

Model Refitment Strategy



I am doing model refitment works & I have completed few of them already. But I want to do more on it

That's great, Nazmuddin! Working on **model refitment** (i.e., retraining or fine-tuning existing models) is a valuable and practical part of maintaining machine learning systems, especially in real-world applications where **data drift**, **concept drift**, or **performance decay** are common.

Let's structure your continued efforts into a **deeper, more strategic learning and hands-on plan**.

✓ What You've Likely Done So Far

You mentioned you've already completed a few refitment tasks. These usually include:

- Re-training models with updated datasets
- Hyperparameter tuning
- Re-evaluating performance using metrics like accuracy, F1-score, AUC
- Maybe even adjusting feature sets

What More You Can Do – Expand Your Skills & Projects

1. Data Drift & Concept Drift Detection

- Learn how to detect if the data distribution has changed.
- Use libraries like:
 - **evidently**
 - **alibi-detect**
 - **River** (for online learning + drift)

Hands-on idea: Simulate data drift and build a pipeline to refit models when drift is detected.

2. Automated Model Refit Pipelines

- Automate model monitoring and retraining using:
 - **MLflow** or **DVC** for model versioning
 - **Airflow** or **Prefect** for scheduling and orchestration
 - **Docker** + **FastAPI** to deploy models

Mini Project: Create a pipeline that:

- Checks accuracy weekly
 - If performance drops > X%, it retrains with new data
 - Logs old and new metrics for comparison
-

3. Use Case: Industry-Based Refit Examples

Pick a domain you're familiar with (facilities management, healthcare, etc.) and apply refitment:

- **Healthcare:** Refitting a sepsis prediction model with 3-month ICU data
 - **Retail:** Retraining sales forecasting model with last 6 months of transaction data
 - **Facilities:** Equipment failure prediction model retrained with newly collected sensor data
-

4. Model Explainability After Refit

- Use SHAP or LIME to compare feature importance before and after refit.
 - Identify if the model is learning the same or different things over time.
-

5. Refitting Non-Tree Models

- Try refitting **logistic regression, SVM, neural networks**, not just decision trees/XGBoost.
 - See how each responds to updated data.
 - Explore techniques like **partial fit** in models that support it (e.g., SGDClassifier).
-

6. Refit vs Rebuild Study

- Create a comparative notebook where:
 - You **refit an existing model** and evaluate performance
 - You **build a new model from scratch** with the same data
 - Compare: Which is better and when?
-

Want Practical Guidance?

Let me know if you'd like:

- A **template project structure** for automated refit pipelines
 - Simulated **datasets for refit exercises**
 - Sample code snippets to detect data drift
 - Guidance on tools like MLflow, Airflow, etc.
-

Would you like help building a full mini project on **automated model refit pipeline with monitoring + drift detection**?



Since I have done model refitment , i want to improve model refitment work by adding value addition. I want to change the code so that it will do the model refitment without manual work, I want to change the code so that it can be used for any model & dataset with minimal manual work

That's a **great next step**, Nazmuddin — you're aiming to **generalize and automate** the model refitment process, which is exactly how real-world MLOps pipelines are designed. You're moving from "just doing" to "doing smartly and scalably." Let's break this down and then I'll share a **starter code framework** you can build on.

✓ Objective

Create a **generic, automated model refitment pipeline** that:

1. Detects if refit is needed
2. Loads any dataset and model dynamically
3. Preprocesses the data (minimally)
4. Refits the model
5. Evaluates and logs results
6. Saves the updated model

All with **minimal manual work**.

🧠 Key Value-Add Features to Include

Feature	Value it Adds
📦 Model/dataset agnostic code	Reusable across projects
⚙️ Automated data loading/preprocessing	Saves time
📊 Performance comparison (before vs after)	Justifies refit
🔍 Drift detection (optional)	Smart refitting trigger
🌱 Evaluation + version control	Trust + rollback option
💾 Save model with version/date	Model management

📦 Suggested Stack

- Python (core)
- **scikit-learn** for modeling

- `joblib` or `pickle` for saving models
- `pandas` for data
- `os` / `argparse` for automation
- (optional) `evidently`, `mlflow`, `prefect` for scaling later

Basic Python Code Template (Starter)

Here's a simple refitment pipeline you can extend:

python

```
import pandas as pd
import joblib
import os
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier

def load_data(file_path):
    df = pd.read_csv(file_path)
    return df

def preprocess(df, target_col):
    # Auto label encode for simplicity
    for col in df.select_dtypes(include='object').columns:
        if col != target_col:
            df[col] = LabelEncoder().fit_transform(df[col])

    X = df.drop(target_col, axis=1)
    y = df[target_col]
    return train_test_split(X, y, test_size=0.2, random_state=42)

def load_existing_model(model_path):
    if os.path.exists(model_path):
        return joblib.load(model_path)
    return None

def train_model(model, X_train, y_train):
    model.fit(X_train, y_train)
    return model

def evaluate(model, X_test, y_test):
    preds = model.predict(X_test)
    acc = accuracy_score(y_test, preds)
    return acc

def save_model(model, save_path):
    joblib.dump(model, save_path)

# === DRIVER CODE ===
if __name__ == "__main__":
    dataset_path = "your_dataset.csv"
    target_column = "target"
    model_path = "model.pkl"

    df = load_data(dataset_path)
    X_train, X_test, y_train, y_test = preprocess(df, target_column)

    existing_model = load_existing_model(model_path)
    if existing_model:
        print("Existing model found. Re-training...")
        model = train_model(existing_model, X_train, y_train)
    else:
        print("Training new model...")
        model = train_model(RandomForestClassifier(), X_train, y_train)
```

```
accuracy = evaluate(model, X_test, y_test)
print(f"Model Accuracy: {accuracy:.2f}")

save_model(model, model_path)
print("Model saved after refit.")
```

How to Make it Model- & Dataset-Agnostic

✓ Accept arguments using **argparse**:

```
bash

python refit_pipeline.py --data data.csv --target target --model model.pkl
```

✓ Use **importlib** to load any model class dynamically.

✓ Optional: Save evaluation metrics in a log file or **.json**.

✓ Bonus: Add a flag for **--check-drift** using **evidently**.

Next Steps You Can Try

1. **Add drift detection**: Only refit if drift detected.
2. **Support for sklearn pipelines**.
3. **Log comparison**: old accuracy vs new accuracy.
4. **Auto-schedule** with a cron job or **prefect** for production-level automation.
5. **MLflow** integration for tracking.





Would you like me to help build a more advanced version with CLI support, model drift checking, and automated logging? Or do you want a project folder template you can reuse across projects?



I will list out the works I do during reitment, please show how to automate & generalize the same

Perfect! Please go ahead and **list the steps** or tasks you perform during model **refitment** — the more detailed you are (e.g., file reading, feature selection, model loading, evaluation, etc.), the better I can help you **automate and generalize each part**.

Once I see your full list, I'll respond with:




-  Which parts can be fully automated
-  Generic code templates for each task
-  Suggestions for improvement and modularization
-  Optional tools/libraries to help automate even more

Please list your **refitment workflow** now.



I will start with firts task

Great, Nazmuddin! Please go ahead and share your **first task** in the model refitment process. After each one, I'll show you:

-  How to **automate** it
-  How to **generalize** it
-  Any **value additions** to improve the workflow

Let's begin!



I firstly create a populaiaon table from production data bases from Snowflake. I get population data from three tables - input table which has input features in json format in text_ftrs column, target variable in ACTL table & metrics of existing model from OTPT table. I have to use joins, need to try_cast to change input featurrs from json.

Excellent start! You're describing a **realistic enterprise-grade task** in model refitment — pulling production data from Snowflake and transforming it for ML consumption. Let's break this down and walk through **how to automate and generalize it** step by step.

Your Task 1: Create Population Table from Snowflake

Input:

- **Source DB:** Snowflake
- **Tables:**
 - **INPUT** → column: **text_ftrs** (JSON string with features)
 - **ACTL** → target variable (e.g., label, outcome)
 - **OTPT** → existing model metrics (for comparison/logging)

Actions You Perform:

1. Connect to Snowflake.
2. Join these tables using key(s).
3. Parse JSON from **text_ftrs** to columns.
4. Select features and target.
5. Create final "population table" DataFrame for modeling.

Automation & Generalization Plan

1. Configurable Snowflake Connection

Use environment variables or a .env/YAML config for flexibility.

```
python

import snowflake.connector
import pandas as pd
import json

def connect_snowflake():
    conn = snowflake.connector.connect(
        user='YOUR_USERNAME',
        password='YOUR_PASSWORD',
        account='YOUR_ACCOUNT',
        warehouse='YOUR_WAREHOUSE',
        database='YOUR_DB',
        schema='YOUR_SCHEMA',
        role='YOUR_ROLE'
    )
    return conn
```

2. Generalized SQL to Join & Extract JSON Fields

Use `try_cast()` or `parse_json()` in Snowflake SQL. This example assumes a common `ID` key:

```
sql

SELECT
    i.ID,
    i.text_fts,
    a.target_column,
    o.metric_name,
    o.metric_value
FROM INPUT i
JOIN ACTL a ON i.ID = a.ID
LEFT JOIN OTPT o ON i.ID = o.ID
```

Then in Python:

```
python

def fetch_population_table(conn):
    query = """YOUR_JOIN_QUERY"""
    df = pd.read_sql(query, conn)

    # Parse JSON from text_fts
    json_df = df['text_fts'].apply(json.loads).apply(pd.Series)

    # Drop text_fts and merge
    df = df.drop(columns=['text_fts']).join(json_df)
    return df
```

3. Make It Model/Dataset-Agnostic with a YAML/JSON Config

Let your code read from a config file like this:

```
yaml

tables:
  input: "INPUT"
  target: "ACTL"
  metrics: "OTPT"
join_key: "ID"
json_column: "text_fts"
target_column: "target_column"
```

Then load config dynamically:

```
python

import yaml

def load_config(path="config.yaml"):
    with open(path, 'r') as f:
        return yaml.safe_load(f)
```

4. Wrap All in a Reusable Function

```
python

def get_population_table(config):
    conn = connect_snowflake()

    query = f"""
    SELECT
        i.{config['join_key']],
        i.{config['json_column']],
        a.{config['target_column']],
        o.metric_name,
        o.metric_value
    FROM {config['tables']['input']} i
    JOIN {config['tables']['target']} a ON i.{config['join_key']} = a.{config['join_key']}
    LEFT JOIN {config['tables']['metrics']} o ON i.{config['join_key']} = o.{config['join_key']}
    """

    df = pd.read_sql(query, conn)
    json_df = df[config['json_column']].apply(json.loads).apply(pd.Series)
    df = df.drop(columns=[config['json_column']]).join(json_df)

    conn.close()
    return df
```

Output:

- Fully cleaned and joined population DataFrame, with target column and expanded features from JSON.

Value Add Ideas

Feature	Benefit
 .env file for secrets	Secure and portable
 Add assert checks for missing values or columns	Robust pipeline

Feature	Benefit
□ Log query time and size	Helps with debugging
📁 Save population table as .parquet	Faster reloads

👉 Next Steps:

Shall we move to the **next task** in your refitment workflow — like **feature selection, preprocessing, or model loading**?

Let me know the **next step**, and I'll show how to automate that too.



I want to use snowflake app to create population table as population table has millions of record & i cant load it in domino python notebook.

Got it — excellent decision. Since your **population table has millions of records**, and Domino Python notebooks can't handle it efficiently, using a **Snowflake Native App or Snowflake worksheet** is the right scalable choice.

Let's now shift gears to:

✅ Objective:

Use Snowflake directly to generate and store the population table inside Snowflake itself — avoiding heavy data pulls into Domino.

Later, you can **sample** or **export** a manageable subset for model refitment in Python.

✅ Recommended Approach

🧱 Step 1: Create a Population Table in Snowflake (in SQL)

Instead of fetching raw data into Domino and joining/parsing JSON there, do all the heavy lifting **inside Snowflake** using SQL.

Here's a generalized SQL template:

```
sql

CREATE OR REPLACE TABLE MY_DB.MY_SCHEMA.POPULATION_TABLE AS
SELECT
  i.ID,
  a.target_column AS target,
  o.metric_name,
  o.metric_value,
  -- Extract and cast fields from JSON safely
  TRY_CAST(PARSE_JSON(i.text_fters):feature_1 AS FLOAT) AS feature_1,
  TRY_CAST(PARSE_JSON(i.text_fters):feature_2 AS STRING) AS feature_2,
  TRY_CAST(PARSE_JSON(i.text_fters):feature_3 AS BOOLEAN) AS feature_3
  -- add more fields as needed
FROM MY_DB.MY_SCHEMA.INPUT i
```

```
JOIN MY_DB.MY_SCHEMA.ACTL a ON i.ID = a.ID
LEFT JOIN MY_DB.MY_SCHEMA.OTPT o ON i.ID = o.ID;
```

Use `PARSE_JSON()` to extract features directly in SQL from the `text_fts` column.

Test Query First

Before creating a table, run a quick test with `LIMIT 10`:

```
sql

SELECT
  i.ID,
  TRY_CAST(PARSE_JSON(i.text_fts):feature_1 AS FLOAT) AS feature_1,
  a.target_column,
  o.metric_value
FROM INPUT i
JOIN ACTL a ON i.ID = a.ID
LEFT JOIN OTPT o ON i.ID = o.ID
LIMIT 10;
```

Once satisfied, use `CREATE OR REPLACE TABLE` to persist it.

Step 2: Use Snowflake App / Worksheet

If you're using the **Snowflake UI App**:

- Navigate to **Worksheets**
- Choose the **correct warehouse**
- Paste the `CREATE TABLE SQL`
- Run it — it will create a persistent, joined table

Now you can easily:

- `SELECT * FROM POPULATION_TABLE LIMIT 10000` in Domino
- Or export small subsets using `COPY INTO @stage` to CSV

Value Add Ideas

Enhancement	Purpose
Use <code>WAREHOUSE = XLARGE</code> for performance	Faster execution on large data
Schedule refresh with <code>TASKS</code>	Keep data always up to date
Add basic profiling: <code>COUNT(*)</code> , <code>NULL %</code> , <code>DISTINCT</code>	Check data quality
Log SQL logic in version-controlled Git repo	Maintain pipeline traceability

Optional: Create a Materialized View (if real-time needed)

```
sql
```

```
CREATE OR REPLACE MATERIALIZED VIEW POPULATION_VIEW AS
SELECT ...
```

Next: Move Light Sample to Domino

Now that you've built the population table inside Snowflake, you can:

- Sample 50k–100k rows with **WHERE** clause
- Pull only that data into Domino for training/refitting

Would you like help writing the Python + Snowflake **snowflake-connector** code to **pull a sample** from the new population table into Domino, or move on to the **next step in your refitment pipeline** (like preprocessing/model training)?



share details related to snowflake-connector. Also explain me the below code,

```
TRY_CAST(PARSE_JSON(i.text_fts):feature_1 AS FLOAT) AS feature_1
```

Great! Let's do two things as you requested:

1. Understanding Snowflake SQL:

Code:

```
sql
```

```
TRY_CAST(PARSE_JSON(i.text_fts):feature_1 AS FLOAT) AS feature_1
```

Explanation in Layman's Terms:

You are working with a column called **text_fts** which stores **JSON-formatted feature data** as plain text. For example, a value in **text_fts** might look like:

```
json
```

```
{
  "feature_1": 23.7,
  "feature_2": "abc",
  "feature_3": true
}
```

Now the goal is to **extract each key-value pair from the JSON** and turn it into a normal SQL column.

Step-by-Step Breakdown:

Function	What it does
<code>PARSE_JSON(i.text_fts)</code>	Converts the text string into a JSON object that Snowflake can query like a dictionary
<code>:feature_1</code>	Accesses the value of the feature_1 key from the JSON
<code>TRY_CAST(... AS FLOAT)</code>	Converts the value to a FLOAT data type (if possible); returns NULL if the value can't be converted (instead of error)
<code>AS feature_1</code>	Names the output column feature_1

✅ **Result:** You get a proper numeric column **feature_1** in your output table, ready for ML use.

✅ 2. Snowflake Connector for Python (snowflake-connector-python)

📦 What It Is:

A Python package to **connect to Snowflake**, run SQL queries, and fetch data into pandas DataFrames.

🔧 Install It:

```
bash

pip install snowflake-connector-python pandas
```

🧱 Basic Usage Template

```
python

import snowflake.connector
import pandas as pd

# Step 1: Connect to Snowflake
conn = snowflake.connector.connect(
    user='your_username',
    password='your_password',
    account='your_account.region.gcp/aws/azure',
    warehouse='your_warehouse',
    database='your_database',
    schema='your_schema',
    role='your_role'
)

# Step 2: Write Query (could be SELECT from POPULATION table)
query = """
SELECT * FROM POPULATION_TABLE
WHERE target IS NOT NULL
LIMIT 100000
"""

# Step 3: Load data into pandas DataFrame
df = pd.read_sql(query, conn)

# Step 4: Close connection
conn.close()
```

🔒 **Tip: Use .env or os.environ to hide credentials:**

```
python

import os
from dotenv import load_dotenv
load_dotenv()

conn = snowflake.connector.connect(
    user=os.getenv("SNOW_USER"),
    password=os.getenv("SNOW_PASS"),
    account=os.getenv("SNOW_ACCOUNT"),
    warehouse=os.getenv("SNOW_WAREHOUSE"),
    database=os.getenv("SNOW_DATABASE"),
    schema=os.getenv("SNOW_SCHEMA"),
    role=os.getenv("SNOW_ROLE")
)
```

Combine: Full Pipeline Idea

1. Use Snowflake worksheet to create and store population table using SQL (`TRY_CAST(PARSE_JSON...)`).
2. In Domino or locally, use `snowflake-connector-python` to fetch a **sampled subset** for ML refitment.
3. Store or version the final ML dataset locally or in S3/GCS/etc.

Would you like me to create a **ready-to-run Python template** with `.env` integration and query a sample from `POPULATION_TABLE` next?



```
explain:
if __name__ == "__main__":
```

Certainly, Nazmuddin! The line:

```
python



if __name__ == "__main__":
```

is one of the **most important** and **commonly used** Python constructs when writing scripts and building reusable code modules. Let's break it down in simple terms:

What it Means (Layman's Explanation)

 **"name" is a built-in variable in Python.**

It tells you **how** a Python file is being run.

- If you run the file **directly** (like `python my_script.py`),
 then `__name__` is set to `"__main__"`
- But if you **import that file** into another Python script (like `import my_script`),
 then `__name__` becomes the **name of the file/module**, like `"my_script"`

✓ Why Use `if __name__ == "__main__":`?

To **control what code gets run when**.

📌 Example:

Let's say you have a file `refit_pipeline.py` with this code:

```
python

def refit_model():
    print("Model is being retrained...")

if __name__ == "__main__":
    refit_model()
```

- If you run it directly:

```
bash

python refit_pipeline.py
```

👉 Output:

```
csharp

Model is being retrained...
```

- But if you import it in another script:

```
python

import refit_pipeline
```

👉 No output! (because `__name__` is `"refit_pipeline"`, not `"__main__"`)

🎯 Why It's So Useful

Benefit	Why it matters
✓ Prevents unwanted code execution	Clean module imports
✓ Lets you reuse functions	You can build libraries, pipelines
✓ Separates logic from script entry point	Good software design

💡 Analogy

Think of `if __name__ == "__main__"` like a **main switch** in a machine:

- If you're **running** the machine (script directly), the main switch turns things on.

- If you're just **inspecting** the machine (importing it elsewhere), it stays off.

✓ When to Use It

Always use it when:

- You're writing scripts that can both **run on their own** *and* be **imported by others**
- You want a **clear entry point** for your Python program (just like `main()` in C, Java, etc.)

Let me know if you'd like a **real-life example** from your model refitment workflow using this construct!



what is class & sub class in python?


Great question! Understanding **classes** and **subclasses** in Python is key to mastering **object-oriented programming (OOP)** — which is useful when building reusable, modular, and scalable ML pipelines or data tools.

Let's break this down **step-by-step in plain English** with real-life relatable analogies and code examples.

What is a Class in Python?

A **class** is a **blueprint** for creating objects (like a template).

Think of it like this:

 A **class** is a "car factory blueprint" — it defines what a car should have (color, engine, wheels) and what it can do (drive, honk).

Then you use this class to create **objects** (actual cars).

✓ Example: Defining a Class in Python

```
python

class Car:
    def __init__(self, brand, color):
        self.brand = brand
        self.color = color

    def drive(self):
        print(f"The {self.color} {self.brand} is driving.")

# Create object
my_car = Car("Toyota", "Red")
my_car.drive()
```

Output:

```
csharp
```

```
The Red Toyota is driving.
```


Key parts:

Part	Meaning
<code>class Car:</code>	You are defining a class named Car
<code>__init__</code>	Constructor: runs when object is created
<code>self</code>	Refers to the object being created
<code>drive()</code>	Method (a function inside a class)
<code>my_car = Car(...)</code>	Creates an object from the class

What is a Subclass in Python?

A **subclass** is a class that **inherits** from another class.

It **reuses the parent class code**, and can **add or override** behavior.

 Think of a **SportsCar** as a subclass of **Car** — it's still a car, but with extra features like turbo or convertible.

✓ Example: Subclass in Python

```
python
```

```
class SportsCar(Car): # Inherits from Car
    def turbo_boost(self):
        print(f"The {self.brand} boosts with turbo!")

# Create SportsCar object
ferrari = SportsCar("Ferrari", "Yellow")
ferrari.drive()
ferrari.turbo_boost()
```

Output:

```
csharp
```

```
The Yellow Ferrari is driving.
The Ferrari boosts with turbo!
```

What's Happening?

- **SportsCar** automatically gets all methods/attributes from **Car**
- You **didn't need to re-write** `__init__()` or `drive()`
- You **added** a new method `turbo_boost()`

Why Classes & Subclasses Matter in Real Projects

Use Case	Benefit of Using Classes
ML pipeline (e.g., preprocessing → train → predict)	Organize logic clearly
Refitment modules (base logic + variations)	Reuse parent code, override only where needed
Custom transformers or estimators in sklearn	Use BaseEstimator , TransformerMixin via subclassing
Simulation, automation tools	Encapsulation of properties and behaviors

Real-Life Example (Refitment Project)

```
python

class BaseRefitter:
    def load_data(self):
        raise NotImplementedError

    def preprocess(self):
        raise NotImplementedError

    def refit_model(self):
        print("Default refit logic")

class SnowflakeRefitter(BaseRefitter):
    def load_data(self):
        print("Fetching data from Snowflake...")

# Usage
refitter = SnowflakeRefitter()
refitter.load_data()
refitter.refit_model()
```

Summary Table

Concept	Meaning
Class	Blueprint to create objects
Object	Instance created from a class
Subclass	Class that inherits from another
<code>__init__</code>	Special method to initialize an object
<code>self</code>	Refers to the current object
Inheritance	Reuse code from base/parent class

Let me know if you'd like to see a **data science example** using class & subclass (e.g., custom sklearn transformers or refitter modules)!



what dos the below mean?

Use environment variables or a .env/YAML config for flexibility.

Great follow-up! That line:

"Use environment variables or a .env/YAML config for flexibility"

means:

🔒 Instead of **hardcoding credentials, file paths, or settings** in your Python script, you store them in **external files or environment variables** and read them at runtime.

✓ Why Use This?

1. **Security:** Keeps sensitive info (like DB passwords) out of your code.
2. **Reusability:** Easily switch between dev/test/prod environments.
3. **Clean code:** Makes your script generic and readable.

🔒 1. Environment Variables

These are variables set **outside Python**, usually in your operating system or Docker/container environment.

🎯 Example:

```
bash

export SNOW_USER='nazm_user'
export SNOW_PASS='secret123'
export SNOW_ACCOUNT='abc-xy123'
```

🐍 In Python:

```
python

import os

user = os.getenv("SNOW_USER")
password = os.getenv("SNOW_PASS")
account = os.getenv("SNOW_ACCOUNT")
```

📄 2. .env File (Common in Projects)

Instead of manually setting env vars, store them in a .env file like:

```
ini

SNOW_USER=nazm_user
SNOW_PASS=secret123
SNOW_ACCOUNT=abc-xy123
```

Python reads it using:

```
bash
```

```
pip install python-dotenv
```

```
python
```

```
from dotenv import load_dotenv
import os

load_dotenv() # Loads from .env

user = os.getenv("SNOW_USER")
```

✓ You now have secure, portable, and non-hardcoded access.

3. YAML Config File (Great for structured config)

For more complex configuration like column names, table names, or ML model settings, use YAML:

config.yaml:

```
yaml
```

```
snowflake:
  user: nazm_user
  password: secret123
  account: abc-xyz123
  warehouse: COMPUTE_WH
  database: PROD_DB
  schema: ML
```

Python reads it using:

```
bash
```

```
pip install pyyaml
```

```
python
```

```
import yaml

with open("config.yaml", "r") as file:
    config = yaml.safe_load(file)

print(config["snowflake"]["user"])
```

✓ Summary Comparison

Method	Use Case	Tool
os.environ or .env	Secrets like DB creds, keys	python-dotenv




Method	Use Case	Tool
YAML config	Structured configs, ML settings, table/column names	pyyaml

Do you want a **working template** that loads `.env` and YAML together for your Snowflake refitment project?



yes

Perfect, Nazmuddin! Here's a **complete, modular Python template** that shows how to:

1.  Load **Snowflake credentials securely** from a `.env` file
2.  Load **configurable settings** (like table names, target column, etc.) from a `config.yaml`
3.  Connect to Snowflake and fetch data using those configs

Project Structure (Recommended)

```

arduino

refit_project/
├── .env
├── config.yaml
├── refit_pipeline.py
└── requirements.txt

```

✓ 1. `.env` (do NOT upload to GitHub)

```

dotenv

SNOW_USER=nazm_user
SNOW_PASS=your_password_here
SNOW_ACCOUNT=your_account_id
SNOW_WAREHOUSE=COMPUTE_WH
SNOW_DATABASE=PROD_DB
SNOW_SCHEMA=ML
SNOW_ROLE=SYSADMIN

```

✓ 2. `config.yaml`

```

yaml

tables:
  input: "INPUT"
  target: "ACTL"
  metrics: "OTPT"
join_key: "ID"
json_column: "text_ftrs"
target_column: "target"

```

```
features:
- feature_1
- feature_2
- feature_3
```

✓ 3. refit_pipeline.py

python

```
import os
import json
import pandas as pd
import snowflake.connector
from dotenv import load_dotenv
import yaml

# Load env vars
load_dotenv()

# Load YAML config
with open("config.yaml") as f:
    config = yaml.safe_load(f)

def connect_snowflake():
    return snowflake.connector.connect(
        user=os.getenv("SNOW_USER"),
        password=os.getenv("SNOW_PASS"),
        account=os.getenv("SNOW_ACCOUNT"),
        warehouse=os.getenv("SNOW_WAREHOUSE"),
        database=os.getenv("SNOW_DATABASE"),
        schema=os.getenv("SNOW_SCHEMA"),
        role=os.getenv("SNOW_ROLE")
    )

def build_query(config):
    features = ",\n".join([
        f"TRY_CAST(PARSE_JSON(i.{config['json_column']}):{feat} AS STRING) AS {feat}"
        for feat in config['features']
    ])

    query = f"""
SELECT
    i.{config['join_key']},
    a.{config['target_column']} AS target,
    o.metric_value,
    {features}
FROM {config['tables']['input']} i
JOIN {config['tables']['target']} a
    ON i.{config['join_key']} = a.{config['join_key']}
LEFT JOIN {config['tables']['metrics']} o
    ON i.{config['join_key']} = o.{config['join_key']}
"""
    return query

def fetch_data():
    conn = connect_snowflake()
    query = build_query(config)
    df = pd.read_sql(query, conn)
    conn.close()
    return df

if __name__ == "__main__":
    df = fetch_data()
    print("Sample data:\n", df.head())
    print(f"\n✓ Total rows fetched: {len(df)}")
```

✓ 4. requirements.txt

```
txt

pandas
snowflake-connector-python
python-dotenv
pyyaml
```

Install everything using:

```
bash

pip install -r requirements.txt
```

How to Use It

1. Add your credentials to `.env`
2. Customize `config.yaml` with table/column names
3. Run:

```
bash

python refit_pipeline.py
```

Bonus Tip for Security

- Add `.env` to `.gitignore` so you don't commit secrets.
- Store `.env` in your cloud vault (AWS Secrets Manager, Azure Key Vault) for production.

Would you like to extend this to automatically **save the DataFrame to a .parquet or .csv**, or move on to the **next task in your refitment pipeline (e.g., preprocessing or model training)**?



I want to automate the below:

- * Once the under-sampled data from population is fed,
 - * It should do the grid search for finding best hyper parameters.
 - * Do the X-y split, train-test split, use best hyper parameters from grid search.
 - * Generate AUC-ROC score, capture rate curve & cumulative capture rate curve.

Great! Here's a general plan and modular Python template to automate your entire pipeline:

✓ Goal

Once under-sampled data is available:

1. **GridSearchCV** to find best hyperparameters
2. **Train/Test split** and model training using best params
3. **Evaluate**: AUC-ROC, Capture Rate Curve, and Cumulative Capture Rate Curve

Generalized Code Template

python

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt
import numpy as np
import xgboost as xgb # Example model; replaceable with any

def capture_rate_curve(y_true, y_probs, k_percent=10):
    # Sort by predicted probabilities
    data = pd.DataFrame({'y_true': y_true, 'y_prob': y_probs})
    data = data.sort_values('y_prob', ascending=False)
    cutoff = int(len(data) * (k_percent / 100))
    capture_rate = data.iloc[:cutoff]['y_true'].sum() / data['y_true'].sum()
    return capture_rate

def cumulative_capture_rate_curve(y_true, y_probs, intervals=[10, 20, 30, 50, 100]):
    rates = {}
    for k in intervals:
        rates[f'{k}%'] = capture_rate_curve(y_true, y_probs, k)
    return rates

def run_pipeline(data, target_column, model, param_grid):
    X = data.drop(columns=[target_column])
    y = data[target_column]

    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.3,
random_state=42)

    grid_search = GridSearchCV(model, param_grid, scoring='roc_auc', cv=5, n_jobs=-1)
    grid_search.fit(X_train, y_train)
    best_model = grid_search.best_estimator_

    y_probs = best_model.predict_proba(X_test)[:, 1]
    auc = roc_auc_score(y_test, y_probs)

    print(f"AUC-ROC Score: {auc:.4f}")

    fpr, tpr, _ = roc_curve(y_test, y_probs)
    plt.plot(fpr, tpr, label='AUC-ROC Curve')
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.title("ROC Curve")
    plt.legend()
    plt.grid()
    plt.show()

    capture_rate = capture_rate_curve(y_test, y_probs)
    cumulative_capture = cumulative_capture_rate_curve(y_test, y_probs)

    print(f"Capture Rate @ 10%: {capture_rate:.4f}")
    print("Cumulative Capture Rate:")
    for k, v in cumulative_capture.items():
        print(f"{k}: {v:.4f}")

    return best_model
```

How to Use

```
python

# Example:
data = pd.read_csv("undersampled_population.csv")
target_column = 'target'

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5],
    'learning_rate': [0.01, 0.1]
}

model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')
best_model = run_pipeline(data, target_column, model, param_grid)
```

💡 Tips for Full Automation

- ✓ Accept file path as input or CLI argument
- ✓ Save best model using **joblib**
- ✓ Store AUC/Capture outputs to CSV
- ✓ Log progress using **logging** module

Would you like this wrapped as a script with **.env** configuration for paths and model settings as well?



I have audience shift matrix in percaentage.
The rows & colums represent deciles from 1 to 10 for old & new model respectively.
This is used to check how customers have moved between deciles for old & new model
How to read it. Explain me with an example

Great question!

An **Audience Shift Matrix** (sometimes called a *Decile Migration Matrix*) shows how customers or records shift across **deciles** (ranked segments like top 10%, next 10%, etc.) from an **old model** to a **new model**. Each cell contains a percentage indicating how many customers moved from one decile in the old model to a decile in the new model.

📄 How to Read It

Let's say your matrix looks like this:

Old ↓ / New →	1	2	3	...	10
1	25%	20%	15%	...	2%
2	10%	20%	25%	...	1%
...

Old ↓ / New →	1	2	3	...	10
10	1%	2%	3%	...	40%

- **Rows:** Deciles in the **old model**
- **Columns:** Deciles in the **new model**
- **Cell (i, j):** % of customers that were in decile **i** in the **old model** and are now in decile **j** in the **new model**

✓ Example Interpretation

Let's focus on just one row:

Row 1 (Old Decile 1 → Top 10% customers in old model):

New Decile	1	2	3	4	...	10
% Shift	25%	20%	15%	10%	...	2%

● This means:

- **25%** of customers who were in the **top decile (1st)** in the old model are still in the top decile in the new model (good stability).
- **20%** moved down to the 2nd decile in the new model.
- **2%** dropped all the way down to decile 10 (the worst 10%).

This gives you a sense of **model stability or churn** — whether your top customers in the old model are still ranked high in the new model.

🎯 What to Look For

1. Diagonal Dominance

- The **diagonal cells** (top-left to bottom-right) show how stable your model is.
- Higher diagonal values = more consistency between old and new models.

2. Upgrades/Downgrades

- Values **above the diagonal** = customers ranked **better** in the new model (upgrades).
- Values **below the diagonal** = customers ranked **worse** (downgrades).

3. Top Decile Analysis

- How many of your **top decile (Decile 1)** customers in the old model stayed in the top few deciles?
 - E.g., if 60–70% remain in top 3 deciles, model retains focus.
- If many dropped to lower deciles, the model is **shifting its targeting**.

Use Cases

- **Model comparison:** Is the new model capturing the same audience?
- **Business risk:** Are high-value customers being lost in the new model?
- **Marketing:** Should we remarket or resegment shifted customers?

Would you like a Python code to visualize this matrix as a heatmap for better understanding?



Explain this in simple understandable manner with an example:

What to Look For

1. Diagonal Dominance

The diagonal cells (top-left to bottom-right) show how stable your model is.

Higher diagonal values = more consistency between old and new models.

2. Upgrades/Downgrades

Values above the diagonal = customers ranked better in the new model (upgrades).

Values below the diagonal = customers ranked worse (downgrades).

3. Top Decile Analysis

How many of your top decile (Decile 1) customers in the old model stayed in the top few deciles?

E.g., if 60–70% remain in top 3 deciles, model retains focus.

If many dropped to lower deciles, the model is shifting its targeting.

Sure! Let me explain the **Audience Shift Matrix** in a very simple and visual way with an **example** so it becomes easy to understand.

What is an Audience Shift Matrix?

It's a **10x10 table** where:

- **Rows = Deciles of the Old Model (Decile 1 to Decile 10)**
- **Columns = Deciles of the New Model (Decile 1 to Decile 10)**
- Each cell contains the **% of customers** who moved from an old decile to a new decile

This helps you **compare ranking shifts** between the old model and the new one.

Example Matrix (simplified)

	New D1	New D2	New D3	...	New D10
Old D1	40%	25%	10%	...	0%
Old D2	10%	30%	20%		
Old D3	5%	10%	25%		
...					
Old D10	0%	0%	5%		60%

✓ How to Read It

1 Diagonal Dominance (Old D1 → New D1, Old D2 → New D2, etc.)

These diagonal cells show the % of customers **who stayed in the same decile**.

- Example: Old D1 → New D1 is 40%
 - ✓ This means 40% of the top customers according to the old model **remain** at the top in the new model.
- A **strong diagonal** means the **new model is similar** to the old one (more stable).
- A **weak diagonal** (low percentages) means the model is **changing rankings** a lot.

2 Upgrades / Downgrades

- **Above the diagonal** = Upgrades (customers moved to **better deciles** in the new model)
 - E.g., Old D2 → New D1 = 10%
 - ▲ These customers were Decile 2 before, now they are Decile 1 — improved score.
- **Below the diagonal** = Downgrades (moved to **worse deciles**)
 - E.g., Old D1 → New D3 = 10%
 - ▼ Top customers now scored lower — possible issue.

3 Top Decile Analysis

Check how your most valuable customers (Old D1) moved:

- From the example:
 - Old D1 → New D1 = 40%
 - Old D1 → New D2 = 25%
 - Old D1 → New D3 = 10%
 - Total = **75% stayed within Top 3 deciles**
 - ✓ This is **good** — your new model **still targets key customers** well.

But if many had dropped to New D7–D10, it means your model's **focus changed**, possibly unintentionally.

Summary

What to Check	What It Tells You
Diagonal values	Stability of customer rankings
Above diagonal values	Customers scored better in new model (Upgrades)
Below diagonal values	Customers scored worse in new model (Downgrades)
Top decile retention	Whether you still target your most important customers

Would you like a **Python heatmap** example for visualization?



Show me the code to calculate top 3 decile analysis, once I have the audience shift matrix

Certainly! To calculate the **Top 3 Decile Analysis** from an **audience shift matrix**, we'll assume:

- The matrix is a 10x10 pandas DataFrame.
- Rows = old model deciles (1 to 10).
- Columns = new model deciles (1 to 10).
- Each cell contains percentage of customers (e.g., 5.4 means 5.4% moved from old Decile X to new Decile Y).

Goal:

Find **what % of customers from old Decile 1** are in the **top 3 new deciles (1, 2, or 3)**.

Sample Code:

```
python

import pandas as pd

# Example audience shift matrix (replace with your actual DataFrame)
# Rows: Old decile (1 to 10), Columns: New decile (1 to 10)
audience_shift_matrix = pd.DataFrame({
    1: [30, 10, 5, 2, 1, 1, 1, 1, 1, 1],
    2: [20, 25, 10, 3, 2, 1, 1, 1, 1, 1],
    3: [10, 20, 20, 5, 2, 2, 1, 1, 1, 1],
    4: [5]*10,
    5: [5]*10,
    6: [5]*10,
    7: [5]*10,
    8: [5]*10,
    9: [5]*10,
    10: [5]*10,
}, index=[1,2,3,4,5,6,7,8,9,10])

# Calculate Top 3 Decile Analysis for old Decile 1
def top_3_decile_analysis(matrix):
    decile_1_row = matrix.loc[1] # Old model Decile 1
    top_3_sum = decile_1_row[1] + decile_1_row[2] + decile_1_row[3] # New model Deciles 1, 2, 3
    total_sum = decile_1_row.sum()
```

```
percentage_in_top3 = (top_3_sum / total_sum) * 100
return round(percentage_in_top3, 2)

result = top_3_decile_analysis(audience_shift_matrix)
print(f"Percentage of Old Decile 1 customers staying in Top 3 New Deciles: {result}%")
```

Output Example:

sql

Percentage of Old Decile 1 customers staying in Top 3 New Deciles: 65.0%

Let me know if:

- Your matrix uses 0-based or 1-based indexing.
- You want analysis for multiple deciles.
- You want a heatmap or visual to go with this.



Suppose i have audience shift matrix not in percentage but the count of cutomers. Show me the code to check top 3 decile

Great! If your **audience shift matrix is in counts (not percentages)** — meaning each cell shows the **number of customers** who moved from an **old decile (row)** to a **new decile (column)** — then the logic stays the same, except we **don't divide by 100**.

Objective:

Calculate what % of customers from Old Decile 1 ended up in New Deciles 1, 2, or 3.

Code Example (for count-based matrix):

python

```
import pandas as pd

# Sample audience shift matrix in counts (rows: old deciles, columns: new deciles)
# This is just sample data. Replace it with your real DataFrame.
audience_shift_matrix = pd.DataFrame({
    1: [300, 100, 50, 20, 10, 10, 5, 5, 0, 0],
    2: [200, 250, 100, 30, 10, 5, 5, 0, 0, 0],
    3: [100, 200, 200, 50, 20, 10, 5, 0, 0, 0],
    4: [20]*10,
    5: [15]*10,
    6: [10]*10,
    7: [10]*10,
    8: [5]*10,
    9: [5]*10,
    10: [2]*10,
}, index=[1,2,3,4,5,6,7,8,9,10])
```

```
# Function to calculate top 3 decile analysis from Old Decile 1
def top_3_decile_analysis(matrix):
    decile_1_row = matrix.loc[1] # Customers from Old Decile 1
    top_3_sum = decile_1_row[1] + decile_1_row[2] + decile_1_row[3] # New deciles 1, 2, 3
    total_sum = decile_1_row.sum()
    percentage = (top_3_sum / total_sum) * 100
    return round(percentage, 2)

result = top_3_decile_analysis(audience_shift_matrix)
print(f"Percentage of Old Decile 1 customers in New Deciles 1-3: {result}%")
```

Example Output:

```
sql

Percentage of Old Decile 1 customers in New Deciles 1-3: 88.24%
```

Optional: Generalize for Any Decile

If you want to check this for **any old decile**, pass the decile number as an argument:

```
python

def top_n_new_deciles_analysis(matrix, old_decile=1, top_n=3):
    decile_row = matrix.loc[old_decile]
    top_n_sum = decile_row.loc[1:top_n].sum() # New Deciles 1 to top_n
    total_sum = decile_row.sum()
    percentage = (top_n_sum / total_sum) * 100
    return round(percentage, 2)
```

```
python

# Usage
result = top_n_new_deciles_analysis(audience_shift_matrix, old_decile=1, top_n=3)
print(f"Old Decile 1 in Top 3 New Deciles: {result}%")
```

Let me know if you'd like to visualize this as a heatmap or stack bar chart.



suppose I have unouque cutomer id for each customer. Show me the code to get top 3 decile using customer id

If you have **individual customer-level data** with:

- Unique `customer_id`
- Their **old decile** (`old_decile`)
- Their **new decile** (`new_decile`)

Then we can calculate **Top 3 New Decile retention** for **Old Decile 1 customers** like this:

Step-by-step Python Code:

```
python

import pandas as pd

# Sample customer-level data
data = {
    'customer_id': range(1, 101),
    'old_decile': [1]*25 + [2]*25 + [3]*25 + [4]*25,
    'new_decile': [1, 2, 2, 1, 3, 4, 5, 1, 2, 3, 4, 5, 6, 1, 2, 3, 7, 8, 9, 10, 3, 2, 1, 4, 5]*4
}

df = pd.DataFrame(data)

# Function to calculate Top N new decile retention for a given old decile
def top_n_decile_retention(df, old_decile=1, top_n=3):
    # Filter customers from the selected old decile
    old_decile_df = df[df['old_decile'] == old_decile]
    total_customers = len(old_decile_df)

    # Count how many went to top N new deciles (i.e., new_decile in [1, 2, 3])
    retained_customers = old_decile_df[old_decile_df['new_decile'].isin(range(1, top_n+1))]
    retained_count = len(retained_customers)

    percentage = (retained_count / total_customers) * 100
    return round(percentage, 2), retained_customers

# Example: Top 3 new decile % for customers in old decile=1
percent, top3_customers = top_n_decile_retention(df, old_decile=1, top_n=3)
print(f"Top 3 Decile Retention for Old Decile 1: {percent}%")
print("\nTop 3 Decile Customers:\n", top3_customers.head())
```

Output Example:

```
yaml

Top 3 Decile Retention for Old Decile 1: 68.0%

Top 3 Decile Customers:
  customer_id  old_decile  new_decile
0           1           1           1
1           2           1           2
2           3           1           2
3           4           1           1
4           5           1           3
```

Notes:

- You can easily modify the `old_decile=1` or `top_n=3` to test other cases.
- If you're working with a **larger real dataset**, this approach remains scalable.
- You can even visualize transitions using a **Sankey diagram** or a **pivot table heatmap**.

Let me know if you want to see **transition matrices**, **charts**, or a **decile shift summary by all deciles**.



how to upload files to your git hub account

To upload files to your GitHub account, follow the steps below based on whether you're using the **GitHub website** or **Git command line**.

✓ Method 1: Upload Files via GitHub Website (Simplest Way)

Step-by-step:

1. **Login to GitHub** → <https://github.com>
2. Go to your repository or create a new one:
 - Click the "+" → **"New repository"**
 - Set the name, visibility (public/private), and click **Create repository**
3. Inside your repo, click the **"Add file"** → **"Upload files"**
4. Drag and drop files or click **"choose your files"**
5. Add a commit message like:

```
sql

Add project files
```

6. Click **"Commit changes"** at the bottom.

✓ Your files are now uploaded to GitHub.

✓ Method 2: Upload Files Using Git Command Line (For local folders)

🔧 Requirements:

- Git installed: <https://git-scm.com/downloads>
- GitHub account with a repo created

Step-by-step:

1. **Open terminal / command prompt**
2. Navigate to your project folder:

```
bash

cd path/to/your/project
```

3. **Initialize Git (if not already):**

```
bash

git init
```


4. Connect to your GitHub repo:

```
bash

git remote add origin https://github.com/your-username/your-repo-name.git
```

5. Add files:

```
bash

git add .
```

6. Commit the files:

```
bash

git commit -m "Initial commit with project files"
```

7. Push to GitHub (first time):

```
bash

git branch -M main
git push -u origin main
```

Tips:

- Replace **your-username** and **your-repo-name** with your GitHub details.
- Use **git status** anytime to check what's staged/unstaged.
- You can also use **GitHub Desktop** app (GUI) if you prefer a visual interface:
<https://desktop.github.com>

Would you like me to generate a **sample README.md** or **.gitignore** file for your project as well?