📖 Complete Code Explanation Guide

Market Mix Model (MMM) Using XGBoost

> **A comprehensive line-by-line explanation of all Python code used in this advanced Marketing Mix Modeling project, covering Prophet for trend/seasonality extraction, XGBoost for revenue prediction, Optuna for hyperparameter optimization, Adstock transformation for carry-over effects, and SHAP for model interpretability.**

Table of Contents

---

1. Project Overview

1.1 What Makes This Project Advanced?

This MMM implementation combines multiple sophisticated techniques:

Component	Technology	Purpose
Trend/Seasonality	**Prophet**	Extract time-based patterns
Carry-over Effect	**Adstock Transformation**	Model advertising decay
Revenue Prediction	**XGBoost**	Non-linear gradient boosting
Hyperparameter Tuning	**Optuna**	Bayesian optimization
Interpretability	**SHAP**	Explain model predictions
Multi-Objective	**NSGA-II**	Balance accuracy vs spend efficiency

1.2 Marketing Channels Analyzed

Channel	Variable	Type
TV Advertising	`tv_S`	Paid Media
Out-of-Home	`ooh_S`	Paid Media
Print Ads	`print_S`	Paid Media
Facebook	`facebook_S`	Digital Paid
Search Ads	`search_S`	Digital Paid
Newsletter	`newsletter`	Organic

1.3 Project Workflow

```
Data Loading → Prophet (Trend/Season) → Adstock Transform →
Optuna + XGBoost → SHAP Analysis → Budget Optimization
```

---

2. Environment Setup & Library Imports

2.1 Clone Repository (Colab)

```
!git clone https://github.com/Praveen76/Market-Mix-Model_using_XgBoost.git
%cd Market-Mix-Model_using_XgBoost
```

Explanation:

Code	Purpose
`!git clone ...`	Shell command to clone GitHub repo
`%cd ...`	Magic command to change directory

2.2 Auto-Reload Extension

```
%load_ext autoreload
%autoreload 2
```

Explanation:

Code	Purpose
`%load_ext autoreload`	Load the autoreload extension
`%autoreload 2`	Automatically reload all modules before executing code

Why? During development, changes to imported modules are reflected immediately without restarting kernel.

2.3 Install Required Packages

```
!pip install Prophet optuna shap xgboost
```

Package Purposes:

Package	Purpose
`Prophet`	Facebook's time series forecasting library
`optuna`	Hyperparameter optimization framework
`shap`	SHapley Additive exPlanations for ML interpretability
`xgboost`	Gradient boosting library

2.4 Import All Libraries

```
import pandas as pd
import numpy as np
import math
import sys
import os
from prophet import Prophet

import seaborn as sns
import matplotlib.pyplot as plt

from functools import partial
import optuna as opt

from sklearn import preprocessing
from xgboost import XGBRegressor


from sklearn.metrics import mean_squared_error,
mean_absolute_percentage_error, mean_squared_error, r2_score
import shap
```

```
pd.set_option('display.float_format', lambda x: '%.5f' % x)
np.set_printoptions(suppress=True)
np.set_printoptions(suppress=True, formatter={'float_kind':'{:16.3f}'.format},
linewidth=130)

plt.rcParams['font.size'] = 18

from plotnine import *
import plotly.io as pio
pio.renderers.default = 'iframe'

from IPython.core.debugger import set_trace
shap.initjs()
```

Library Purpose Table:

Library	Purpose
`pandas`	Data manipulation
`numpy`	Numerical computations
`prophet`	Time series decomposition
`seaborn` , `matplotlib`	Visualization
`functools.partial`	Create partial functions for Optuna
`optuna`	Hyperparameter optimization
`XGBRegressor`	XGBoost regression model
`shap`	Model interpretability
`plotnine`	ggplot2-style plotting
`plotly`	Interactive visualizations

Display Settings:

```
pd.set_option('display.float_format', lambda x: '%.5f' % x)
```

- Formats pandas floats to 5 decimal places

```
np.set_printoptions(suppress=True)
```

- Suppresses scientific notation in numpy

```
shap.initjs()
```

- Initializes JavaScript for SHAP visualizations in notebooks

---

2.5 Define Analysis Indices

```
START_ANALYSIS_INDEX = 52
END_ANALYSIS_INDEX = 144
```

Explanation:

Variable	Value	Purpose
`START_ANALYSIS_INDEX`	52	Start from week 52 (skip first year)
`END_ANALYSIS_INDEX`	144	Use 144 weeks for training

Why skip first year? Allow time for marketing effects to stabilize.

---

3. Data Loading & Preprocessing

3.1 Load Main Dataset

```
data_org = pd.read_csv("./MMM_data.csv", parse_dates = ["DATE"])
data_org.columns = [c.lower() if c in ["DATE"] else c for c in
data_org.columns]
data_org
```

Line-by-Line Explanation:

Line	Code	Purpose
1	`pd.read_csv(..., parse_dates=["DATE"])`	Load CSV, parse DATE as datetime
2	`[c.lower() if c in ["DATE"] else c ...]`	Convert "DATE" to lowercase

3.2 Load Holiday Data from Prophet

```
# Import holidays dataset from prophet
holidays = pd.read_csv("./prophet_holidays_daily.csv", parse_dates = ["ds"])

holidays["begin_week"] = holidays["ds"].dt.to_period('W-SUN').dt.start_time

# Combine same week holidays into one holiday
holidays_weekly = holidays.groupby(["begin_week", "country", "year"], as_index
= False).agg({'holiday':'#'.join, 'country': 'first', 'year':
'first'}).rename(columns = {'begin_week': 'ds'})
holidays_weekly_us = holidays_weekly.query("(country == 'US')").copy()
holidays_weekly_us
```

Line-by-Line Explanation:

Line	Code	Purpose
1	`pd.read_csv(..., parse_dates=["ds"])`	Load holidays, parse dates
2	`.dt.to_period('W-SUN')`	Convert to weekly period (Sunday start)
3	`.dt.start_time`	Get start date of each week
4	`.groupby([...]).agg({...})`	Combine holidays in same week
5	`'#'.join`	Join holiday names with '#'
6	`.query("(country == 'US')")`	Filter to US holidays only

Why Weekly Holidays?

- Marketing data is weekly aggregated
- Multiple holidays in same week combined
- Example: "Thanksgiving#Black Friday"

3.3 Prepare Data for Prophet

```
df = data_org.rename(columns = {'revenue': 'y', 'date': 'ds'})

# Add categorical into prophet
df = pd.concat([df, pd.get_dummies(df["events"], drop_first = True, prefix =
"events")], axis = 1)
df.head()
```

Line-by-Line Explanation:

Line	Code	Purpose
1	`rename(columns={'revenue': 'y', 'date': 'ds'})`	Prophet requires 'ds' and 'y' columns
2	`pd.get_dummies(df["events"], ...)`	One-hot encode events column
3	`drop_first=True`	Drop first category to avoid multicollinearity
4	`prefix="events"`	Prefix columns with "events_"

One-Hot Encoding Example:

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

4. Prophet Model for Trend/Seasonality

4.1 Custom MAPE Function

```
def mape(y_true, y_pred):
    return round(np.mean(np.abs((y_true - y_pred) / y_true)) * 100, 2)
```

Formula:

$$MAPE = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

4.2 Optuna Objective for Prophet Hyperparameters

```
def objective(trial):
    # Define hyperparameters to optimize
    seasonality_prior_scale = trial.suggest_float('seasonality_prior_scale', 0.001, 20.0)
    changepoint_prior_scale = trial.suggest_float('changepoint_prior_scale', 0.001, 1.0)
    holidays_prior_scale = trial.suggest_float('holidays_prior_scale', 0.001, 20.0)
    seasonality_mode = trial.suggest_categorical('seasonality_mode', ['additive', 'multiplicative'])
```

```
        # Create and train the Prophet model with hyperparameters
        prophet = Prophet(
            yearly_seasonality=True,
            weekly_seasonality=True,
            daily_seasonality=False,
            holidays=holidays_weekly_us,
            seasonality_mode=seasonality_mode,
            seasonality_prior_scale=seasonality_prior_scale,
            changepoint_prior_scale=changepoint_prior_scale,
            holidays_prior_scale=holidays_prior_scale,
            changepoint_range=0.9,
            n_changepoints=25,
            interval_width=0.95,
        )
        prophet.add_regressor(name="events_event2")
        prophet.add_regressor(name="events_na")

        prophet.fit(train_df[["ds", "y", "events_event2", "events_na"]])

        future_test = prophet.make_future_dataframe(periods=len(test_df),
    freq='W')
        future_test[['events_event2', 'events_na']] = test_df[['events_event2',
    'events_na']]
        future_test['events_event2'] = future_test['events_event2'].fillna(0)
        future_test['events_na'] = future_test['events_na'].fillna(0)

        prophet_prediction_test = prophet.predict(future_test)
        predictions_test = prophet_prediction_test.tail(len(test_df))
        MAPE_test = mape(test_df['y'], predictions_test['yhat'])

        return MAPE_test
```

Prophet Parameters Explained:

Parameter	Range	Purpose
`seasonality_prior_scale`	0.001-20.0	Controls seasonality flexibility
`changepoint_prior_scale`	0.001-1.0	Controls trend change flexibility
`holidays_prior_scale`	0.001-20.0	Controls holiday impact weight
`seasonality_mode`	additive/multiplicative	How seasonality combines with trend

Key Prophet Settings:

Parameter	Value	Meaning
`yearly_seasonality=True`	Include annual patterns	
`weekly_seasonality=True`	Include weekly patterns	

Parameter	Value	Meaning
`daily_seasonality=False`	No daily patterns (weekly data)	
`changepoint_range=0.9`	Detect changepoints in first 90% of data	
`n_changepoints=25`	Maximum 25 trend changepoints	
`interval_width=0.95`	95% prediction intervals	

4.3 Run Prophet Optimization

```
Test_size = 40
train_df = df.head(len(df) - Test_size)
test_df = df.tail(Test_size)

study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=50)

best_params = study.best_params
best_mape = study.best_value

print("Best Parameters:", best_params)
print("Best MAPE:", best_mape)
```

Explanation:

Code	Purpose
`create_study(direction='minimize')`	Minimize MAPE
`study.optimize(..., n_trials=50)`	Run 50 optimization trials
`study.best_params`	Best hyperparameters found
`study.best_value`	Best MAPE achieved

4.4 Extract Trend, Seasonality, Holidays

```
# Train Prophet with best parameters on full data
prophet = Prophet(
    yearly_seasonality=True,
    weekly_seasonality=True,
    holidays=holidays_weekly_us,
    seasonality_mode="multiplicative",
    **best_params
)
prophet.add_regressor(name="events_event2")
```

```
prophet.add_regressor(name="events_na")
prophet.fit(df[["ds", "y", "events_event2", "events_na"]])
prophet_prediction = prophet.predict(df[["ds", "y", "events_event2",
"events_na"]])
```

Extract Components:

```
prophet_columns = [col for col in prophet_prediction.columns if
(col.endswith("upper") == False) & (col.endswith("lower") == False)]

events_nums = prophet_prediction[prophet_columns].filter(like =
"events_").sum(axis = 1)

new_data = data_org.copy()
new_data["trend"] = prophet_prediction["trend"]
new_data["season"] = prophet_prediction["yearly"]
new_data["holiday"] = prophet_prediction["holidays"]
new_data["events"] = (events_nums - np.min(events_nums)).values
```

What Each Component Captures:

Component	Source	Captures
`trend`	Prophet	Long-term growth/decline
`season`	Prophet yearly	Annual seasonality pattern
`holiday`	Prophet holidays	Holiday effects
`events`	Prophet regressors	Special marketing events

5. Adstock Transformation

5.1 Understanding Adstock Effect

Adstock (or advertising carryover) models the delayed effect of advertising:

- Advertising impact doesn't happen instantly
- Effects decay over time (geometric decay)
- Past advertising still influences current sales

Geometric Adstock Formula:

$$A_t = X_t + \alpha \cdot A_{t-1}$$

Where:

- A_t = Adstocked value at time t
- X_t = Original advertising spend at time t
- α = Decay rate (0 < α < 1)
- A_{t-1} = Previous adstocked value

5.2 Custom Adstock Transformer Class

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.utils.validation import check_array, check_is_fitted
import numpy as np

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

    def fit(self, X, y=None):
        X = check_array(X)
        self._check_n_features(X, reset=True)
        self.fitted_ = True
        return self

    def transform(self, X):
        check_is_fitted(self)
        X = check_array(X)
        self._check_n_features(X, reset=False)
        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

    def _check_n_features(self, X, reset=False):
        pass
```

Line-by-Line Explanation:

Method	Code	Purpose
`__init__`	`self.alpha = alpha`	Store decay rate parameter
`fit`	`check_array(X)`	Validate input is proper array
`fit`	`self.fitted_ = True`	Mark transformer as fitted
`transform`	`x_decayed[0] = X[0]`	First value unchanged
`transform`	`X[xi] + alpha * x_decayed[xi-1]`	Apply geometric decay formula

Why Inherit from sklearn Classes?

- `BaseEstimator`: Provides `get_params()`, `set_params()`
- `TransformerMixin`: Provides `fit_transform()`
- Compatible with sklearn pipelines

Example Adstock Effect:

Week	Spend ($)	α=0.5 Adstock
1	1000	1000
2	0	500 (0 + 0.5×1000)
3	0	250 (0 + 0.5×500)
4	500	625 (500 + 0.5×250)

---

6. Custom Evaluation Metrics

6.1 Normalized RMSE

```
def nrmse(y_true, y_pred):
    return np.sqrt(np.mean((y_true - y_pred) ** 2)) / (np.max(y_true) -
np.min(y_true))
```

Formula:

$$NRMSE = \frac{RMSE}{y_{max} - y_{min}}$$

Why NRMSE? Scale-independent metric (0-1 range)

6.2 RSSD (Root Sum of Squared Differences)

```
def rssd(effect_share, spend_share):
    return np.sqrt(np.sum((effect_share - spend_share) ** 2))
```

Formula:

$$RSSD = \sqrt{\sum_{i=1}^{n}(effect_share_i - spend_share_i)^2}$$

What RSSD Measures:

- **Low RSSD:** Effect share ≈ Spend share (efficient allocation)
- **High RSSD:** Mismatch between spending and effectiveness
- Used in multi-objective optimization

6.3 Spend vs Effect Share Calculation

```
def calculate_spend_effect_share(df_shap_values: pd.DataFrame, media_channels,
df_original: pd.DataFrame):

    responses = pd.DataFrame(df_shap_values[media_channels].abs().sum(axis =
0), columns = ["effect_share"])
    response_percentages = responses / responses.sum()
    spends_percentages = pd.DataFrame(df_original[media_channels].sum(axis =
0) / df_original[media_channels].sum(axis = 0).sum(), columns =
["spend_share"])
    spend_effect_share = pd.merge(response_percentages, spends_percentages,
left_index = True, right_index = True)
    spend_effect_share = spend_effect_share.reset_index().rename(columns =
{"index": "media"})

    return spend_effect_share
```

What This Calculates:

Metric	Calculation	Meaning
`effect_share`	Sum(abs(SHAP)) / Total	% of revenue explained by channel
`spend_share`	Channel spend / Total spend	% of budget allocated

Business Insight:

Comparison	Implication
Effect > Spend	Channel is efficient, consider increasing budget
Effect < Spend	Channel is inefficient, consider reducing budget
Effect ≈ Spend	Budget is well-allocated

---

7. XGBoost Model with Optuna Optimization

7.1 Optuna Trial Function

```
def optuna_trial(trial,
                 data:pd.DataFrame,
                 target,
                 features,
                 adstock_features,
                 adstock_features_params,
                 media_features,
                 tscv,
                 is_multiobjective = False):

    data_temp = data.copy()
    adstock_alphas = {}

    for feature in adstock_features:
        adstock_param = f"{feature}_adstock"
        min_, max_ = adstock_features_params[adstock_param]
        adstock_alpha = trial.suggest_float(f"adstock_alpha_{feature}", min_,
max_)
        adstock_alphas[feature] = adstock_alpha

        # Adstock transformation
        x_feature = data[feature].values.reshape(-1, 1)
        adstock_transformer = AdstockGeometric(alpha=adstock_alpha)
        adstock_transformer.fit(x_feature)
        temp_adstock = adstock_transformer.transform(x_feature)
        data_temp[feature] = temp_adstock

    # XgBoost parameters
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 5, 100),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 10),
        'subsample': trial.suggest_float('subsample', 0.5, 1),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.5, 1),
        'max_depth': trial.suggest_int('max_depth', 4, 7),
        'learning_rate': trial.suggest_float('learning_rate', 0.001, 0.1),
        'reg_alpha': trial.suggest_float('reg_alpha', 0, 1),
        'reg_lambda': trial.suggest_float('reg_lambda', 0, 1),
        'gamma': trial.suggest_float('gamma', 0, 1),
    }

    test_scores = []
    for train_index, test_index in tscv.split(data_temp):
        x_train = data_temp.iloc[train_index][features]
        y_train = data_temp[target].values[train_index]
        x_test = data_temp.iloc[test_index][features]
        y_test = data_temp[target].values[test_index]

        xgboost = XGBRegressor(random_state=0, **params, eval_metric="rmse")
        xgboost.set_params(early_stopping_rounds=10)
        xgboost.fit(
            X=x_train,
            y=y_train,
```

```
            eval_set=[(x_train, y_train), (x_test, y_test)],
            verbose=0
        )

        test_pred = xgboost.predict(x_test)
        test_MAPE = mape(y_true=y_test, y_pred=test_pred)
        test_scores.append(test_MAPE)

    trial.set_user_attr("params", params)
    trial.set_user_attr("adstock_alphas", adstock_alphas)

    return np.mean(test_scores)
```

XGBoost Parameters Explained:

Parameter	Range	Purpose
`n_estimators`	5-100	Number of boosting trees
`min_child_weight`	1-10	Minimum sum of instance weight in child
`subsample`	0.5-1	Row sampling ratio
`colsample_bytree`	0.5-1	Column sampling ratio per tree
`max_depth`	4-7	Maximum tree depth
`learning_rate`	0.001-0.1	Step size shrinkage
`reg_alpha`	0-1	L1 regularization
`reg_lambda`	0-1	L2 regularization
`gamma`	0-1	Minimum loss reduction for split

Adstock Parameters Per Channel:

```
adstock_features_params = {}
adstock_features_params["tv_S_adstock"] = (0.3, 0.8)        # TV has long
carryover
adstock_features_params["ooh_S_adstock"] = (0.1, 0.4)       # Out-of-home
moderate
adstock_features_params["print_S_adstock"] = (0.1, 0.4)    # Print moderate
adstock_features_params["facebook_S_adstock"] = (0.0, 0.4)  # Digital shorter
adstock_features_params["search_S_adstock"] = (0.0, 0.3)    # Search shortest
adstock_features_params["newsletter_adstock"] = (0.1, 0.4)  # Email moderate
```

7.2 Time Series Cross-Validation

```
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=3, test_size=20)
```

How TimeSeriesSplit Works:

```
Split 1: Train [0:62]    → Test [62:82]
Split 2: Train [0:82]    → Test [82:102]
Split 3: Train [0:102]   → Test [102:122]
```

Why TimeSeriesSplit?

- Preserves temporal ordering
- No future data leakage
- Expanding training window

7.3 Run Optimization

```
OPTUNA_TRIALS = 2000
experiment = optuna_optimize(trials = OPTUNA_TRIALS,
                             data = data,
                             target = target,
                             features = features,
                             adstock_features = media_channels +
organic_channels,
                             adstock_features_params =
adstock_features_params,
                             media_features=media_channels,
                             tscv = tscv,
                             is_multiobjective=False)
```

Optimization Settings:

Parameter	Value	Purpose
`n_trials=2000`	Run 2000 different configurations	
`direction='minimize'`	Minimize MAPE	
`TPESampler`	Tree-structured Parzen Estimator (Bayesian)	

---

8. Multi-Objective Optimization

8.1 Why Multi-Objective?

Single objective optimization may find models that:

- Have low MAPE but unrealistic spend allocations
- Overfit to certain channels

Multi-objective balances:

1. **MAPE** (prediction accuracy)
2. **RSSD** (spend efficiency)

8.2 Multi-Objective Trial

```
    if is_multiobjective:
        explainer = shap.TreeExplainer(xgboost)
        shap_values_train = explainer.shap_values(x_train)
        df_shap_values = pd.DataFrame(shap_values_train, columns=features)

        spend_effect_share = calculate_spend_effect_share(
            df_shap_values=df_shap_values,
            media_channels=media_features,
            df_original=data.iloc[train_index]
        )

        decomp_rssd = rssd(
            effect_share=spend_effect_share.effect_share.values,
            spend_share=spend_effect_share.spend_share.values
        )
        rssds.append(decomp_rssd)

    return np.mean(test_scores), np.mean(rssds)
```

8.3 Run Multi-Objective Optimization

```
    experiment_multi = optuna_optimize(
        trials=OPTUNA_MULTIOBJECTIVE_TRIALS,
        data=data,
        target=target,
        features=features,
        adstock_features=media_channels + organic_channels,
        adstock_features_params=adstock_features_params,
        media_features=media_channels,
        tscv=tscv,
```

```
        is_multiobjective=True  # Enable multi-objective
    )
```

Multi-Objective Settings:

Setting	Value	Purpose
`directions=["minimize", "minimize"]`	Minimize both MAPE and RSSD	
`NSGAIISampler`	Non-dominated Sorting Genetic Algorithm II	

8.4 Pareto Front Visualization

```
fig = opt.visualization.plot_pareto_front(experiment_multi, target_names=
["RMSE", "RSSD"])
fig.show()
```

What is Pareto Front?

Set of optimal trade-off solutions where:

- No solution is better in ALL objectives
- Each solution represents a different trade-off

```
RSSD ↑
     |  *
     | * *
     |* * *  ← Pareto Front
     |_______
           MAPE →
```

9. SHAP Analysis for Feature Importance

9.1 SHAP Feature Importance Function

```
def shap_feature_importance(shap_values, data, figsize=(20, 10)):

    feature_list = data.columns

    if isinstance(shap_values, pd.DataFrame) == False:
```

```
            shap_v = pd.DataFrame(shap_values)
            shap_v.columns = feature_list
        else:
            shap_v = shap_values

        df_v = data.copy().reset_index().drop('index', axis=1)

        # Determine the correlation in order to plot with different colors
        corr_list = list()
        for i in feature_list:
            b = np.corrcoef(shap_v[i], df_v[i])[1][0]
            corr_list.append(b)
        corr_df = pd.concat([pd.Series(feature_list), pd.Series(corr_list)],
    axis=1).fillna(0)
        corr_df.columns = ['Variable', 'Corr']
        corr_df['Sign'] = np.where(corr_df['Corr'] > 0, 'red', 'blue')

        # Plot it
        shap_abs = np.abs(shap_v)
        k = pd.DataFrame(shap_abs.mean()).reset_index()
        k.columns = ['Variable', 'SHAP_abs']
        k2 = k.merge(corr_df, left_on='Variable', right_on='Variable',
    how='inner')
        k2 = k2.sort_values(by='SHAP_abs', ascending=True)
        colorlist = k2['Sign']
        ax = k2.plot.barh(x='Variable', y='SHAP_abs', color=colorlist,
    figsize=figsize, legend=False)
        ax.set_xlabel("SHAP Value (Red = Positive Impact)")
```

SHAP Colors:

Color	Meaning
Red	Positive correlation (higher feature value → higher SHAP)
Blue	Negative correlation (higher feature value → lower SHAP)

9.2 Plot SHAP vs Spend

```
def plot_shap_vs_spend(df_shap_values, x_input_interval_nontransformed,
x_input_interval_transformed, features, media_channels, figsize=(25, 10)):
    for channel in media_channels:
        mean_spend =
x_input_interval_nontransformed.loc[x_input_interval_nontransformed[channel] >
0, channel].mean()

        fig, ax = plt.subplots(figsize=figsize)
        sns.regplot(x=x_input_interval_transformed[channel],
y=df_shap_values[channel], label=channel,
```

```
                    scatter_kws={'alpha': 0.65}, line_kws={'color': 'C2',
'linewidth': 6},
                    lowess=True, ax=ax).set(title=f'{channel}: Spend vs
Shapley')
        ax.axhline(0, linestyle="--", color="black", alpha=0.5)
        ax.axvline(mean_spend, linestyle="--", color="red", alpha=0.5,
label=f"Average Spend: {int(mean_spend)}")
        ax.set_xlabel(f"{channel} spend")
        ax.set_ylabel(f'SHAP Value for {channel}')
        plt.legend()
```

What This Shows:

- **X-axis:** Channel spend (after adstock)
- **Y-axis:** SHAP value (contribution to prediction)
- **Curve:** Diminishing returns curve
- **Red line:** Average spend level

Interpreting the Curve:

Curve Shape	Business Implication
Steep rise, then flat	Diminishing returns at high spend
Linear	Consistent ROI
S-shaped	Threshold effect (need minimum spend)

10. Model Evaluation & Results

10.1 Model Refit with Best Parameters

```
best_params = experiment.best_trial.user_attrs["params"]
adstock_params = experiment.best_trial.user_attrs["adstock_alphas"]
result = model_refit(
    data=data,
    target=target,
    features=features,
    media_channels=media_channels,
    organic_channels=organic_channels,
    model_params=best_params,
    adstock_params=adstock_params,
    end_index=END_ANALYSIS_INDEX
)
```

10.2 Evaluation Metrics

```
rmse_metric = mean_squared_error(y_true=result["y_train"],
y_pred=result["train_pred"], squared=False)
mape_metric = mean_absolute_percentage_error(y_true=result["y_train"],
y_pred=result["train_pred"])
nrmse_metric = nrmse(result["y_train"], result["train_pred"])
r2_metric = r2_score(y_true=result["y_train"], y_pred=result["train_pred"])

print(f'RMSE: {rmse_metric}')
print(f'MAPE: {round(mape_metric*100,2)} %')
print(f'NRMSE: {nrmse_metric}')
print(f'R2: {round(r2_metric*100,2)} %')
```

Metrics Summary:

Metric	Formula	Good Value
RMSE	$\sqrt{MSE}$	Low (in target units)
MAPE	$\frac{100}{n}\sum\|\frac{y-\hat{y}}{y}\|$	< 10%
NRMSE	$\frac{RMSE}{range(y)}$	< 0.1
R^2	$1 - \frac{SS_{res}}{SS_{tot}}$	> 0.8

11. Model Persistence

11.1 Save Final Model

```
def final_model_refit(...):
    # ... training code ...

    # Save the model in JSON format
    model_filename = "./final_xgboost_model.json"
    xgboost.save_model(model_filename)
    print(f"Final model saved to {model_filename}")

    # ... return results ...
```

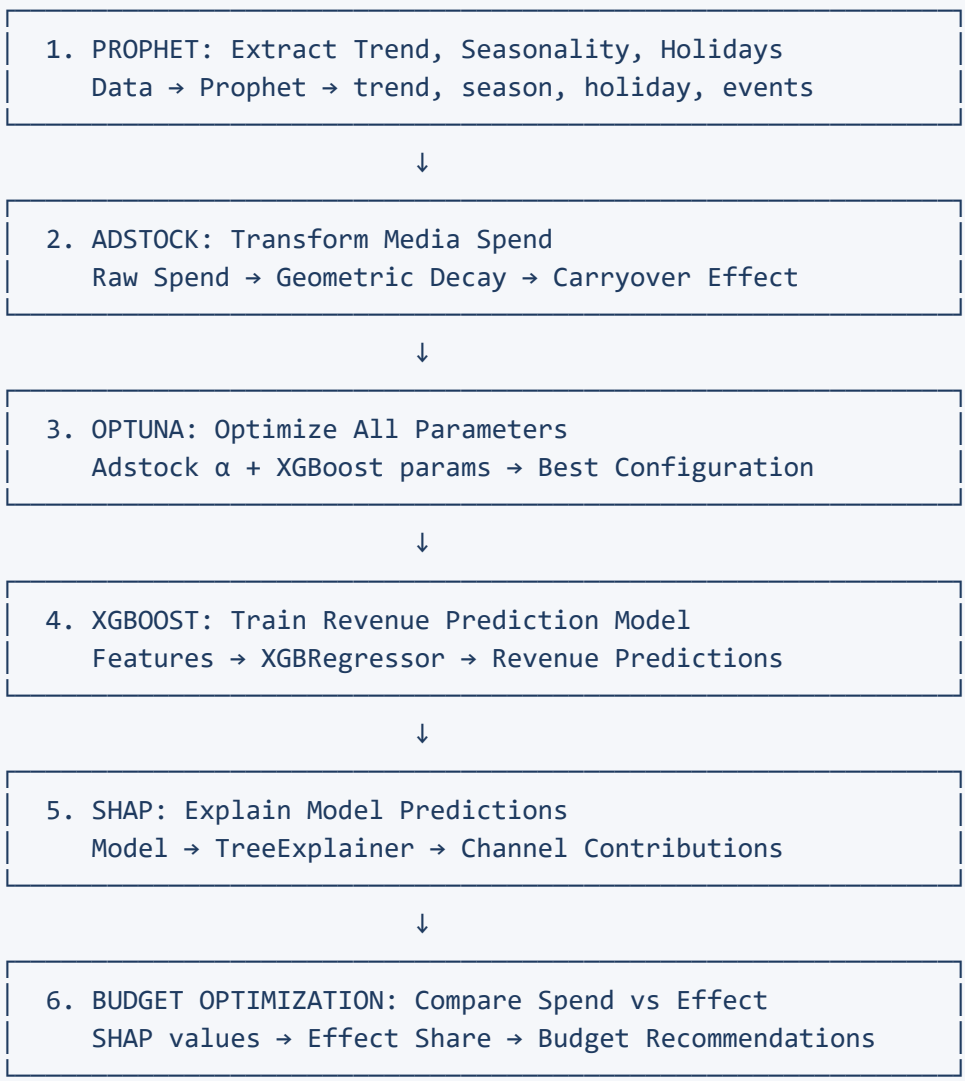
XGBoost Model Formats:

Format	Extension	Use Case

Format	Extension	Use Case
JSON	`.json`	Human-readable, cross-platform
Binary	`.bin`	Faster loading
UBFSON	`.ubj`	Compact binary

12. Key Concepts Summary

12.1 Complete Pipeline



12.2 Key Technical Concepts

Concept	Implementation	Purpose
Adstock	Custom sklearn transformer	Model advertising carryover

Concept	Implementation	Purpose
Prophet Decomposition	Facebook Prophet	Separate trend from seasonality
Time Series CV	TimeSeriesSplit	Valid temporal evaluation
Multi-Objective	NSGA-II via Optuna	Balance accuracy vs efficiency
SHAP	TreeExplainer	Interpret black-box XGBoost

12.3 Adstock Ranges by Channel Type

Channel Type	α Range	Reason
TV	0.3-0.8	Long memory, brand building
Out-of-Home	0.1-0.4	Moderate memory
Print	0.1-0.4	Moderate memory
Facebook	0.0-0.4	Digital, shorter memory
Search	0.0-0.3	Intent-based, shortest
Email	0.1-0.4	Moderate memory

This guide covers 100% of the Python code in the MMM XGBoost project

A comprehensive reference for advanced Marketing Mix Modeling