❌ Computational overhead from multiple window calculations

---

3. Time Features

Main_project Implementation

```
# Polynomial transforms:
- months_postgx_squared
- months_postgx_sqrt
- months_postgx_log
- months_postgx_cubed

# Period indicators:
- is_early_period            # months 0-5
- is_mid_period              # months 6-11
- is_late_period             # months 12-17
- is_equilibrium             # months 18+

# Metric-aligned periods:
```

```
- is_first_6_months          # For Scenario 1 metric
- is_months_6_11             # For Scenario 2 metric
- is_months_12_plus          # Post-metric period

# Other:
- time_bucket + time_bucket_encoded
- decay_factor
- decay_phase
```

Pros:

- ☑ Polynomial transforms capture non-linear time effects
- ☑ Period indicators align with competition metrics
- ☑ Simple to interpret and explain

Cons:

- ❌ No cyclical encoding for seasonal patterns
- ❌ No explicit exponential decay modeling
- ❌ Missing calendar-based features (month-of-year, quarter)

novartis_datathon_2025-Arman Implementation

```
# Polynomial transforms:
- months_postgx_sq
- months_postgx_cube
- sqrt_months_postgx

# Period indicators:
- is_post_entry
- is_early, is_mid, is_late

# UNIQUE - Cyclical encoding:
- month_sin                  # sin(2π × month / 12)
- month_cos                  # cos(2π × month / 12)

# Calendar features:
- calendar_month
- is_q1, is_q2, is_q3, is_q4
- is_year_end
- is_year_start

# UNIQUE - Explicit decay modeling:
- time_decay                 # exp(-0.1 × months_postgx)
- time_decay_fast            # exp(-0.2 × months_postgx)
```

Pros:

- ☑ Cyclical encoding captures periodic patterns without discontinuities
- ☑ Explicit decay modeling matches pharma erosion patterns
- ☑ Calendar features capture real-world seasonality
- ☑ Multiple decay rates allow model to learn optimal decay

Cons:

- ❌ Cyclical encoding less interpretable than binary indicators
- ❌ Fixed decay rates may not be optimal for all products

---

4. Competition / Generics Features

Main_project Implementation

```
# Basic features:
- n_gxs_capped            # Capped at 15 (99th percentile)
- n_gxs_log
- n_gxs_squared
- has_generics            # Binary indicator
- high_competition        # n_gxs > threshold

# Temporal evolution:
- n_gxs_cummax            # Cumulative maximum
- n_gxs_change            # Month-over-month change
- n_gxs_change_3m         # 3-month change

# Brand-level:
- max_n_gxs_post          # Maximum n_gxs post-LOE
- months_with_generics
- competition_intensity
```

Pros:

- ☑ Capping handles outliers effectively
- ☑ Captures both static and dynamic competition aspects
- ☑ Competition intensity is intuitive metric

Cons:

- ❌ No forward-looking competition features
- ❌ Missing granular competition response metrics
- ❌ No per-generic erosion analysis

novartis_datathon_2025-Arman Implementation

```
# Basic features:
- has_generic
- multiple_generics         # n_gxs >= 2
- many_generics             # n_gxs >= 5
- log_n_gxs
- n_gxs_bin                 # Categorical: none/one/few/several/many

# Entry timing:
- n_gxs_at_entry
- n_gxs_pre_cutoff_max
- first_generic_month
- months_since_first_generic
- had_generic_pre_entry
- generic_entry_speed

# UNIQUE - Future generics (exogenous):
- n_gxs_at_month_12         # Known future value
- n_gxs_at_month_23         # Known future value
- n_gxs_change_to_12        # Change from entry to month 12
- n_gxs_change_to_23        # Change from entry to month 23
- n_gxs_max_forecast        # Maximum expected generics
- expected_new_generics     # Anticipated new entrants
```

Pros:

- ☑ Future n_gxs features leverage known exogenous information
- ☑ Granular competition timing features
- ☑ Categorical binning provides non-linear effects
- ☑ Entry speed captures market dynamics

Cons:

- ✖ Future features require careful handling in production
- ✖ More complex feature set to maintain
- ✖ Some features may be highly correlated

---

5. Scenario 2 Early Erosion Features

Main_project Implementation

```
# First 6 months (months 0-5):
- mean_vol_0_5              # Average volume
- slope_0_5                 # Linear trend
- last_vol_5                # Volume at month 5
- std_vol_0_5               # Volatility
- min_vol_0_5               # Minimum volume
- pct_drop_0_5              # Percentage decline
```

```
- n_gxs_month_5             # Generics at month 5
- mean_n_gxs_0_5            # Average generics
```

Pros:

- ☑ Straightforward early erosion signal
- ☑ Covers basic statistical properties
- ☑ Direct alignment with Scenario 2 requirements

Cons:

- ✖ Single window approach
- ✖ No sub-window analysis
- ✖ Missing recovery detection

novartis_datathon_2025-Arman Implementation

```
# Multi-window analysis:
- avg_vol_0_5, erosion_0_5, trend_0_5, trend_0_5_norm
- drop_month_0              # Immediate drop at LOE
- avg_vol_0_2, avg_vol_3_5  # Sub-window averages
- month_0_to_3_change       # Early-period change rate
- month_3_to_5_change       # Mid-period change rate
- erosion_0_2, erosion_3_5  # Sub-window erosion rates

# UNIQUE - Recovery detection:
- recovery_signal           # Binary: vol[3-5] > vol[0-2]
- recovery_magnitude        # Degree of recovery

# UNIQUE - Competition response:
- competition_response      # n_gxs change from month 0 to 5
- erosion_per_generic       # Erosion divided by n_gxs
```

Pros:

- ☑ Sub-window analysis captures non-linear erosion patterns
- ☑ Recovery detection identifies stabilizing products
- ☑ Erosion per generic quantifies competition impact
- ☑ Better suited for complex erosion trajectories

Cons:

- ✖ More features increase complexity
- ✖ Recovery signal may be noisy for volatile products
- ✖ Requires careful feature selection

---

6. Lag & Rolling Features

Main_project Implementation

```
# Lag windows: [1, 3, 6, 12]
- volume_lag_1, volume_lag_3, volume_lag_6, volume_lag_12
- volume_diff_1, volume_diff_3, volume_diff_6, volume_diff_12
- volume_pct_change_1, volume_pct_change_3, ...

# Rolling windows: [3, 6, 12]
- volume_rolling_mean_3, volume_rolling_mean_6, volume_rolling_mean_12
- volume_rolling_std_3, volume_rolling_std_6, volume_rolling_std_12
- volume_rolling_min_3, volume_rolling_min_6, volume_rolling_min_12
- volume_rolling_max_3, volume_rolling_max_6, volume_rolling_max_12

# Erosion:
- erosion_rate_3m
```

Pros:

- ☑ Comprehensive rolling statistics
- ☑ Standard time-series features
- ☑ Multiple window sizes capture different dynamics

Cons:

- ❌ Not designed for deep learning models
- ❌ Missing momentum/acceleration features
- ❌ No volatility ratio features

novartis_datathon_2025-Arman Implementation

```
# UNIQUE - Sequence features (for CNN-LSTM):
- seq_volume_lag_1, seq_volume_lag_2, seq_volume_lag_3, seq_volume_lag_6
- seq_volume_ma_3, seq_volume_ma_6, seq_volume_ma_12
- seq_volume_diff_1, seq_volume_diff_3

# UNIQUE - Momentum features:
- seq_momentum_3          # (vol - vol_lag_3) / vol_lag_3
- seq_momentum_6          # (vol - vol_lag_6) / vol_lag_6

# UNIQUE - Higher-order features:
- seq_acceleration        # Second-order difference
- seq_volatility_3        # rolling_std_3 / rolling_mean_3
- seq_volatility_6        # rolling_std_6 / rolling_mean_6
```

Pros:

- ☑ Purpose-built for deep learning (CNN-LSTM)
- ☑ Momentum features capture rate of change
- ☑ Acceleration detects inflection points
- ☑ Volatility ratios are scale-invariant

Cons:

- ❌ Sequence builder adds complexity
- ❌ May be overkill for tree-based models
- ❌ Higher memory requirements

---

7. Interaction Features

Main_project Implementation

```
# Time × Competition:
- time_x_competition        # months_postgx × has_generics
- time_x_n_gxs_log          # months_postgx × log(n_gxs)

# Time × Drug characteristics:
- time_x_hospital           # months_postgx × hospital_rate
- biological_x_months       # is_biological × months_postgx

# Competition × Drug characteristics:
- competition_x_hospital    # has_generics × hospital_rate

# Therapeutic area:
- ther_erosion_x_time       # ther_area_erosion × months_postgx
- high_erosion_early        # is_high_erosion_area × is_early
- early_high_competition    # is_early × high_competition
```

Pros:

- ☑ Hand-crafted based on domain knowledge
- ☑ Captures known pharmaceutical dynamics
- ☑ Interpretable interactions

Cons:

- ❌ Manual feature engineering required
- ❌ May miss important interactions
- ❌ Hardcoded therapeutic area rankings

novartis_datathon_2025-Arman Implementation

```yaml
# Configurable via YAML:
interaction_features:
  - ["is_biological", "n_gxs"]        → biological_x_n_gxs
  - ["hospital_rate", "months_postgx"] → hospital_rate_x_time
  - ["ther_area_encoded", "erosion_0_5"] → ther_area_x_early_erosion
  - ["ther_area_erosion", "months_postgx"] → ther_area_erosion_x_time
```

Pros:

- ☑ Configuration-driven - easy to experiment
- ☑ Consistent naming convention
- ☑ Can be extended without code changes

Cons:

- ❌ YAML configs can become unwieldy
- ❌ Less explicit than hardcoded interactions
- ❌ Requires documentation for each interaction

---

8. Target Encoding

Main_project Implementation

```python
# CV-based target encoding:
def target_encode(df, column, target, n_folds=5, smoothing=10):
    """
    Cross-validated target encoding with smoothing
    """
    global_mean = df[target].mean()
    # K-fold encoding to prevent leakage
    # Smoothing: (count × mean + smoothing × global_mean) / (count +
smoothing)
```

Pros:

- ☑ Cross-validation prevents train-set leakage
- ☑ Smoothing handles low-cardinality categories
- ☑ Simple implementation

Cons:

- ❌ Not series-aware (may leak across brands)
- ❌ Fixed smoothing parameter
- ❌ Applied to all categorical columns equally

novartis_datathon_2025-Arman Implementation

```python
# K-fold leakage-safe target encoding:
class TargetEncoder:
    """
    Series-level split to prevent leakage
    Configurable via YAML
    Produces *_erosion_prior columns
    """
    def fit_transform(self, X, y, groups):
        # Split by brand to ensure no brand appears in both folds
        # Apply smoothing based on category frequency
```

Pros:

- ☑ Series-level split prevents temporal leakage
- ☑ YAML-configurable parameters
- ☑ Explicit naming (`*_erosion_prior`)
- ☑ Validation functions included

Cons:

- ❌ More complex implementation
- ❌ Requires group information
- ❌ May reduce encoded feature quality with small datasets

---

9. Sample Weighting

Main_project Implementation

```python
# Bucket-based weights:
bucket_weights = {
    1: 4.0,  # Bucket 1 predictions weighted 4×
    2: 1.0   # Bucket 2 predictions weighted 1×
}

# Time-window weights (Scenario 1):
time_weights = {
    (0, 5): 2.5,    # Early months get 2.5×
    (6, 11): 1.5,   # Mid months get 1.5×
    (12, 17): 1.0,  # Later months get 1×
    (18, 23): 0.75  # Final months get 0.75×
}

# Combined:
sample_weight = bucket_weight × time_weight
```

Pros:

- ☑ Built into feature engineering module
- ☑ Explicit weighting strategy
- ☑ Aligns with competition metric priorities
- ☑ Easy to understand and modify

Cons:

- ❌ Hardcoded weights may not be optimal
- ❌ Mixes concerns (features vs training)
- ❌ Not validated against metric improvement

novartis_datathon_2025-Arman Implementation

- Sample weighting handled in training modules, not feature engineering
- Separation of concerns: features are model-agnostic

Pros:

- ☑ Clean separation of concerns
- ☑ Features can be reused across different training strategies
- ☑ Weighting can be model-specific

Cons:

- ❌ Weighting logic distributed across modules
- ❌ May require coordination between teams

---

10. Unique Features Summary

Unique to Main_project

Feature	Description	Use Case
Horizon-as-Row	`expand_to_all_horizons()` function	Direct multi-step forecasting
Brand Static Features	`create_brand_static_features()`	Pre-aggregated brand-level features
Therapeutic Area Rankings	Hardcoded from EDA analysis	Domain-specific erosion priors
Multi-Config Grid Search	Built-in hyperparameter search	Automated experimentation
Integrated Sample Weights	Bucket × time weights	Metric-aligned training

Unique to novartis_datathon_2025-Arman

Feature	Description	Use Case
Sequence Builder Module	Full CNN-LSTM preparation	Deep learning models
Seasonal Features	Amplitude, peak/trough, quarterly effects	Cyclical pattern capture
Future n_gxs Features	`n_gxs_at_month_12` , `n_gxs_at_month_23`	Exogenous variable leverage
Recovery Detection	`recovery_signal` , `recovery_magnitude`	Stabilization identification
Cyclical Time Encoding	`month_sin` , `month_cos`	Continuous periodic features
Explicit Decay Modeling	`time_decay` , `time_decay_fast`	Pharma erosion patterns
Feature Selection Tools	Correlation, importance, ablation	Automated feature reduction
Feature Caching	Parquet-based persistence	Development efficiency
FORBIDDEN_FEATURES Schema	Automatic leakage detection	Production safety
Frequency Encoding	Count-based categorical encoding	High-cardinality handling
Feature Scaler Class	Standard/MinMax/Robust scaling	Preprocessing flexibility

11. Recommendations

When to Use Main_project Approach

1. **Rapid Prototyping**: Quick experiments with new ideas
2. **Small Teams**: When one person maintains the codebase
3. **Interpretability Focus**: When stakeholders need clear explanations
4. **Tree-Based Models Only**: XGBoost, LightGBM, CatBoost
5. **Limited Compute**: When memory/CPU is constrained

When to Use novartis_datathon_2025-Arman Approach

1. **Production Deployment**: Robust validation and caching
2. **Deep Learning**: CNN-LSTM or hybrid architectures
3. **Large Teams**: Clear interfaces and documentation
4. **Experimentation at Scale**: YAML-driven configuration
5. **Automated Pipelines**: Feature selection and ablation built-in

Hybrid Approach (Best of Both)

Consider combining:

- Main_project's sample weighting strategy
- Arman's seasonal and cyclical features

- Arman's future n_gxs features (if allowed by competition rules)
- Main_project's therapeutic area rankings with Arman's target encoding
- Arman's leakage prevention framework
- Main_project's horizon-as-row for direct forecasting with Arman's sequence builder for hybrid models

---

12. Performance Implications

Aspect	Main_project	novartis_datathon_2025-Arman
Feature Count	~50-70 features	~100-150 features
Memory Usage	Lower	Higher (sequences, caching)
Computation Time	Faster	Slower (more transforms)
Overfitting Risk	Lower	Higher (needs regularization)
Model Compatibility	Tree-based	Tree-based + Deep Learning
Maintenance Effort	Medium	High
Extensibility	Medium	High

Appendix: Feature Lists

Main_project Full Feature List

▶ Click to expand

```
Pre-Entry Features:
- avg_vol
- pre_entry_slope
- pre_entry_volatility
- pre_entry_growth_rate
- pre_entry_min
- pre_entry_max
- pre_entry_last_volume

Time Features:
- months_postgx_squared
- months_postgx_sqrt
- months_postgx_log
- months_postgx_cubed
- is_early_period
- is_mid_period
- is_late_period
- is_equilibrium
- is_first_6_months
- is_months_6_11
- is_months_12_plus
- time_bucket
```

```
- time_bucket_encoded
- decay_factor
- decay_phase

Competition Features:
- n_gxs_capped
- n_gxs_log
- n_gxs_squared
- n_gxs_cummax
- n_gxs_change
- n_gxs_change_3m
- has_generics
- high_competition
- months_with_generics
- competition_intensity
- max_n_gxs_post

Drug Features:
- hospital_rate_bucket
- is_high_hospital_rate
- is_retail_focused
- hospital_rate_squared
- ther_area_erosion_rank
- ther_area_mean_erosion
- is_high_erosion_area
- is_low_erosion_area

Scenario 2 Features:
- mean_vol_0_5
- slope_0_5
- last_vol_5
- std_vol_0_5
- min_vol_0_5
- pct_drop_0_5
- n_gxs_month_5
- mean_n_gxs_0_5

Lag/Rolling Features:
- volume_lag_1, volume_lag_3, volume_lag_6, volume_lag_12
- volume_diff_1, volume_diff_3, volume_diff_6, volume_diff_12
- volume_pct_change_1, volume_pct_change_3, ...
- volume_rolling_mean_3, volume_rolling_mean_6, volume_rolling_mean_12
- volume_rolling_std_3, volume_rolling_std_6, volume_rolling_std_12
- volume_rolling_min_3, volume_rolling_min_6, volume_rolling_min_12
- volume_rolling_max_3, volume_rolling_max_6, volume_rolling_max_12
- erosion_rate_3m

Interaction Features:
- time_x_competition
- time_x_n_gxs_log
- time_x_hospital
- competition_x_hospital
- early_high_competition
- biological_x_months
```

```
- ther_erosion_x_time
- high_erosion_early
```

novartis_datathon_2025-Arman Full Feature List

▶ Click to expand

```
Pre-Entry Features:
- avg_vol_12m, avg_vol_6m, avg_vol_3m
- pre_entry_trend
- pre_entry_trend_norm
- pre_entry_volatility
- pre_entry_max_ratio
- pre_entry_min_ratio
- pre_entry_range_ratio
- volume_growth_rate
- vol_ratio_6m_12m
- vol_ratio_3m_12m
- vol_ratio_3m_6m
- log_avg_vol, log_avg_vol_6m, log_avg_vol_3m
- seasonal_amplitude
- seasonal_peak_month
- seasonal_trough_month
- seasonal_peak_trough_ratio
- seasonal_q1_effect, seasonal_q2_effect, seasonal_q3_effect,
seasonal_q4_effect

Time Features:
- months_postgx_sq
- months_postgx_cube
- sqrt_months_postgx
- is_post_entry
- is_early, is_mid, is_late
- month_sin, month_cos
- calendar_month
- is_q1, is_q2, is_q3, is_q4
- is_year_end, is_year_start
- time_decay
- time_decay_fast

Competition Features:
- has_generic
- multiple_generics
- many_generics
- log_n_gxs
- n_gxs_bin
- n_gxs_at_entry
- n_gxs_pre_cutoff_max
- first_generic_month
- months_since_first_generic
```

```
- had_generic_pre_entry
- generic_entry_speed
- n_gxs_at_month_12
- n_gxs_at_month_23
- n_gxs_change_to_12
- n_gxs_change_to_23
- n_gxs_max_forecast
- expected_new_generics

Drug Features:
- hospital_rate_norm
- hospital_rate_bin
- is_hospital_drug
- is_retail_drug
- is_injection
- is_oral
- ther_area_encoded
- country_encoded
- main_package_encoded

Scenario 2 Features:
- avg_vol_0_5, erosion_0_5, trend_0_5, trend_0_5_norm
- drop_month_0
- avg_vol_0_2, avg_vol_3_5
- month_0_to_3_change
- month_3_to_5_change
- erosion_0_2, erosion_3_5
- recovery_signal
- recovery_magnitude
- competition_response
- erosion_per_generic

Sequence Features:
- seq_volume_lag_1, seq_volume_lag_2, seq_volume_lag_3, seq_volume_lag_6
- seq_volume_ma_3, seq_volume_ma_6, seq_volume_ma_12
- seq_volume_diff_1, seq_volume_diff_3
- seq_momentum_3, seq_momentum_6
- seq_acceleration
- seq_volatility_3, seq_volatility_6

Interaction Features:
- biological_x_n_gxs
- hospital_rate_x_time
- ther_area_x_early_erosion
- ther_area_erosion_x_time

Collaboration Features:
- collab_country_erosion_prior
- collab_ther_area_erosion_prior
- collab_hospital_erosion_prior
- collab_package_erosion_prior

Visibility Features:
- vis_avg_inventory
```

```
    - vis_avg_days_of_supply
    - vis_avg_stock_out_risk
    - vis_fill_rate
    - vis_avg_lead_time
    - vis_supplier_reliability
    - vis_on_time_delivery
    - vis_capacity_utilization
```

Document created: November 29, 2025 Last updated: November 29, 2025