## **Preprocessing Strategy for Multiple Prediction Tasks**

**The Challenge:**

Our project involves two main prediction tasks using largely the same dataset:

1. Predicting **First Year Persistence** (Classification: Yes/No)
2. Predicting **Second Term GPA** (Regression: Numerical Value)

While most input features (like High School scores, demographics, etc.) are relevant for both tasks, the optimal set might differ slightly. Specifically, we need to consider:

* Should Second Term Gpa be used as an input when predicting First Year Persistence? (Likely no, to avoid using future information).
* Should First Year Persistence be used as an input when predicting Second Term Gpa? (Possibly yes, as persistence status might influence future grades).

We need a consistent and correct way to prepare the input features (X) for each task before feeding them into our models.

**Two Approaches Considered:**

Here are two ways we could structure our preprocessing code to handle this:

***Approach 1: Feature Selection Before Preprocessing***

1. **How it Works:**
   * Modify the data splitting step (train\_test\_split).
   * When creating the features (X) for *either* task, *always* drop *both* potential target columns (First Year Persistence AND Second Term Gpa).
   * Use a single, fixed preprocessing pipeline definition (handling imputation, scaling, encoding) that always receives this consistent set of input columns.
2. **Pros:**
   * Keeps the preprocessing pipeline definition itself very simple and stable.
3. **Cons:**
   * Might unnecessarily remove a potentially useful feature. For example, it prevents using First Year Persistence as an input for predicting Second Term Gpa, even if it might be predictive.
   * The logic for feature exclusion is handled during the data splitting phase.

***Approach 2: Feature Selection Within Preprocessing (Our Chosen Approach)***

1. **How it Works:**
   * Create a flexible preprocessing pipeline structure (e.g., using a function that builds a ColumnTransformer).
   * Pass a flag (e.g., predict\_persistence=True or False) to this function.
   * *Inside* the function, dynamically adjust the lists of columns that get specific treatments (imputation, scaling, encoding) based on the flag.
   * For example, Second Term Gpa is only included in the list of numerical features to be scaled if predict\_persistence is False. First Year Persistence could potentially be added to the list of categorical features to be encoded if predict\_persistence is False.
2. **Pros:**
   * **Flexibility:** Allows each task to use the most appropriate set of input features without prematurely dropping potentially useful columns.
   * **Maximizes Data Use:** We don't discard First Year Persistence when predicting GPA, or Second Term Gpa when defining features (though we exclude it from the *final list* for the persistence task).
   * **Bundled Logic:** Keeps the feature selection logic directly tied to the preprocessing steps for that specific task.
3. **Cons:**
   * Makes the preprocessing definition function slightly more complex internally.

**Why We Chose Approach 2:**

We selected **Approach 2** because it offers greater flexibility and avoids potentially discarding valuable information.

* It allows us to easily tailor the input feature set precisely for each prediction task.
* Crucially, it lets us keep columns like First Year Persistence available as a potential input feature when predicting Second Term Gpa, which might improve model accuracy.
* While Second Term Gpa is correctly excluded *by the preprocessor logic* when predicting persistence (avoiding data leakage), the column itself isn't dropped unnecessarily early in the workflow.
* This approach ensures each model receives the most relevant set of inputs based on the specific prediction goal, maximizing the potential of our data.

This method provides a robust and adaptable way to handle preprocessing for our distinct prediction tasks within a single, organized codebase.