🔬 Data Preprocessing to Model Selection Rationale

Market Mix Modeling with XGBoost

> **This document explains the logical connection between specific data preprocessing techniques and the choice of XGBoost as the final model, demonstrating why each preprocessing step is essential and how it informs model selection.**

Table of Contents

---

1. Executive Summary

The Core Question

Why use these specific preprocessing techniques, and why does XGBoost emerge as the optimal model?

Quick Answer

Preprocessing Technique	Problem Solved	How It Enables XGBoost
Prophet Decomposition	Separates time patterns from marketing effects	Provides clean trend/seasonality features XGBoost can learn from
Adstock Transformation	Models advertising carryover effects	Converts raw spend into accumulated impact features
One-Hot Encoding	Converts categorical events to numerical	Makes events usable as binary features for tree splits

Model Selection Rationale

```
┌──────────────────────────────────────────────────────────────────┐
│  After preprocessing, the data has:                              │
│  ✓ Non-linear relationships (diminishing returns)                │
│  ✓ Feature interactions (channel synergies)                      │
│  ✓ Mixed feature types (continuous + binary)                     │
│  ✓ No clear linear structure                                     │
│                                                                  │
│  → XGBoost is optimal for this transformed data structure        │
└──────────────────────────────────────────────────────────────────┘
```

---

2. Data Characteristics & Challenges

2.1 Original Data Structure

The raw data (`MMM_data.csv`) contains:

Feature Type	Variables	Challenge
Target	`revenue`	Weekly aggregated sales
Media Spend	`tv_S` , `ooh_S` , `print_S` , `facebook_S` , `search_S`	Raw spend doesn't equal immediate impact
Organic	`newsletter`	Different response dynamics
External	`competitor_sales_B`	Context variable
Categorical	`events`	Promotional events (non-numeric)
Temporal	`DATE`	Weekly time series (208 observations)

2.2 Key Challenges Requiring Specific Preprocessing

Challenge 1: Time Series Components Mixed with Marketing Effects

Problem:

```
Revenue = f(Trend, Seasonality, Holidays, Marketing, Events)
```

Without separation, a model might attribute seasonal revenue spikes to marketing spend that coincidentally increased during Q4.

Solution: Prophet decomposition to isolate time-based patterns.

Challenge 2: Advertising Carryover Effect

Problem:

- TV ad aired in Week 1 still influences sales in Week 2, 3, 4...
- Raw spend in Week 5 ≠ Total advertising impact in Week 5
- Different channels have different decay rates

Example:

Week	TV Spend	Actual TV Impact (with carryover)
1	$100K	$100K
2	$0	$50K (carryover from Week 1)
3	$0	$25K (continued decay)
4	$80K	$80K + $12.5K = $92.5K

Solution: Adstock transformation with channel-specific decay rates.

Challenge 3: Non-Linear Relationships

Problem: Marketing has **diminishing returns**:

- First $50K on TV generates $200K revenue
- Next $50K generates only $100K revenue
- After $150K, additional spend yields minimal gains

```
Revenue
  ↑
  |      ___________    ← Saturation
  |    /
  |   /
  |  /
  |_/________________→ Spend
```

Solution: A model capable of learning non-linear response curves.

Challenge 4: Feature Interactions

Problem: Channels work together (synergy effects):

- TV + Facebook together > TV alone + Facebook alone
- Search captures demand created by TV

Solution: A model that automatically detects interactions.

Challenge 5: Categorical Events

Problem: The `events` column contains text values: `event1`, `event2`, `na`

Solution: One-hot encoding for tree-based model compatibility.

---

3. Preprocessing Pipeline Overview

Complete Pipeline Flow

```
┌─────────────────────────────────────────────────────────────────┐
│                         ORIGINAL DATA                           │
│  revenue, tv_S, ooh_S, print_S, facebook_S, search_S,          │
│  newsletter, competitor_sales_B, events, DATE                   │
└─────────────────────────────────────────────────────────────────┘
                            │
                            ▼
┌─────────────────────────────────────────────────────────────────┐
│  STEP 1: PROPHET DECOMPOSITION                                  │
│  • Extracts: trend, season, holiday, events                     │
│  • Why: Isolate time patterns from marketing signal             │
└─────────────────────────────────────────────────────────────────┘
                            │
                            ▼
┌─────────────────────────────────────────────────────────────────┐
│  STEP 2: ADSTOCK TRANSFORMATION                                 │
│  • Transforms: tv_S, ooh_S, print_S, facebook_S, search_S,     │
│               newsletter                                         │
│  • Why: Model advertising carryover/decay effects               │
└─────────────────────────────────────────────────────────────────┘
                            │
                            ▼
┌─────────────────────────────────────────────────────────────────┐
│  STEP 3: ONE-HOT ENCODING                                       │
│  • Transforms: events → events_event2, events_na                │
│  • Why: Convert categorical to binary for tree splits           │
└─────────────────────────────────────────────────────────────────┘
                            │
                            ▼
┌─────────────────────────────────────────────────────────────────┐
│                   PREPROCESSED FEATURES                         │
│  trend, season, holiday, events (Prophet)                       │
│  tv_S, ooh_S, print_S, facebook_S, search_S, newsletter        │
│  (all with adstock applied)                                     │
│  competitor_sales_B                                             │
└─────────────────────────────────────────────────────────────────┘
                            │
                            ▼
┌─────────────────────────────────────────────────────────────────┐
```

```
│                         XGBOOST MODEL                                │
│  Why: Handles non-linear relationships, interactions,                │
│       mixed feature types, and provides SHAP interpretability        │
└──────────────────────────────────────────────────────────────────────┘
```

---

4. Preprocessing Step 1: Prophet Time Series Decomposition

4.1 What Prophet Does

Prophet decomposes revenue into interpretable components:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

Where:

- $g(t)$ = **Trend** (long-term growth/decline)
- $s(t)$ = **Seasonality** (yearly patterns)
- $h(t)$ = **Holiday effects** (demand shocks)
- ϵ_t = Residual (what marketing explains)

4.2 Why Prophet Specifically?

Alternative	Why Not Used
Manual seasonal dummies	Can't capture complex seasonal shapes
STL decomposition	Doesn't handle holidays or events
ARIMA	Focuses on forecasting, not decomposition
Prophet ✓	Handles holidays, events, multiple seasonalities

4.3 Prophet Configuration Choices

```
Prophet(
    yearly_seasonality=True,    # Capture annual patterns
    weekly_seasonality=True,    # Capture weekly patterns
    daily_seasonality=False,    # Data is weekly, not daily
    holidays=holidays_weekly_us, # US holiday calendar
    seasonality_mode="multiplicative",  # % change, not absolute
)
```

Why Multiplicative Seasonality?

Mode	Formula	When to Use
Additive	$y = trend + season$	Constant seasonal fluctuation
Multiplicative	$y = trend \times season$	Seasonal effect scales with trend

Marketing data typically shows multiplicative seasonality: holiday spikes are **proportionally larger** when baseline sales are higher.

4.4 How This Enables XGBoost

After Prophet preprocessing:

Feature	Type	What XGBoost Learns
`trend`	Continuous	Baseline revenue trajectory
`season`	Continuous	Annual pattern multiplier
`holiday`	Continuous	Holiday impact adjustment
`events`	Continuous	Marketing event effects

Without Prophet: XGBoost might attribute December revenue spike to TV ads rather than Christmas.

With Prophet: Trend/season effects are explicitly provided, so XGBoost focuses on **incremental marketing contribution**.

---

5. Preprocessing Step 2: Adstock Transformation

5.1 The Mathematical Model

Geometric adstock transformation:

$$Adstock_t = Spend_t + \alpha \times Adstock_{t-1}$$

Where α is the **decay rate** (0 to 1):

- $\alpha = 0$: No carryover (immediate effect only)
- $\alpha = 0.5$: 50% of last week's effect carries over
- $\alpha = 0.9$: 90% carryover (very long memory)

5.2 Why Channel-Specific Decay Rates?

Different advertising media have different "memory":

Channel	α Range	Reasoning
TV	0.3-0.8	Brand building, long recall
Out-of-Home	0.1-0.4	Visual memory, moderate recall

Channel	α Range	Reasoning
Print	0.1-0.4	Physical presence, can be re-read
Facebook	0.0-0.4	Digital, faster consumption
Search	0.0-0.3	Intent-based, immediate conversion
Newsletter	0.1-0.4	Email can be read later

5.3 Implementation as sklearn Transformer

```
class AdstockGeometric(BaseEstimator, TransformerMixin):
    def __init__(self, alpha=0.5):
        self.alpha = alpha

    def transform(self, X):
        x_decayed = np.zeros_like(X)
        x_decayed[0] = X[0]
        for xi in range(1, len(x_decayed)):
            x_decayed[xi] = X[xi] + self.alpha * x_decayed[xi - 1]
        return x_decayed
```

Why sklearn Transformer Pattern?

Benefit	Explanation
Pipeline compatible	Can chain with other transformers
Optuna compatible	Alpha can be tuned as hyperparameter
Reproducible	`fit_transform()` ensures consistent application

5.4 How This Enables XGBoost

Without Adstock	With Adstock
Week 5 TV spend = $0	Week 5 TV impact = $12K (carryover)
Model sees no TV effect	Model sees accumulated effect
Underestimates TV ROI	Accurate TV attribution

Key Insight: Adstock transforms **spend** (an input) into **accumulated advertising pressure** (what actually drives sales).

XGBoost then learns the relationship:

```
Revenue = f(accumulated_tv_pressure, accumulated_facebook_pressure, ...)
```

Rather than:

```
Revenue = f(raw_tv_spend, raw_facebook_spend, ...)  ← Misleading
```

---

6. Preprocessing Step 3: One-Hot Encoding

6.1 Why Needed

The `events` column contains categorical values:

```
events: ['na', 'event1', 'event2', 'na', 'event2', ...]
```

XGBoost **cannot process text directly**—it requires numerical features.

6.2 One-Hot Encoding Process

```
pd.get_dummies(df["events"], drop_first=True, prefix="events")
```

Original `events`	`events_event2`	`events_na`
event1	0	0
event2	1	0
na	0	1

Why `drop_first=True`?

Prevents **multicollinearity**: If we know `events_event2=0` and `events_na=0`, we know it's `event1`.

6.3 How This Enables XGBoost

Tree-based models split on binary features effectively:

```
                    [revenue prediction]
                            |
                     events_event2 ≤ 0.5?
```

```
              /                    \
            YES                     NO
            /                         \
    [lower revenue]          [higher revenue]
    (no event2)              (event2 active)
```

One-hot encoding creates clean binary splits for categorical effects.

---

7. Why XGBoost Was Chosen

7.1 Matching Data Characteristics to Model Capabilities

After preprocessing, the data has specific characteristics that align with XGBoost's strengths:

Data Characteristic	Model Requirement	XGBoost Capability
Non-linear diminishing returns	Handle non-linear relationships	☑ Tree-based, captures any shape
Channel interactions	Detect feature interactions	☑ Automatic interaction detection
Mixed feature types	Handle continuous + binary	☑ Native support
Small dataset (208 rows)	Avoid overfitting	☑ Strong regularization options
Interpretability needed	Explain predictions	☑ SHAP compatible

7.2 XGBoost's Key Advantages for MMM

Advantage 1: Non-Linear Response Curves

Marketing has **diminishing returns**—XGBoost naturally captures this:

```
Linear Model:                  XGBoost:
Revenue                        Revenue
  ↑                              ↑
  |    /                         |      __________
  |   /                          |    /
  |  /                           |   /
  | /                            |  /
  |/________________→            |_/_______________→
                Spend                          Spend
```

Advantage 2: Automatic Feature Interactions

XGBoost detects channel synergies without manual specification:

```
If tv_S > 50K AND facebook_S > 20K:
    → Revenue boost 15% higher than sum of individual effects
```

Advantage 3: Regularization for Small Datasets

With only 208 weekly observations, overfitting is a risk.

XGBoost's regularization parameters:

- `reg_alpha` (L1): Shrinks irrelevant feature weights to zero
- `reg_lambda` (L2): Penalizes large weights
- `gamma` : Requires minimum gain for splits
- `max_depth` : Limits tree complexity

Advantage 4: SHAP Interpretability

XGBoost works seamlessly with SHAP for model explanation:

```
explainer = shap.TreeExplainer(xgboost_model)
shap_values = explainer.shap_values(X)
```

This enables:

- **Feature importance ranking**
- **Spend vs. effect share analysis**
- **Response curve visualization**

7.3 XGBoost Hyperparameters Tuned via Optuna

Parameter	Range	Purpose
`n_estimators`	5-100	Number of trees
`max_depth`	4-7	Tree depth (controls complexity)
`learning_rate`	0.001-0.1	Step size
`subsample`	0.5-1.0	Row sampling (prevents overfitting)
`colsample_bytree`	0.5-1.0	Column sampling
`reg_alpha` , `reg_lambda`	0-1	Regularization strength
`gamma`	0-1	Minimum loss reduction

---

8. Alternative Models Considered

8.1 Why Not Linear Regression?

Aspect	Linear Regression	Problem for MMM
Relationship	$y = \beta_0 + \beta_1 x_1 + ...$	Assumes linearity
Diminishing returns	Cannot capture	Marketing has saturation
Interactions	Manual specification needed	Too many potential combinations
Verdict	❌ Too simplistic	Would miss non-linear effects

8.2 Why Not Random Forest?

Aspect	Random Forest	Comparison to XGBoost
Accuracy	Good	XGBoost typically better
Overfitting	More prone	XGBoost has better regularization
Training speed	Slower	XGBoost faster (sequential optimization)
Verdict	⚠️ Viable alternative	XGBoost preferred for MMM

8.3 Why Not Neural Networks?

Aspect	Neural Networks	Problem for MMM
Data requirement	Needs large datasets	Only 208 observations
Interpretability	Black box	SHAP harder to apply
Overfitting	High risk with small data	Would require heavy regularization
Verdict	❌ Overkill	Insufficient data

8.4 Why Not Prophet Alone?

Aspect	Prophet	Limitation
Purpose	Time series forecasting	Not designed for attribution
Marketing effects	Only via regressors	Limited flexibility
Non-linearity	Linear regressors only	Can't capture diminishing returns
Verdict	⚠️ Used for preprocessing	Not suitable as final model

---

9. Preprocessing-Model Synergy

9.1 How Each Preprocessing Step Unlocks XGBoost's Power

Preprocessing	Creates	XGBoost Learns
Prophet decomposition	Trend, season, holiday features	Baseline revenue patterns
Adstock transformation	Accumulated advertising pressure	Channel-specific impact curves
One-hot encoding	Binary event indicators	Event effect magnitudes

9.2 The Complete Feature Set for XGBoost

```
features = [
    # Prophet-derived (time patterns)
    'trend',
    'season',
    'holiday',
    'events',

    # Adstock-transformed (marketing channels)
    'tv_S',          # With carryover applied
    'ooh_S',         # With carryover applied
    'print_S',       # With carryover applied
    'facebook_S',    # With carryover applied
    'search_S',      # With carryover applied
    'newsletter',    # With carryover applied

    # Contextual
    'competitor_sales_B'
]
```

9.3 Why This Combination Works

```
┌──────────────────────────────────────────────────────────────┐
│                   PREPROCESSING BENEFITS                     │
├──────────────────────────────────────────────────────────────┤
│ 1. Prophet isolates time effects                             │
│    → XGBoost doesn't confuse seasonality with marketing      │
│                                                              │
│ 2. Adstock captures carryover                                │
│    → XGBoost sees TRUE advertising impact, not just spend    │
│                                                              │
│ 3. One-hot encoding handles categories                       │
│    → XGBoost can split on event presence/absence             │
├──────────────────────────────────────────────────────────────┤
│                   XGBOOST BENEFITS                           │
├──────────────────────────────────────────────────────────────┤
│ 4. Non-linear learning                                       │
│    → Captures diminishing returns without manual curves      │
│                                                              │
```

```
│ 5. Automatic interactions                               │
│    → Detects channel synergies (TV + Facebook > sum)    │
│                                                         │
│ 6. Regularization                                       │
│    → Prevents overfitting with 208 observations         │
│                                                         │
│ 7. SHAP compatibility                                   │
│    → Enables spend vs. effect analysis for budget decisions │
└─────────────────────────────────────────────────────────┘
```

9.4 Multi-Objective Optimization: The Final Touch

Beyond accuracy, the model optimizes for **budget efficiency**:

Objective	Metric	Purpose
Accuracy	MAPE	Minimize prediction error
Efficiency	RSSD	Match effect share to spend share

Using NSGA-II algorithm, the model finds Pareto-optimal solutions balancing both objectives.

---

10. Conclusion

10.1 Summary of Preprocessing-Model Logic

Step	Preprocessing Technique	Problem Solved	Model Benefit
1	**Prophet Decomposition**	Separate time patterns from marketing	XGBoost focuses on incremental marketing contribution
2	**Adstock Transformation**	Model advertising carryover	XGBoost sees accumulated impact, not raw spend
3	**One-Hot Encoding**	Convert categorical events	XGBoost can split on event presence
4	**Optuna Tuning**	Optimize all hyperparameters	XGBoost configuration tailored to data

10.2 Why XGBoost is the Right Choice

1. **Non-linearity:** Marketing has diminishing returns—XGBoost captures this naturally
2. **Interactions:** Channel synergies detected automatically via tree splits
3. **Regularization:** Prevents overfitting with limited data (208 weeks)
4. **Interpretability:** SHAP provides actionable insights for budget allocation
5. **Multi-objective:** Can balance accuracy vs. efficiency via Pareto optimization

10.3 The End-to-End Value Chain

```
Raw Marketing Data
        │
        ▼
[Prophet] → Isolate time patterns → trend, season, holiday
        │
        ▼
[Adstock] → Model carryover → accumulated channel pressure
        │
        ▼
[One-Hot] → Encode events → binary indicators
        │
        ▼
[Optuna + XGBoost] → Learn non-linear response curves
        │
        ▼
[SHAP] → Explain predictions → Spend vs. Effect share
        │
        ▼
Budget Reallocation Recommendations
```

📊 **This document explains the complete rationale from preprocessing to model selection**

Understanding why each technique is used ensures reproducible and trustworthy Marketing Mix Models