Code Explanation

1. Target and Type of regression

The dataset file path and metadata such as the target variable and prediction type are extracted from **`algoparams_from_ui.json`** file.

- **`df`** holds the full dataset.
- **`target_variable`** defines the column to be predicted.
- **`prediction_type`** specifies whether the task is **regression** or **classification**.

2. `Custom Feature Handling`

I built a **custom transformer** called `Custom_Feature_Handling` to clean and prepare both **numerical** and **text** features — all based on instructions from a JSON configuration.

Handling Numerical Features

For each numerical column in the dataset, we check:

- **Is the column selected for modelling?**
- **Are there missing values?** If yes, we fill them using:
  - The **average (mean)** of the column, or
  - A **custom value** defined in the config file.

This makes sure that model doesn't break because of missing values and keeps things consistent.

Handling Text Features

Text columns are handled slightly different:

- If there's any **missing text**, fill it with `"missing_text"` to avoid errors.
- If the configuration says to use `"Tokenize and hash"`:
  - We apply a **hashing trick** to turn the text into a fixed number of numerical columns.

This step ensures that text data can be used in the model, without adding unnecessary complexity.

3. Custom `Custom_Feature_Reduction`

We created a custom class called `Custom_Feature_Reduction` that reduces dataset's size by keeping only the most relevant columns.

And the best part?

It adapts its strategy based on what's defined in the configuration JSON.

Modes Strategies:

Depending on what's set in the config, the transformer can use **one of four strategies**:

1. Correlation with Target

- Measures how strongly each feature relates to the target variable.
- Keeps the **top K** features that show the strongest correlation.

2. Tree-Based Importance

- Trains a **Random Forest** to find which features are most important in making predictions.
- Ranks features by their importance scores from the forest.
- Selects the **top K** features that contribute most to accuracy.

3. Principal Component Analysis (PCA)

- PCA compresses many correlated features into a few **uncorrelated components**.
- It's like blending information from several features into fewer new ones.
- Keeps only the top **K components** that explain the most variance.

4. No Reduction

- If the config says `"No Reduction"`, the transformer **keeps all columns** (or a fixed number of them).

4. Hyper parameter tuning (GridSearchCV)

Most important step is **choosing the best model** to make predictions. But different models have different strengths — and they need the **right hyperparameters** to perform well.

Selects, tunes and evaluates models using GridSearchCV

Step 1: `MODEL_for_tuning()` Mapping of models to hyperparameters

If I want to use model X, then these are the hyperparameters I should try tuning and here's how to extract those tuning ranges from the config data.

`MODEL_for_tuning` declares what to tune and how to fetch the range definition from the input config.

The following regression and classification models are tuned:

- **Regression**:
  - `LinearRegression`, `Ridge`, `Lasso`, `ElasticNet`
  - `RandomForestRegressor`, `DecisionTreeRegressor`
  - `GradientBoostingRegressor`, `ExtraTreesRegressor`, `XGBRegressor`
  - `MLPRegressor` (neural network)
- **Classification**:
  - `LogisticRegression`, `RandomForestClassifier`, `DecisionTreeClassifier`
  - `GradientBoostingClassifier`, `ExtraTreesClassifier`, `XGBClassifier`
  - `SVM`, `KNeighborsClassifier`, `MLPClassifier`, `SGDClassifier`

Step 2: `building_pmt_grid()` Creates the actual parameter grid

This function **interprets the hyperparameter definitions** from `MODEL_for_tuning` and **builds the final parameter grid** that will be passed to `GridSearchCV`.

`building_pmt_grid` **converts abstract definitions into real values** to form a concrete `pmt_grid`

Step 3: Custom `ModelSelection_With_GridSearch`

This project automates that process through a custom scikit-learn-compatible class called `ModelSelection_With_GridSearch`.

It is designed to **choose, configure, and fine-tune** machine learning models using parameters provided in a JSON configuration file.

What This Component Does

The `ModelSelection_With_GridSearch` class acts as the **final estimator** in the pipeline. It performs the following steps programmatically:

1. **Parses the JSON configuration** to determine:
    - Whether the task is **regression** or **classification**
    - Which models are selected (`"is_selected": true`)
    - What parameter ranges should be explored during hyperparameter tuning
2. **Builds model instances** using scikit-learn classes like `RandomForestRegressor`, `LogisticRegression`, `SVC`, `MLPClassifier`, etc., depending on the type of task.
3. **Creates parameter grids** from the JSON file using keys like `range(min, max)` or `range_float(min, max, step)` that are parsed into actual hyperparameter value ranges.
4. **Runs `GridSearchCV`** on each selected model:
    - Uses **5-fold cross-validation** (`K-fold`) to evaluate each hyperparameter combination
    - Identifies the best-performing combination for each model
    - Stores the model and its best parameters for later use
5. **Returns a list of best estimators**, one for each selected model, sorted by their validation performance.

Each of these models can be **turned on/off** and **customized** just by changing values in the JSON file — making the system highly modular and reusable.

5. Execute pipeline

The pipeline is constructed with three core components:

```
Custom_Feature_Handling

Custom_Feature_Reduction

ModelSelection_With_GridSearch
```

Output

At the end of this step:

- We get the **best version of each selected model**, tuned and validated.
- These models are stored in `self.selected_models_` as a list of tuples: `(model_name, best_estimator_)`.

This design makes it **extremely reusable and configurable**. Just change the JSON file, and you can handle a completely new dataset, apply different reduction strategies, or try different model/hyperparameter combinations — **without changing the code**.