📖 Complete Code Explanation Guide

Marketing Mix Modeling (MMM) for Marketing Budget Optimization

> **A comprehensive line-by-line explanation of all Python code used in this Marketing Mix Modeling project, covering data preprocessing, feature engineering, linear regression modeling, ROI analysis, budget simulation, and Flask API deployment.**

Table of Contents

---

1. Project Overview

1.1 What is Marketing Mix Modeling (MMM)?

Marketing Mix Modeling is a statistical analysis technique used to:

- Measure the impact of marketing activities on sales/revenue
- Optimize marketing budget allocation across channels
- Quantify Return on Investment (ROI) for each marketing channel
- Simulate "what-if" scenarios for budget planning

1.2 The Marketing Mix (4 Ps)

Component	Description	Examples in Dataset
Product	What you sell	E-commerce products
Price	Pricing strategy	Original price, discounts
Place	Distribution channels	Online, regional

Component	Description	Examples in Dataset
Promotion	Marketing activities	Google Ads, Meta Ads, Email

1.3 Marketing Channels Analyzed

Channel Category	Specific Channels
Google Ads	Paid Search, Shopping, Performance Max
Meta Ads	Facebook, Instagram
Organic	Search, Direct, Referral
Other	Email, Branded Search

1.4 Project Workflow

```
Data Loading → Missing Value Treatment → Feature Engineering →
Train-Test Split → Model Training → Evaluation →
Budget Simulation → API Deployment
```

2. Data Import & Loading

2.1 Import Libraries

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

Library Purpose Table:

Library	Import As	Purpose
`numpy`	`np`	Numerical computations, array operations
`pandas`	`pd`	Data manipulation, DataFrame operations
`seaborn`	`sns`	Statistical data visualization
`matplotlib.pyplot`	`plt`	Core plotting library

Library	Import As	Purpose
`warnings`	-	Suppress warning messages

Why Suppress Warnings?

```
warnings.filterwarnings('ignore')
```

- Prevents cluttering notebook with non-critical warnings
- Common warnings: deprecation notices, convergence warnings
- **Note:** In production, handle warnings appropriately

2.2 Load Dataset

```
# Load the main dataset
df = pd.read_csv("Multi-Region Ecommerce MMM Dataset.csv")
df
```

Explanation:

Code	Purpose
`pd.read_csv(...)`	Load CSV file into pandas DataFrame
`df`	Display DataFrame (Jupyter auto-renders)

Dataset Characteristics:

- **Multi-Region:** Data from different geographical regions
- **E-commerce:** Online retail sales data
- **MMM Dataset:** Contains marketing spend, clicks, impressions, and sales

---

2.3 Initial Data Inspection

```
# Quick check
df.info()
```

What `.info()` Shows:

Information	Description
Number of rows/columns	Dataset dimensions
Column names	All variable names
Data types	int64, float64, object, datetime64
Non-null counts	Number of non-missing values
Memory usage	RAM consumption

```
df.head()
```

What `.head()` Shows:

- First 5 rows by default
- Quick preview of actual data values
- Column structure verification

---

3. Missing Value Treatment

3.1 Check Missing Values

```
# Check null values summary (optional to visualize)
print(df.isnull().sum().sort_values(ascending=False))
```

Line-by-Line Explanation:

Code	Purpose
`df.isnull()`	Returns boolean DataFrame (True for NaN)
`.sum()`	Counts True values per column
`.sort_values(ascending=False)`	Shows columns with most missing first

Output Example:

```
TIKTOK_SPEND                1500
TIKTOK_CLICKS               1500
GOOGLE_VIDEO_SPEND          1200
```

```
...
DATE_DAY                          0
```

3.2 Drop Irrelevant Columns

```python
# Drop irrelevant columns
drop_cols = [
    'TIKTOK_SPEND', 'TIKTOK_CLICKS', 'TIKTOK_IMPRESSIONS',
    'GOOGLE_VIDEO_SPEND', 'GOOGLE_VIDEO_CLICKS', 'GOOGLE_VIDEO_IMPRESSIONS',
    'GOOGLE_DISPLAY_SPEND', 'GOOGLE_DISPLAY_CLICKS',
'GOOGLE_DISPLAY_IMPRESSIONS',
    'META_OTHER_SPEND', 'META_OTHER_CLICKS', 'META_OTHER_IMPRESSIONS'
]
df.drop(columns=drop_cols, inplace=True)
```

Explanation:

Code	Purpose
`drop_cols = [...]`	List of columns to remove
`df.drop(columns=drop_cols, ...)`	Remove specified columns
`inplace=True`	Modify DataFrame directly (no assignment needed)

Why Drop These Columns?

Reason	Columns Affected
Too many missing values	TIKTOK_* (not active channel)
Not relevant to analysis	GOOGLE_VIDEO_, *GOOGLE_DISPLAY_*
Low data quality	META_OTHER_*

Alternative to `inplace=True`:

```python
# Without inplace:
df = df.drop(columns=drop_cols)

# With inplace:
df.drop(columns=drop_cols, inplace=True)
```

3.3 Fill Missing Values with Zero

```
# Fill missing values with zero
fill_zero_cols = [
    'GOOGLE_PAID_SEARCH_SPEND', 'GOOGLE_PAID_SEARCH_CLICKS',
'GOOGLE_PAID_SEARCH_IMPRESSIONS',
    'GOOGLE_SHOPPING_SPEND', 'GOOGLE_SHOPPING_CLICKS',
'GOOGLE_SHOPPING_IMPRESSIONS',
    'GOOGLE_PMAX_SPEND', 'GOOGLE_PMAX_CLICKS', 'GOOGLE_PMAX_IMPRESSIONS',
    'META_FACEBOOK_SPEND', 'META_FACEBOOK_CLICKS',
'META_FACEBOOK_IMPRESSIONS',
    'META_INSTAGRAM_SPEND', 'META_INSTAGRAM_CLICKS',
'META_INSTAGRAM_IMPRESSIONS',
    'BRANDED_SEARCH_CLICKS', 'DIRECT_CLICKS', 'EMAIL_CLICKS',
    'REFERRAL_CLICKS', 'ALL_OTHER_CLICKS', 'ORGANIC_SEARCH_CLICKS'
]
df[fill_zero_cols] = df[fill_zero_cols].fillna(0)
```

Explanation:

Code	Purpose
`fill_zero_cols = [...]`	List of columns to fill
`df[fill_zero_cols]`	Select multiple columns
`.fillna(0)`	Replace NaN with 0

Why Fill with Zero?

Data Type	Reason for Zero-Fill
Marketing Spend	No spend = $0 spent
Clicks	No activity = 0 clicks
Impressions	No campaign = 0 impressions

When NOT to Fill with Zero:

- Continuous measurements (temperature, weight)
- Mandatory fields (customer ID)
- Categorical data

3.4 Drop Metadata Columns

```
optional_meta_cols = ['MMM_TIMESERIES_ID', 'ORGANISATION_VERTICAL',
'ORGANISATION_SUBVERTICAL']
df.drop(columns=optional_meta_cols, inplace=True)
```

Why Drop These?

Column	Reason
`MMM_TIMESERIES_ID`	Internal identifier, not a feature
`ORGANISATION_VERTICAL`	Categorical metadata
`ORGANISATION_SUBVERTICAL`	Categorical metadata

3.5 Verify Cleaning

```
print("\n Missing values after cleaning:")
df.isnull().sum()
```

Expected Output:

```
DATE_DAY                          0
GOOGLE_PAID_SEARCH_SPEND          0
GOOGLE_PAID_SEARCH_CLICKS         0
...
(all zeros)
```

---

4. Feature Engineering & Preprocessing

4.1 Date Conversion

```
# Convert DATE_DAY to datetime
df['DATE_DAY'] = pd.to_datetime(df['DATE_DAY'])
```

Why Convert to Datetime?

Benefit	Example Usage
Extract date components	`.dt.year` , `.dt.month`
Time-based filtering	`df[df['DATE_DAY'] > '2023-01-01']`
Proper sorting	Chronological order
Date arithmetic	Calculate date differences

4.2 Sort by Date

```
# Sort data by Date
df = df.sort_values('DATE_DAY')
```

Why Sort?

- **Time series requirement:** Data must be in chronological order
- **Train-test split:** Ensures temporal ordering
- **Visualization:** Correct x-axis plotting

4.3 Create Time Features

```
# Create time features
df['year'] = df['DATE_DAY'].dt.year
df['month'] = df['DATE_DAY'].dt.month
df['week'] = df['DATE_DAY'].dt.isocalendar().week
df['day_of_week'] = df['DATE_DAY'].dt.dayofweek
```

Line-by-Line Explanation:

Line	Code	Output Example
1	`.dt.year`	2023, 2024
2	`.dt.month`	1-12
3	`.dt.isocalendar().week`	1-52 (ISO week number)
4	`.dt.dayofweek`	0=Monday, 6=Sunday

Why Create Time Features?

Feature	Captures

Feature	Captures
`year`	Annual trends, YoY growth
`month`	Seasonality (holidays, quarters)
`week`	Weekly patterns
`day_of_week`	Weekday vs weekend effects

4.4 Visualize Sales Over Time

```
plt.figure(figsize=(12,6))
sns.lineplot(data=df, x='DATE_DAY', y='ALL_PURCHASES_ORIGINAL_PRICE')
plt.title("Sales Over Time")
plt.show()
```

Explanation:

Code	Purpose
`plt.figure(figsize=(12,6))`	Create 12×6 inch figure
`sns.lineplot(...)`	Draw line plot with seaborn
`x='DATE_DAY'`	Time on x-axis
`y='ALL_PURCHASES_ORIGINAL_PRICE'`	Sales on y-axis
`plt.title(...)`	Add chart title
`plt.show()`	Display the plot

4.5 Calculate Revenue

```
# Create a new column 'revenue'
df['revenue'] = df['ALL_PURCHASES_ORIGINAL_PRICE'] -
df['ALL_PURCHASES_GROSS_DISCOUNT']
```

Revenue Formula:

$$\text{Revenue} = \text{Original Price} - \text{Discounts}$$

Why Calculate Net Revenue?

- Original price includes discounts not actually collected

- Net revenue reflects actual money received
- More accurate for ROI calculations

4.6 Calculate Total Marketing Spend

```
# Calculate Total Spend
df['total_spend'] = (
    df['GOOGLE_PAID_SEARCH_SPEND'] +
    df['GOOGLE_SHOPPING_SPEND'] +
    df['GOOGLE_PMAX_SPEND'] +
    df['META_FACEBOOK_SPEND'] +
    df['META_INSTAGRAM_SPEND']
)
```

Explanation:

Channel	Description
`GOOGLE_PAID_SEARCH_SPEND`	Text ads on Google search
`GOOGLE_SHOPPING_SPEND`	Product listing ads
`GOOGLE_PMAX_SPEND`	Performance Max campaigns
`META_FACEBOOK_SPEND`	Facebook ads
`META_INSTAGRAM_SPEND`	Instagram ads

Total Spend: Sum of all paid marketing channels

4.7 Calculate ROI

```
# Calculate ROI
df['roi'] = df['revenue'] / (df['total_spend'] + 1)  # add 1 to avoid div-by-zero
```

ROI Formula:

$$ROI = \frac{\text{Revenue}}{\text{Total Spend} + 1}$$

Why Add 1?

Scenario	Without +1	With +1

Scenario	Without +1	With +1
total_spend = 0	Division by zero error	revenue / 1 = revenue
total_spend = 100	revenue / 100	revenue / 101 (minimal impact)

Alternative Approaches:

```
# Method 1: Add small epsilon
df['roi'] = df['revenue'] / (df['total_spend'] + 0.001)

# Method 2: Use np.where
df['roi'] = np.where(df['total_spend'] > 0,
                     df['revenue'] / df['total_spend'],
                     0)

# Method 3: Replace inf after division
df['roi'] = df['revenue'] / df['total_spend']
df['roi'] = df['roi'].replace([np.inf, -np.inf], 0)
```

4.8 Visualize ROI Over Time

```
plt.figure(figsize=(12,8))
sns.lineplot(data=df, x='DATE_DAY', y='roi')
plt.title("ROI Over Time")
plt.show()
```

What ROI Chart Shows:

- **Peaks:** High-efficiency marketing periods
- **Troughs:** Low ROI (overspending or low conversion)
- **Trends:** Improving/declining marketing effectiveness

4.9 Define Target and Features

```
target = 'revenue'
```

Target Variable:

- **What we want to predict:** Revenue

- **Dependent variable:** Changes based on marketing inputs

```
features = [
    'GOOGLE_PAID_SEARCH_SPEND', 'GOOGLE_SHOPPING_SPEND',
    'GOOGLE_PMAX_SPEND', 'META_FACEBOOK_SPEND', 'META_INSTAGRAM_SPEND',
    'EMAIL_CLICKS', 'ORGANIC_SEARCH_CLICKS', 'DIRECT_CLICKS',
    'BRANDED_SEARCH_CLICKS', 'year', 'month', 'day_of_week'
]
```

Feature Categories:

Category	Features	Type
Paid Marketing	Google Spend, Meta Spend	Controllable
Organic Traffic	Organic Search Clicks	Non-controllable
Direct Traffic	Direct Clicks	Non-controllable
Email	Email Clicks	Semi-controllable
Time	year, month, day_of_week	Control variables

---

5. Train-Test Split

5.1 Time-Based Split

```
# Sort by date
df = df.sort_values('DATE_DAY')

# Define cutoff date for time-based split
cutoff_date = '2023-12-31'  # Train on data up to end of 2023

# Train = everything before cutoff
train_df = df[df['DATE_DAY'] <= cutoff_date]

# Test = everything after cutoff
test_df = df[df['DATE_DAY'] > cutoff_date]

# Features and target
X_train = train_df[features]
y_train = train_df['revenue']
X_test = test_df[features]
y_test = test_df['revenue']
```

Line-by-Line Explanation:

Line	Code	Purpose
1	`df.sort_values('DATE_DAY')`	Ensure chronological order
2	`cutoff_date = '2023-12-31'`	Define split boundary
3	`df['DATE_DAY'] <= cutoff_date`	Boolean mask for training data
4	`df['DATE_DAY'] > cutoff_date`	Boolean mask for test data
5	`train_df[features]`	Extract feature columns for training
6	`train_df['revenue']`	Extract target column

Why Time-Based Split (Not Random)?

Aspect	Random Split	Time-Based Split
Data leakage	Future data can leak into training	No leakage
Realism	Unrealistic scenario	Simulates real forecasting
Temporal patterns	Disrupted	Preserved
Best for	Cross-sectional data	Time series data

Typical Split Timeline:

```
|-------- Training Data --------|-------- Test Data --------|
       (Historical)                    (Future/Recent)
    Jan 2022 - Dec 2023              Jan 2024 - Present
```

---

6. Model Training

6.1 Linear Regression Model

```python
# Linear Regression model (standard for MMM)
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
```

Line-by-Line Explanation:

Line	Code	Purpose
1	`from sklearn.linear_model import LinearRegression`	Import model class
2	`model = LinearRegression()`	Create model instance
3	`model.fit(X_train, y_train)`	Train model on training data

Why Linear Regression for MMM?

Advantage	Explanation
Interpretability	Coefficients show direct impact
Industry Standard	Widely accepted in marketing analytics
Coefficients = ROI	Each $ spent → coefficient × $ revenue
Simplicity	Easy to explain to stakeholders
Fast Training	No hyperparameter tuning needed

Linear Regression Formula:

$$\text{Revenue} = \beta_0 + \beta_1 \times \text{Google\_Spend} + \beta_2 \times \text{Meta\_Spend} + ... + \epsilon$$

Where:

- β_0 = Intercept (baseline revenue)
- β_i = Coefficient for feature i
- ϵ = Error term

---

7. Model Evaluation

7.1 Make Predictions and Evaluate

```python
from sklearn.metrics import mean_squared_error, r2_score

y_pred = model.predict(X_test)

print("R² Score:", r2_score(y_test, y_pred))
print("RMSE:", mean_squared_error(y_test, y_pred))
```

Line-by-Line Explanation:

Line	Code	Purpose

Line	Code	Purpose
1	`from sklearn.metrics import ...`	Import evaluation metrics
2	`model.predict(X_test)`	Generate predictions
3	`r2_score(y_test, y_pred)`	Calculate R^2 score
4	`mean_squared_error(y_test, y_pred)`	Calculate MSE (note: should use `squared=False` for RMSE)

Evaluation Metrics Explained:

Metric	Formula	Interpretation
R^2 Score	$1 - \frac{SS_{res}}{SS_{tot}}$	% of variance explained (0-1, higher is better)
MSE	$\frac{1}{n}\sum(y - \hat{y})^2$	Average squared error
RMSE	$\sqrt{MSE}$	Error in same units as target

R^2 Score Interpretation:

R^2 Value	Quality
> 0.9	Excellent
0.7 - 0.9	Good
0.5 - 0.7	Moderate
< 0.5	Poor

⚠ Code Note:

```
# The code shows:
print("RMSE:", mean_squared_error(y_test, y_pred))

# But this actually returns MSE, not RMSE. To get RMSE:
print("RMSE:", mean_squared_error(y_test, y_pred, squared=False))
# OR
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
```

---

8. Feature Importance Analysis

8.1 Extract Coefficients

```
importance = pd.DataFrame({
    'feature': features,
    'coefficient': model.coef_
}).sort_values(by='coefficient', ascending=False)

print(importance)
```

Line-by-Line Explanation:

Line	Code	Purpose
1	`pd.DataFrame({...})`	Create DataFrame from dictionary
2	`'feature': features`	Column 1: feature names
3	`'coefficient': model.coef_`	Column 2: regression coefficients
4	`.sort_values(by='coefficient', ascending=False)`	Sort by importance

Coefficient Interpretation:

Coefficient Sign	Meaning
Positive	Increases revenue
Negative	Decreases revenue
Larger absolute value	Stronger impact

Example Output:

Feature	Coefficient	Interpretation
META_FACEBOOK_SPEND	2.5	$1 Facebook spend → $2.50 revenue
GOOGLE_PAID_SEARCH_SPEND	1.8	$1 Google spend → $1.80 revenue
month	-50	Seasonal effect (summer slump?)

Marketing ROI Calculation:

$$ROI_{channel} = \text{Coefficient} - 1$$

Example: Coefficient = 2.5 → ROI = 150% (for every $1 spent, get $1.50 profit)

---

9. Budget Shift Simulation

9.1 Create Simulation Scenario

```
scenario = X_test.copy()
```

Why `.copy()` ?

- Creates independent copy of test data
- Modifications don't affect original `X_test`
- Enables comparison between scenarios

9.2 Simulate Budget Increase

```python
# Simulate increased Google Ads, decreased TV
scenario['GOOGLE_PAID_SEARCH_SPEND'] *= 1.3  # +30%
# scenario['TV_SPEND'] *= 0.8  # if applicable

y_simulated = model.predict(scenario)

# Compare with actual predicted
change = ((y_simulated.mean() - y_pred.mean()) / y_pred.mean()) * 100
print(f"Simulated change in revenue: {change:.2f}%")
```

Line-by-Line Explanation:

Line	Code	Purpose
1	`scenario['GOOGLE_PAID_SEARCH_SPEND'] *= 1.3`	Increase spend by 30%
2	`model.predict(scenario)`	Predict with new budget
3	`y_simulated.mean() - y_pred.mean()`	Calculate change in average revenue
4	`/ y_pred.mean() * 100`	Convert to percentage

Simulation Logic:

```
Original Budget → Original Predicted Revenue
    ↓ +30% Google Spend
Modified Budget → Simulated Revenue
    ↓ Compare
Revenue Change % = (Simulated - Original) / Original × 100
```

What-If Scenarios to Test:

Scenario	Code
+30% Google Spend	`scenario['GOOGLE_PAID_SEARCH_SPEND'] *= 1.3`
-20% Meta Spend	`scenario['META_FACEBOOK_SPEND'] *= 0.8`
Shift budget	Increase one, decrease another
Double email	`scenario['EMAIL_CLICKS'] *= 2`

9.3 Visualize Predictions

```
plt.figure(figsize=(12,6))
plt.plot(test_df['DATE_DAY'], y_test, label='Actual Revenue')
plt.plot(test_df['DATE_DAY'], y_pred, label='Predicted Revenue')
plt.title("Actual vs Predicted Revenue")
plt.legend()
plt.show()
```

Chart Components:

Element	Purpose
Blue line (Actual)	Real observed revenue
Orange line (Predicted)	Model's predictions
Gap between lines	Prediction error
Trend alignment	Model captures patterns

---

10. Model Persistence

10.1 Save Model

```
# Save the trained model
import joblib
joblib.dump(model, 'linear_mmm_model.pkl')
```

Explanation:

Code	Purpose
`import joblib`	Library for model serialization

Code	Purpose
`joblib.dump(model, ...)`	Save model to file
`'linear_mmm_model.pkl'`	Output filename (.pkl = pickle)

Why Use joblib?

Method	Best For
`joblib`	Large numpy arrays (sklearn models)
`pickle`	General Python objects
`json`	Simple data structures

10.2 Save Feature Names

```python
# Save the features used for training
import json
with open('mmm_model_features.json', 'w') as f:
    json.dump(features, f)
```

Line-by-Line Explanation:

Line	Code	Purpose
1	`import json`	JSON serialization library
2	`open('...', 'w')`	Open file for writing
3	`as f`	File handle
4	`json.dump(features, f)`	Write features list to JSON

Why Save Features Separately?

- Ensures prediction uses same features in same order
- Documentation of model inputs
- API needs to know expected inputs

mmm_model_features.json Contents:

```json
["GOOGLE_PAID_SEARCH_SPEND", "GOOGLE_SHOPPING_SPEND", "GOOGLE_PMAX_SPEND",
 "META_FACEBOOK_SPEND", "META_INSTAGRAM_SPEND", "EMAIL_CLICKS",
 "ORGANIC_SEARCH_CLICKS", "DIRECT_CLICKS", "BRANDED_SEARCH_CLICKS",
 "year", "month", "day_of_week"]
```

11. Flask API Deployment

11.1 Complete Flask Application

```
from flask import Flask, request, jsonify
import joblib
import json
import numpy as np

# Load model and features
model = joblib.load('linear_mmm_model.pkl')

with open('mmm_model_features.json', 'r') as f:
    feature_names = json.load(f)

app = Flask(__name__)

@app.route('/')
def home():
    return "MMM Model is running!"

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()

    # Extract input features in order
    X = [data.get(feat, 0) for feat in feature_names]
    X = np.array(X).reshape(1, -1)

    prediction = model.predict(X)[0]
    return jsonify({'predicted_revenue': prediction})

if __name__ == '__main__':
    print("Starting Flask server...")
    app.run(debug=True, use_reloader=False)
```

11.2 Import Statements

```
from flask import Flask, request, jsonify
import joblib
import json
import numpy as np
```

Library Purpose:

Library	Import	Purpose
`Flask`	Flask class	Create web application
`request`	Access request data	Get JSON from POST requests
`jsonify`	Create JSON responses	Return predictions as JSON
`joblib`	Model loading	Load saved sklearn model
`json`	JSON parsing	Load feature names
`numpy`	Array operations	Reshape input for prediction

11.3 Load Model and Features

```python
# Load model and features
model = joblib.load('linear_mmm_model.pkl')

with open('mmm_model_features.json', 'r') as f:
    feature_names = json.load(f)
```

Explanation:

Code	Purpose
`joblib.load(...)`	Deserialize saved model
`open(..., 'r')`	Open file for reading
`json.load(f)`	Parse JSON to Python list

Why Load at Module Level?

- Model loaded once when server starts
- Not reloaded for each request
- Faster response times

11.4 Create Flask Application

```python
app = Flask(__name__)
```

Explanation:

Code	Purpose
`Flask(__name__)`	Create Flask application instance
`__name__`	Module name (helps Flask find resources)
`app`	Application object for routing

11.5 Home Route

```
@app.route('/')
def home():
    return "MMM Model is running!"
```

Explanation:

Code	Purpose
`@app.route('/')`	Decorator: maps URL "/" to function
`def home()`	Function executed when "/" is accessed
`return "..."`	Response sent to client

Testing:

```
GET http://localhost:5000/
Response: "MMM Model is running!"
```

11.6 Prediction Endpoint

```
@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()

    # Extract input features in order
    X = [data.get(feat, 0) for feat in feature_names]
    X = np.array(X).reshape(1, -1)

    prediction = model.predict(X)[0]
    return jsonify({'predicted_revenue': prediction})
```

```
```

Line-by-Line Explanation:

Line	Code	Purpose
1	`@app.route('/predict', methods=['POST'])`	Only accept POST requests
2	`request.get_json()`	Parse JSON body from request
3	`data.get(feat, 0)`	Get feature value or default to 0
4	`[... for feat in feature_names]`	Build feature list in correct order
5	`np.array(X).reshape(1, -1)`	Convert to 2D array (1 sample, n features)
6	`model.predict(X)[0]`	Get prediction (single value)
7	`jsonify({...})`	Return as JSON response

Why `.reshape(1, -1)` ?

```python
# sklearn expects 2D array: (n_samples, n_features)
# Single prediction needs shape (1, 12) not (12,)

X = [100, 200, 150, ...]  # Shape: (12,) - 1D
X = np.array(X).reshape(1, -1)  # Shape: (1, 12) - 2D
```

API Request Example:

```bash
curl -X POST http://localhost:5000/predict \
  -H "Content-Type: application/json" \
  -d '{
    "GOOGLE_PAID_SEARCH_SPEND": 1000,
    "GOOGLE_SHOPPING_SPEND": 500,
    "GOOGLE_PMAX_SPEND": 300,
    "META_FACEBOOK_SPEND": 800,
    "META_INSTAGRAM_SPEND": 600,
    "EMAIL_CLICKS": 150,
    "ORGANIC_SEARCH_CLICKS": 200,
    "DIRECT_CLICKS": 100,
    "BRANDED_SEARCH_CLICKS": 50,
    "year": 2024,
    "month": 6,
    "day_of_week": 2
  }'
```

Response:

```
{
  "predicted_revenue": 15234.56
}
```

11.7 Run Server

```
if __name__ == '__main__':
    print("Starting Flask server...")
    app.run(debug=True, use_reloader=False)
```

Explanation:

Code	Purpose
`if __name__ == '__main__'`	Only run if executed directly (not imported)
`debug=True`	Enable debug mode (auto-reload, detailed errors)
`use_reloader=False`	Disable auto-reloader (prevents double loading)

Running the Server:

```
python mmm_app.py
# Output: Starting Flask server...
#  * Running on http://127.0.0.1:5000
```

---

12. Key Concepts Summary

12.1 MMM Workflow

```
┌──────────────────────────────────────────────────────────────┐
│  1. DATA PREPARATION                                         │
│     Load → Clean → Handle Missing Values                     │
└──────────────────────────────────────────────────────────────┘
                           ↓
┌──────────────────────────────────────────────────────────────┐
```

```
│   2. FEATURE ENGINEERING                                          │
│      Date Features → Revenue Calculation → ROI                    │
└───────────────────────────────────────────────────────────────────┘
                            ↓
┌───────────────────────────────────────────────────────────────────┐
│   3. MODELING                                                     │
│      Time-Based Split → Linear Regression → Evaluate              │
└───────────────────────────────────────────────────────────────────┘
                            ↓
┌───────────────────────────────────────────────────────────────────┐
│   4. ANALYSIS                                                     │
│      Coefficients → Channel ROI → Budget Recommendations          │
└───────────────────────────────────────────────────────────────────┘
                            ↓
┌───────────────────────────────────────────────────────────────────┐
│   5. SIMULATION                                                   │
│      What-If Scenarios → Optimal Budget Allocation                │
└───────────────────────────────────────────────────────────────────┘
                            ↓
┌───────────────────────────────────────────────────────────────────┐
│   6. DEPLOYMENT                                                   │
│      Save Model → Flask API → Production                          │
└───────────────────────────────────────────────────────────────────┘
```

12.2 Key Metrics for MMM

Metric	Formula	Business Meaning
ROI	(Revenue - Spend) / Spend	Return per dollar spent
ROAS	Revenue / Spend	Revenue per dollar spent
Coefficient	Model output	Revenue generated per unit input
R^2	Model fit	How well model explains variance

12.3 Files Created

File	Type	Purpose
`linear_mmm_model.pkl`	Binary	Trained sklearn model
`mmm_model_features.json`	JSON	Feature names list
`mmm_app.py`	Python	Flask API application

12.4 API Endpoints Summary

Endpoint	Method	Purpose	Response
`/`	GET	Health check	"MMM Model is running!"

Endpoint	Method	Purpose	Response
`/predict`	POST	Get revenue prediction	`{"predicted_revenue": float}`

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

A complete reference for Marketing Mix Modeling and API deployment