**FAQ**

What is Cross-Validation?

* Cross-validation (CV) is a model evaluation technique that splits the dataset into multiple parts (folds) to test how well a model generalizes to unseen data.
* Instead of training once on 70% and testing on 30%, cross-validation trains multiple times on different splits of the data, then averages the performance.

Why Cross-Validation?

* Prevents overfitting (model working well only on training data).
* Gives a more reliable estimate of model performance.
* Uses data more efficiently (all samples are used for training and testing).

Stratified K-Fold CV

* Same as K-Fold but maintains class distribution in each fold.
* Useful for classification with imbalanced data.

What is Normalization?

* Normalization is a preprocessing technique that scales numeric features so they are on a similar range (usually 0–1 or -1–1).
* The goal is to make sure no single feature dominates others just because it has larger values.
* For example:
  + Age ranges 20–80
  + Income ranges 20,000–200,000
  + Without normalization, models might give too much importance to income just because the numbers are bigger.

**How PyCaret Handles Categorical Features**

When preprocessing, PyCaret applies encoding:

1. **Imputation** (categorical\_imputation)
   * Fills missing values in categorical columns (mode by default).
2. **One-Hot Encoding** (default True)
   * Example: city = {Delhi, Mumbai, Chennai} → city\_Delhi, city\_Mumbai, city\_Chennai.
3. **Ordinal Encoding** (if specified with ordinal\_features)
   * Example: Education level → {High School < Bachelor < Master < PhD}.
4. **High Cardinality Encoding** (high\_cardinality\_features)
   * For features with many unique categories, you can use methods like frequency or clustering.
5. **Rare Categories Handling** (rare\_to\_value)
   * Rare categories can be grouped into "rare".

# Preprocessing in PyCaret

When you run setup(data, target=...), PyCaret applies a pipeline of preprocessing steps.

## 1. ****Missing Value Handling****

* numeric\_imputation → Fills missing numeric values (mean, median, mode).
* categorical\_imputation → Fills missing categorical values (mode or constant).
* imputation\_type → simple (default) or iterative (advanced imputation).

## 2. ****Encoding Categorical Features****

* one\_hot\_encoding=True → Converts categorical variables into dummy variables.
* ordinal\_features → Define order for ordinal variables (e.g., Education).
* high\_cardinality\_features → Columns with many categories (special encoding).
* rare\_to\_value=True → Groups rare categories into "rare".

## 3. ****Scaling & Normalization****

* normalize=True → Scales numeric features.
* normalize\_method= →
  + "zscore" (standard scaling: mean=0, std=1)
  + "minmax" (scales 0–1)
  + "maxabs" (scales by max absolute value)
  + "robust" (robust to outliers).

## 4. ****Transformation****

* power\_transform=True → Normalizes distribution (Box-Cox or Yeo-Johnson).
* transform\_target=True → Applies transformation to target (regression only).

## 5. ****Feature Engineering****

* polynomial\_features=True → Adds polynomial terms (x², x³).
* polynomial\_degree=2 → Degree of polynomial expansion.
* feature\_interaction=True → Creates interaction terms (x1 × x2).
* feature\_ratio=True → Creates ratio features (x1 / x2).

## 6. ****Feature Selection****

* feature\_selection=True → Selects most important features.
* feature\_selection\_threshold → Importance threshold.
* remove\_multicollinearity=True → Drops correlated features.
* multicollinearity\_threshold=0.9 → Correlation cutoff.

## 7. ****Outlier Handling****

* remove\_outliers=True → Removes outliers.
* outliers\_threshold=0.05 → % of samples allowed to be outliers.

## 8. ****Balancing Target Classes****

* fix\_imbalance=True → Applies SMOTE (or other methods).
* fix\_imbalance\_method= → Specify method.

## 9. ****Dimensionality Reduction****

* pca=True → Apply PCA.
* pca\_method= → "linear", "kernel", "incremental".
* pca\_components= → Number of components.

**What is imputation\_type?**

* It’s a parameter in setup() that controls **how missing values are imputed (filled)**.
* PyCaret supports two main strategies:

1. **simple (default)**
   * Fills missing values using simple statistics.
   * Works with:
     + numeric\_imputation → mean, median, or mode
     + categorical\_imputation → mode or constant

✅ Fast and commonly used.

1. **iterative**
   * Uses **Iterative Imputer** (from scikit-learn).
   * Predicts missing values by modeling each feature with missing values as a function of other features.
   * More accurate but slower than simple imputation.

✅ Good for datasets with many missing values or complex relationships.

Difference between USI and Session id

* **session\_id**: The random seed you provide to control reproducibility (e.g., train-test splits, shuffling).
* **USI**: An internal unique identifier automatically generated by PyCaret for your experiment.

PyCaret setup Options

|  |  |  |
| --- | --- | --- |
| SN | Description | Meaning |
| 1 | Session id | It’s basically a **random seed**.  It ensures that all the random processes inside PyCaret (like train-test split, cross-validation folds, shuffling, etc.) are **reproducible**.  If you use the same session\_id, you’ll get the **same results every time** you run the code.  If you don’t set it, PyCaret will pick a random number, and your results may differ on each run. |
| 2 | Target | The column name of the target variable. |
| 3 | Target type | the target type refers to the kind of variable you are trying to predict |
| 4 | Target mapping | target\_mapping is an optional parameter in setup() (mainly for classification).  It allows you to manually map categorical target labels to numeric values. |
| 5 | Original data shape |  |
| 6 | Transformed data shape |  |
| 7 | Transformed train set shape |  |
| 8 | Transformed test set shape |  |
| 9 | Numeric features | numeric features are all columns in your dataset that contain numbers (integers or floats), excluding the target column. |
| 10 | Categorical features | By default, PyCaret detects features with data type object or category in pandas as categorical.  If you have numeric-looking codes stored as numbers (e.g., 1=Male, 2=Female), PyCaret may treat them as numeric unless you force them as categorical. |
| 11 | Preprocess | True . It performs pre processing. |
| 12 | Imputation type | It’s a parameter in setup() that controls how missing values are imputed (filled).  **simple (default)**  numeric\_imputation → mean, median, or mode  categorical\_imputation → mode or constant |
| 13 | Numeric imputation | numeric\_imputation → mean, median, or mode |
| 14 | Categorical imputation | categorical\_imputation → mode or constant |
| 15 | Maximum one-hot encoding | This parameter controls the maximum number of categories allowed for one-hot encoding.  Default = 25  If a categorical feature has more unique levels than this number, PyCaret does not one-hot encode it. |
| 16 | Encoding method | Specifies the type of encoding. |
| 17 | Normalize | normalize is a parameter in setup() that tells PyCaret whether to scale/standardize numeric features. |
| 18 | Normalize method | Specifies the method utilized |
| 19 | Fold Generator | A fold refers to a split of your dataset used in cross-validation (CV).  PyCaret uses k-fold cross-validation by default, which means:  The dataset is split into k equal parts (folds).  Each fold is used once as a validation set, while the rest are used for training.  Results are averaged over all folds. |
| 20 | Fold Number | Specifies the number. |
| 21 | CPU Jobs | Number of CPU cores to use (-1 = all). |
| 22 | Use GPU | Use GPU if available (True/False). |
| 23 | Log Experiment | Log results (MLflow integration). |
| 24 | Experiment Name | Name of MLflow experiment. |
| 25 | USI (Unique Session Identifier) | Every time you run setup(), PyCaret assigns a USI (Unique Session Identifier).  It’s basically a hash key that uniquely represents that experiment’s configuration.  It ensures reproducibility and lets PyCaret internally track transformations and pipelines. |

List of Plot Types - Classfication

|  |  |  |
| --- | --- | --- |
| SN | Plot Type | Meaning |
| 1 | Pipeline Plot | creates a pipeline of transformations and modeling steps. |
| 2 | Hyperparameters | A hyperparameter in machine learning is a parameter set before training a model—it is not learned from the data. Instead, it controls the learning process or the structure of the model. |
| 3 | AUC | In PyCaret, AUC stands for Area Under the ROC Curve. It is a performance metric for binary classification models that measures how well the model distinguishes between the two classes.  **Range:** 0 to 1   * **1** → perfect classification * **0.5** → model is no better than random guessing * **<0.5** → model is performing worse than random (rare) |
| 4 | Confusion Matrix | A confusion matrix is a table used to evaluate the performance of a classification model by showing the counts of true vs predicted labels. It helps you understand where your model is getting things right or wrong. |
| 5 | Threshold | In PyCaret, the threshold in a classification plot refers to the probability cutoff used to decide class labels from predicted probabilities. |
| 6 | Precision and Recall | In **PyCaret**, the **Precision-Recall plot** is a tool to visualize the trade-off between **precision** and **recall** for a classification model as the **decision threshold** changes. It’s especially useful for **imbalanced datasets**, where accuracy alone can be misleading. |
| 7 | Prediction Error | In PyCaret, a Prediction Error plot is mainly used for regression tasks to visualize how well the model’s predictions match the actual values. It shows the relationship between predicted values and actual values and highlights prediction errors (residuals). |
| 8 | classification report plot | In PyCaret, the Class Report plot (or classification report plot) is a visual representation of the classification metrics—precision, recall, F1-score—for each class. |
| 9 | Feature Selection | In PyCaret, the Feature Selection Plot helps you identify which features (columns) are most important for predicting the target variable. This is especially useful for reducing dimensionality or understanding your model. |
| 10 | Learning Curve | In **PyCaret**, the **learning curve** is one of the diagnostic plots you can generate to check how well a model is learning as the training size increases. It helps you see **bias vs. variance trade-off** and detect **underfitting / overfitting**. |
| 11 | Mainfold Learning | In **PyCaret**, manifold learning is available as one of the visualization plots to **see your dataset in reduced dimensions (2D)** using **t-SNE** or **UMAP** (sometimes Isomap depending on the backend). |
| 12 | Calibration Curve | in **PyCaret**, you can generate a **calibration curve** to check how well your model’s predicted probabilities match the actual outcomes.  What is a Calibration Curve?   * A calibration curve compares predicted probabilities with observed probabilities. * Ideally, predictions should lie on the diagonal line (perfect calibration). * If the curve is above the diagonal → model underestimates probabilities. * If below the diagonal → model overestimates probabilities. |
| 13 | Validation Curve | A **validation curve** shows how a model’s performance changes when you vary the value of a **single hyperparameter**.  **Where does "L1 ratio" come from?**   * Some models in PyCaret (like **Logistic Regression**, **Elastic Net**, and related linear models) support **elastic net regularization**. * Elastic Net combines:   + **L1 (Lasso)** → feature selection (drives some coefficients to 0).   + **L2 (Ridge)** → coefficient shrinkage (keeps all features, reduces variance).   The **L1 ratio** parameter (α) controls the mix between L1 and L2:     * l1\_ratio = 0 → pure **Ridge (L2)** * l1\_ratio = 1 → pure **Lasso (L1)** * 0 < l1\_ratio < 1 → **Elastic Net**   X-axis → l1\_ratio values (from 0 to 1).  Y-axis → performance metric (e.g., Accuracy, AUC depending on your setup). |
| 14 | Dimension Plot | The **Dimensions plot** is a **2D projection** of your dataset created using **dimensionality reduction** techniques (like **PCA, t-SNE, or UMAP**) so you can **visualize high-dimensional data**.  It helps you see:   * Whether classes are separable in lower dimensions. * Clusters, overlaps, or hidden structures in the data. * Possible class imbalance visually. |
| 15 | Feature Importance | Helps you:   * Identify the most influential features. * Detect irrelevant / weak predictors. * Guide feature selection and interpretability. |
| 16 | Decision Boundary | A decision boundary plot visualizes how a classifier separates classes in 2D feature space.  It shows the regions of the input space where the model predicts each class.  Useful for understanding how linear or non-linear the classifier’s boundaries are.  Works best when dataset has 2 numerical features (or reduced to 2D via PCA). |
| 17 | Lift Chart | A **lift chart** is used to evaluate the effectiveness of a **classification model (usually binary)** by comparing how much better the model is at predicting positive outcomes than random guessing.   * It plots the **lift score** (improvement over baseline) against **deciles of predicted probability**. * **Lift = (Model’s precision in a decile) / (Baseline precision)**. * A good model shows **higher lift in the top deciles**, meaning it ranks true positives better than random chance.   **Interpretation**   * Lift > 1 in the first few deciles → the model effectively identifies positive cases early. * Lift close to 1 → the model is no better than random. * Steep curve in early deciles → good model for targeted campaigns (like marketing, fraud detection) |
| 18 | Gain Chart | A **Gain Chart** (also called **Cumulative Gain Chart**) shows how well a classification model separates the positive class compared to random guessing. |
| 19 | Decision Tree | It’s a graphical representation of the splits made by a Decision Tree model:   * Root Node → best feature for first split. * Branches → decision rules (feature thresholds). * Leaf Nodes → final prediction (class or value). * This chart helps you interpret the decision-making process of the model. |
| 20 | **KS statistic Plot** | The **KS statistic** is used to measure the **separability** between the positive and negative classes in a **binary classification** model.  A higher KS value → better class separation.  Values range between 0 and 1:   * 0.2 – 0.3 → poor model * 0.4 – 0.6 → good model * > 0.7 → excellent model |

**What is L1 (lasso) and L2 (ridge) regularization when solver is 'saga' and penalty is 'elasticnet'.?**

**Grid Search and Random search?**

List of Plot Types – Regression

|  |  |  |
| --- | --- | --- |
| SN | Plot Type | Meaning |
| 1 | Pipeline Plot | In PyCaret, the pipeline plot shows you the machine learning workflow (pipeline) that PyCaret automatically creates during setup().  It includes all the preprocessing steps (imputation, encoding, scaling, transformations, feature selection, etc.) along with the estimator (model) at the end. |
| 2 | HyperParameters | Parameters that are set for a model. |
| 3 | Residuals | In PyCaret Regression, the residuals plot is used to visualize how well the model fits the data by comparing predicted values vs. residuals (errors).  Residual=Actual Value−Predicted Value |
| 4 | Prediction Error | What the Prediction Error Plot shows   * X-axis → Actual values (from dataset) * Y-axis → Predicted values (from the model) * 45° diagonal line → Perfect predictions (Predicted = Actual)   **Prediction Error Plot** → shows Predicted vs. Actual values → helps check overall predictive ability & alignment. |
| 5 | Cooks Distance | **Cook’s distance** helps identify **influential observations (outliers)** that have a disproportionate impact on the regression coefficients. Large Cook’s distance values indicate points that might unduly affect the regression model.  X-axis: Observation index (data points).  Y-axis: Cook’s distance values.  Dashed line: A threshold. Points above it are considered influential outliers |
| 6 | Feature Selection | the feature selection plot helps you see which features are most important for your trained model. |
| 7 | Learning Curve | **Learning Curve plot** to see how model performance changes as the training set size increases  If training and validation curves converge at a high score → model has good generalization.  If there’s a large gap (training much better than validation) → model may be overfitting.  If both curves are low and close → model may be underfitting (too simple).  If curves improve with more data → you might benefit from a larger dataset. |
| 8 | Manifold Learning | the **manifold learning plot** is a visualization that projects your dataset into a **2D space** using non-linear dimensionality reduction methods (like **t-SNE** or **UMAP**) and colors the points by their **target values**. |
| 9 | Validation Curve | **Validation Curve** to study how model performance changes with respect to **one hyperparameter**.   **X-axis** → values of a specific hyperparameter (e.g., max\_depth, n\_estimators).   **Y-axis** → model performance (e.g., R², RMSE, MAE depending on the scoring metric).   **Two lines** →   * **Training score** (performance on training set). * **Cross-validation score** (performance on unseen data).   How to Interpret   * If training score is high but validation score is much lower → overfitting. * If both training & validation scores are low → underfitting. * The sweet spot is where validation score is maximized and close to training score. |
| 10 | Feature Importance | **Feature Importance** tells you how much each input variable contributes to the predictions made by the model. |
| 11 | Decision Tree | In **PyCaret regression**, once you’ve created a **Decision Tree Regressor**  What the Decision Tree Plot Shows  Root node → the first split (most important feature).  Internal nodes → conditions (feature & threshold) that split the data.  Leaf nodes → the predicted target value (mean of samples in that node).  Numbers in nodes → sample counts & average prediction. |
| 12 | Interactive Residuals | **Interactive Residuals Plot** in **PyCaret regression**. This is a very useful plot to **explore model errors** in an interactive way. |

**List of Plot Types – Clustering**

|  |  |  |
| --- | --- | --- |
| SN | Plot Type | Meaning |
| 1 | 2d Cluster plot | In **PyCaret regression**, a **2D Cluster PCA Plot** is used to **visualize high-dimensional data** by reducing it to 2 principal components and optionally showing clusters in the feature space. |
| 2 | 3D TSNE Plot for clusters | In **PyCaret regression**, you can generate a **3D t-SNE plot** to visualize clusters or structure in your high-dimensional data.  **Why Use 3D t-SNE**   * Detect clusters or subgroups in your data. * Identify outliers that don’t belong to any cluster. * Observe non-linear structures that PCA might miss. * Especially useful for high-dimensional datasets where 2D PCA is insufficient. |
| 3 | Elbow Plot | In **PyCaret**, the **Elbow Plot** is used to determine the **optimal number of clusters** for clustering algorithms like **K-Means**.  What the Elbow Plot Shows   * X-axis: Number of clusters k. * Y-axis: Sum of Squared Errors (SSE) or within-cluster inertia. * Elbow point: The value of k where adding more clusters does not significantly reduce SSE → considered optimal. |
| 4 | **Silhouette Plot** | The **Silhouette Plot** is used to evaluate the **quality of clustering** after applying clustering algorithms like **K-Means**, **Agglomerative Clustering**, etc. It shows how well each data point fits within its cluster.  What the Silhouette Plot Shows   * X-axis: Silhouette coefficient (ranges from -1 to 1). * Y-axis: Cluster assignment for each data point.   Interpretation:   * Value close to +1: Data point is well-clustered. * Value around 0: Data point lies between clusters. * Value close to -1: Data point may be assigned to the wrong cluster.   Why It’s Useful   * Evaluates the cohesion and separation of clusters. * Helps identify poorly assigned points. * Useful to validate the number of clusters chosen (complements the Elbow Plot). |
| 5 | Distance Plot | In PyCaret clustering, the Distance Plot (also called Distance to Centroid Plot) helps visualize how far each data point is from its assigned cluster centroid. It’s useful for detecting outliers or points that don’t fit well into any cluster. How to Interpret the Plot  1. **Clusters:**    * Points that are close together → similar in the original feature space.    * Different colors → cluster assignments from K-Means or other clustering algorithm. 2. **Separation Between Clusters:**    * Large gaps between clusters → well-separated clusters.    * Overlapping clusters → may indicate poor separation or too many clusters. 3. **Outliers:**    * Points far away from their cluster → possible outliers. 4. **Distance to Centroid (if using distance plot):**    * Distance from each point to its cluster centroid is often represented by **size or color intensity**.    * Bigger distance → poorly clustered / potential anomaly. |
| 6 | Distribution Plot | The **distribution plot** shows how the **predicted values** (or residuals, or target variable) are distributed. |