Issues Identified in the Old Project

The original dbt project had several design inefficiencies and areas that deviated from dbt best practices. Below is a detailed analysis:

**a. Fragmented Project Structure**

* Each model was treated as a separate dbt project, requiring individual execution.
* This approach caused:
  + Increased complexity in managing dependencies between models.
  + Inefficiency due to multiple execution steps for a single environment.

**b. Overuse of Custom Scripts**

* A shell script (dbt\_command\_loop.sh) was used to orchestrate model executions. This is redundant because dbt already provides commands to manage models dynamically.

**c. Hardcoded Environment Configurations**

* Files like preprod\_table\_config.yml and table\_config.yml were used for environment-specific configurations.
* This made the project rigid, harder to maintain, and less scalable for additional environments.

**d. Lack of Centralized Control**

* There was no centralized management for:
  + Targets (e.g., preprod and prod environments).
  + Configuration settings.
  + Model execution order and dependencies.

**e. Non-Adherence to dbt Best Practices**

* dbt best practices encourage a single, well-structured project with models organized logically, using environment-specific configurations within profiles.yml.

Steps Taken to Resolve the Issues

**a. Unified Project Structure**

* Consolidated all models into a single dbt project under the models/ folder.
* Organized models based on functionality and environment:
  + Created subdirectories like models/preprod/ and models/prod/ for environment-specific models.

**b. Leveraged dbt’s Native Environment Configuration**

* Used profiles.yml to define environment-specific configurations (e.g., preprod, prod) with parameters such as host, schema, database, and credentials.
* Example of profiles.yml:

yaml

Copy code

unified\_project:

target: preprod

outputs:

preprod:

type: postgres

host: preprod\_host

schema: preprod\_schema

user: preprod\_user

password: preprod\_password

port: 5432

prod:

type: postgres

host: prod\_host

schema: prod\_schema

user: prod\_user

password: prod\_password

port: 5432

**c. Simplified Model Execution**

* Removed dbt\_command\_loop.sh and replaced it with a single dbt run command for each environment:
  + Example:

dbt run --target preprod

dbt run --target prod

**d. Enabled Dynamic Logic in Models**

* Added Jinja logic in model configurations to enable or disable models based on the target environment:

{{ config(

enabled=target.name == 'preprod'

) }}

SELECT ...

**e. Modularized the Code Using Macros**

* Replaced repeated logic with reusable macros for cleaner and more maintainable code.
* Example macro for dynamic schema naming:

{% macro generate\_schema\_name(custom\_schema\_name, node) %}

{{ target.schema }}

{% endmacro %}

**f. Improved Documentation**

* Added a clear and concise README explaining the project’s setup, execution, and design choices.

**g. Tested for Both Environments**

* Ran models using dbt run for both preprod and prod environments.
* Verified that only environment-specific models were executed in each case.

**3. Results Achieved**

* **Efficiency**: Reduced complexity by consolidating multiple projects into a single dbt project.
* **Scalability**: The project now supports multiple environments without requiring additional scripts or configuration files.
* **Maintainability**: Centralized control of configurations and dependencies makes the project easier to understand and manage.
* **Compliance with dbt Best Practices**:
  + Used profiles.yml for target configurations.
  + Centralized and logically structured models.
  + Eliminated redundant scripts.