# How to Use My Bayesian Template

Summary & Client Adaptation

This summary document outlines the key principles of using the modular Bayesian marketing analytics template across multiple clients. It includes:

1. **Template Structure**
2. **Client-Specific Customizations**
3. **Best Practices for Duplication & Adaptation**

## 1. Template Structure

The core template (bayesian\_ads.ipynb) relies on the following modules:

* **modules/data\_loader.py**: Stubs for data ingestion from Meta Ads, LinkedIn, GA4.
* **modules/bayesian\_model.py**: Statistical logic for prior/posterior calculations.
* **modules/visualization.py**: Functions for plotting and saving posterior distributions.
* **modules/openai\_client.py**: Generates narrative insights using GPT-4o.
* **modules/reporting.py**: Assembles HTML report and exports to PDF.
* **modules/semantics.py** (optional): Narrative functions for explaining Bayesian concepts.

Notebook cells orchestrate these modules in a logical sequence (setup → load → compute → visualize → narrate → export).

## 2. Client-Specific Customizations

For each new client, you will typically modify:

### Data Loaders:

Replace stub functions in modules/data\_loader.py with real API integrations or CSV loader functions.

### Configuration Files:

Create a JSON file in the /configs directory (e.g., client\_xyz\_config.json), which holds all client-specific settings in one place.

**This file should include:**

{

"api\_credentials": {

"meta": {

"access\_token": "YOUR\_META\_ACCESS\_TOKEN",

"ad\_account\_id": "act\_123456789"

},

"linkedin": {

"client\_id": "YOUR\_LINKEDIN\_CLIENT\_ID",

"client\_secret": "YOUR\_LINKEDIN\_CLIENT\_SECRET",

"access\_token": "YOUR\_LINKEDIN\_ACCESS\_TOKEN"

},

"ga4": {

"property\_id": "YOUR\_GA4\_PROPERTY\_ID",

"credentials\_path": "/path/to/ path/to/service-account.json"

}

},

"campaign\_mappings": {

"ad\_names": ["Ad A", "Ad B"],

"campaign\_ids": ["123", "456"]

},

"report\_settings": {

"brand\_logo\_path": "configs/client\_xyz\_logo.png",

"brand\_colors": {

"primary": "#0044cc",

"secondary": "#ff6600"

}

}

}

**To use this in code:**

import json

with open("configs/client\_xyz\_config.json") as f:

config = json.load(f)

meta\_token = config["api\_credentials"]["meta"]["access\_token"]

campaign\_ids = config["campaign\_mappings"]["campaign\_ids"]

This approach keeps secrets and mappings out of your code, centralizes client settings, and can be safely excluded from version control via .gitignore.

### Styling & Branding:

* Update HTML/CSS in modules/reporting.py to include client logos, brand colors, and font choices.
* Adjust layout (e.g., add cover page or additional sections).

### Narrative Tone:

Pass a tone parameter to generate\_insight\_narrative(), or create a customized system prompt for each client.

## 3. Best Practices for Duplication & Adaptation

1. **Branch & Tag**: Create a new Git branch for each client. Tag releases (e.g., client\_xyz\_v1.0).
2. **Maintain Config Files**: Keep client credentials and settings in /configs, never in code. Use .gitignore to avoid leaking secrets.
3. **Version Control**: Commit changes specific to each client; merge upstream template improvements periodically.
4. **Documentation**: In each client’s folder, add a README.md with client-specific instructions (e.g., which cells to run, API key placement).
5. **Testing**: For each client, write a small test script (e.g., test\_client\_xyz.py) that calls each data loader and ensures valid data shapes.