# Classification Algorithm Development Framework – Built using OPAIR

#### **O – Objective**

To help beginners in data science build, interpret, and evaluate classification models through a structured, beginner-friendly coding process that is designed to be implemented in **Jupyter Notebook** using clear cell-by-cell explanations and modular code blocks.

#### **P – Principles**

|  |  |
| --- | --- |
| **Principle** | **Description** |
| Target-Centric Design | Start with a clear definition of the classification goal. |
| Notebook-Oriented Thinking | Use markdown cells to explain logic and code cells to implement steps. |
| Clean Code for Beginners | Keep syntax simple, avoid overly abstract code, and comment generously. |
| Visual Interpretability | Include charts and outputs to explain model behavior (e.g., confusion matrix, ROC curves). |
| Structured Learning Flow | Each notebook section reflects a distinct stage: cleaning → prep → modeling → evaluation. |

#### **A – Actions Enabled by This Framework**

|  |  |
| --- | --- |
| **Capability** | **Description** |
| Create beginner-ready classification notebooks | Easy-to-follow Jupyter Notebooks that combine code and concept. |
| Learn-by-doing model workflows | Step-by-step implementation with visual feedback and minimal abstraction. |
| Simplify model training logic | Use scikit-learn pipelines or explicit sequential steps that make code readable. |
| Embed visuals for learning | Confusion matrices, feature importance charts, ROC curves, etc. |
| Adapt for classroom or self-study | Ideal for workshops, tutorials, and portfolio-building. |

#### **I – Inputs Needed to Activate the Framework**

|  |  |
| --- | --- |
| **Input Type** | **Description** |
| Jupyter Notebook | Preferred environment for code and commentary-based execution. |
| Labeled Dataset | Dataset (CSV, Excel, or dataframe) with a target variable. |
| Python Libraries | pandas, numpy, scikit-learn, matplotlib, seaborn |
| Learning Objective | Whether the user wants to understand cleaning, modeling, or evaluation more deeply. |
| Beginner Level | Assumes minimal exposure to ML—no need for advanced object-oriented syntax. |

#### **R – Real-World Examples**

|  |  |
| --- | --- |
| **Use Case** | **Jupyter Notebook Learning Path** |
| Credit Approval | Step-by-step logistic regression with clear labeling and evaluation visuals. |
| Churn Prediction | Feature selection using correlation matrix and tree-based models. |
| Email Spam Detection | Simple text vectorization followed by naive Bayes or SVM. |
| Medical Diagnosis Support | Explainable models with visuals on false positives and false negatives. |

### **Core Steps of the Framework (Notebook-Compatible)**

|  |  |
| --- | --- |
| **Notebook Section** | **What to Include** |
| **1. Target Feature Identification** | Markdown: Define the classification goal Code: Display class distribution using value\_counts() and plots |
| **2. Data Cleaning** | Code: - Drop ID columns - df.isnull().sum() with imputation logic  - Encode categorical vars (pd.get\_dummies or LabelEncoder)  - Handle outliers with IQR or zscore() |
| **3. Data Preprocessing** | Code: - Scale numerical features using StandardScaler or MinMaxScaler - Convert date/time if needed  - Check class imbalance |
| **4. Variable Selection** | Code: - Correlation matrix + heatmap  - Univariate tests (e.g., SelectKBest)  - Tree-based feature importance plots |
| **5. Train-Test Split** | Code: Use train\_test\_split from sklearn.model\_selection with stratification Visuals: Show sample class balance in both splits |
| **6. Model Building** | Code: Start with LogisticRegression, then optionally try RandomForestClassifier, XGBoost Use pipeline only after basic model is understood |
| **7. Model Evaluation** | Code + Markdown: - Use classification\_report, confusion\_matrix, roc\_auc\_score  - Plot confusion matrix, ROC curve, precision-recall curve  - Interpret results and next steps |

### **Additional Notes for Beginner-Friendly Jupyter Notebooks**

* Use **section headers** like ### Step 1: Data Cleaning to guide navigation.
* Add **markdown explanations** before each code block.
* Include **error handling tips** (e.g., what to do if LabelEncoder throws an error).
* Ensure **consistent variable naming** and avoid overwriting variables unless intentional.
* Save model outputs using joblib or pickle for deployment previews.