**Data Cleaning vs Data Processing**

**Data Cleaning**

Focuses on fixing or removing bad data:

* Handling missing values
* Removing duplicates
* Correcting inconsistent entries

"North Amer", "North America", "N. America" — same meaning, different spellings

"Petrol", "petrol", "GAS" — inconsistent capitalization

* Handling outliers
* Data type issues:

Make sure each column’s data type is correct:

Year → integer

Price → float

* Logical consistency:

Check relationships make sense, e.g.:

Newer cars shouldn’t have huge mileage.

Higher prices should roughly align with “High” sales classification.

**Data Preprocessing**

Goes beyond cleaning — it’s about preparing the data for modeling:

* **Encoding** categorical features (like Fuel\_Type, Region)
* **Scaling/normalizing** numeric columns (if your model needs it)
* **Feature engineering** (creating new columns from existing ones)
* **Splitting** data into training/testing sets
* **Balancing** target classes if needed

**In short:**

* If your dataset is clean → you can **skip cleaning** steps.
* But you still **must do preprocessing** steps like encoding, scaling, and splitting before training a mode
* **Choosing the target**

When we do **data preprocessing**, we usually prepare the data for a **machine learning task** — such as predicting something.  
Looking at your dataset, the column Sales\_Classification contains values like **“High”** and **“Low”**, which look like *labels*.

That suggests the dataset could be used for a **classification task**, for example:

“Predict whether a car model’s sales will be High or Low based on its features (price, engine size, region, etc.)”

In this case:

* The **input features (X)** are: all other columns (Model, Year, Region, etc.)
* The **target (y)** is: Sales\_Classification

That’s why I said it’s *likely the target* — because it’s what you might want the model to learn to predict.

* **When we do up/down sampling**

**Upsampling (or downsampling)** is done **only on the target variable** — more precisely, on the **rows of the dataset based on the target classes**.

**What to choose?**

It depends on your model and dataset size:

* Since your dataset is **fairly large**, **downsampling** is a valid choice — it’ll still leave you with plenty of data.
* But if you don’t want to lose data, **upsampling** (or even better, **SMOTE**) is a safer and more balanced approach.

**When you upsample (the "High" class)**

You don’t fill other columns manually — you **duplicate** the *entire rows* that belong to the minority class (High).

So every time a “High” row is copied, all its feature values (like Model, Engine\_Size\_L, Region, Year, etc.) are copied with it.

That way, your dataset remains **consistent** — the features always match their correct target label.

**Handle Irrelevant or Redundant Features(Feature Selection) (Data Processing)**

Not all features contribute to your model.

**Methods:**

1. **Drop irrelevant columns**
   * Example: IDs, URLs, or any unrelated metadata.
2. **Drop highly correlated features**
   * Remove one of two features with strong correlation (redundant info).
3. **Feature selection based on variance**

Remove columns with very little change (low variance).

**Data Cleaning Process**

**Step 1: Handle Missing Values**

Missing data is one of the most common issues.

**Methods:**

1. **Remove missing data**
   * **Drop rows** that have missing values (if few rows are affected).
   * **Drop columns** if a large portion of that column is missing (e.g., >60%).
2. **Imputation (filling in values)**
   * **Mean/Median/Mode imputation** (for numeric/categorical features).
   * **Forward/Backward fill** (for time-series data).
   * **Constant fill** (replace with a fixed value like 0 or “unknown”).
3. **Prediction-based imputation**
   * Use a model (e.g., regression, kNN, decision tree) to predict missing values based on other features.

**Method 1: Remove Missing Data**

This is the simplest approach.

**🟢 When to Drop Rows:**

* Only **a few rows** have missing data (e.g., less than 5%).
* The rows are not critical or removing them won’t bias your data.
* Example: In a 10,000-row dataset, 50 missing rows is fine to remove.

**🟢 When to Drop Columns:**

* The column has **too many missing values** (e.g., >50–60%).
* The column’s information is **not important** or **can’t be reliably filled**.
* Example: “Middle Name” or “Second Phone Number”.

**🔴 Avoid Dropping:**

* When missingness is **not random** (dropping could bias results).
* When the column is **important for prediction**.

**Situation:**

* The **column is important** (you can’t just drop it).
* About **50% of its values are missing** (too much to ignore).

So what do you do?

**Why You *Can’t* Simply Drop It**

Dropping the column would mean losing **critical predictive information**, which can harm model performance.  
But simple mean/median filling might **introduce bias** — because half of the values are made-up, not real data.

So you need to **balance preservation and reliability**.

**What You Can Do Instead — Step by Step**

**1)Try to Understand *Why* It’s Missing**

Ask:

* Is it **randomly missing** (no pattern)?
* Or **systematically missing** (e.g., certain groups didn’t report it)?

If it’s systematic (not random), you might need to **model** or **mark** that missingness.

👉 Example:  
If 50% of “Income” is missing only for unemployed people, then missingness is **meaningful** — it carries information about employment.

**2)Impute Using Smarter Techniques**

**A. Predictive Imputation (Best for Important Columns)**

Use other correlated features to **predict the missing values**.

Examples:

* Predict income from **age, job title, education, and experience**.
* Predict temperature from **humidity, season, and time**.

Or build a simple **regression model** yourself (using the non-missing part as training data).

🧠 Why this works:  
It uses actual data relationships instead of random averages.

**B. Group-Based Imputation (Good Compromise)**

If full predictive modeling is too complex, you can fill missing values **within similar groups**.

Why this works:  
People with the same job title or education level tend to have similar incomes — so the group median makes more sense.

**Method 2: Imputation (Filling Values)**

You fill in the missing values with substitutes that make sense statistically.

**Mean / Median / Mode Imputation**

* **Mean:** For numeric continuous data (e.g., income, height).
  + Use when data is roughly **normally distributed**.
* **Median:** For numeric skewed data (e.g., house prices).
  + More robust to outliers, For **skewed** numeric data.
* **Mode:** For categorical data (e.g., “Male/Female/Other)

**How to detect distribution type**

You can find out in several ways — visual, statistical, or descriptive.

**1. Visual inspection (the easiest and most common)**

* **Histogram** → Shows frequency of values.
  + Symmetrical = normal.
  + Tail to right = positively skewed.
  + Tail to left = negatively skewed.
* **Boxplot** → Helps detect skewness through the whiskers.
  + If one whisker is much longer → skewed.
  + If box centered → roughly normal.

**2. Summary statistics**

Check **mean, median, and mode** relation:

| **Distribution Type** | **Relation Between Mean, Median, Mode** |
| --- | --- |
| Normal | mean ≈ median ≈ mode |
| Right (positive) skew | mean > median > mode |
| Left (negative) skew | mean < median < mode |

**How to detect missing values:**

A screen shot of a computer

AI-generated content may be incorrect.

**Step 2: Handle Duplicates**

Duplicate rows may exist, especially in merged or scraped datasets.

**Methods:**

1. **Identify and drop exact duplicates** (same values across all columns).

There are **two levels** of duplicates you should look for:

**1. Exact duplicates (full row duplicates)**

These are rows where **every column has identical values**.

**2. Partial duplicates (based on specific columns)**

Sometimes rows differ slightly, but represent the same entity.

Example:

| **ID** | **Name** | **Email** | **City** |
| --- | --- | --- | --- |
| 101 | Ahmed Ali | ahmed@gmail.com | Cairo |
| 101 | Ahmed Ali | ahmed.ali@gmail.com | Cairo |

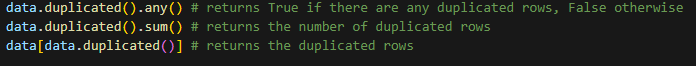
→ These are duplicates **based on “ID” or “Name”**, but not exactly identical.

So, you can check duplicates **based on selected columns** — for example:

* Customer ID
* Product name and date
* Email address

💡 This is important if your dataset doesn’t have a strict unique identifier.

**How to detect:**



**Step 3: Handle Inconsistent Data**

This happens when data is entered in multiple formats or units.

**Methods:**

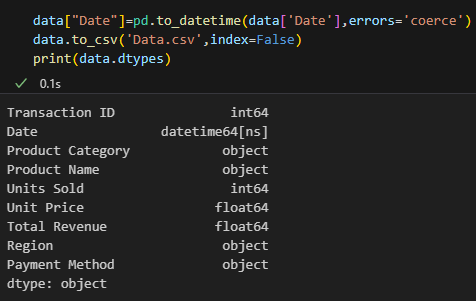
1. **Standardize formats**
   * Dates → convert all to one format (e.g., YYYY-MM-DD).
   * Text → make lowercase, trim spaces, unify spellings.
2. **Standardize units**
   * Example: convert all distances to kilometers, or all currencies to one type.
   * If some numeric columns are in different units (e.g., revenue in $ vs cents), standardize them
   * When we say **standardize units**, it usually refers to **making sure all values are measured in the same scale or unit**, e.g.:
   * Weight: grams vs kilograms → convert all to grams
   * Price: USD vs EUR → convert all to one currency
3. **Normalize categorical values**
   * Example: “Male”, “male”, “M” → all to “Male”.
4. Check for text encoding or special characters

There is some steps you need to check on before handling outliers

1. Check if the columns is in the right data types

A screen shot of a computer

AI-generated content may be incorrect.

1. convert Date column to datetime64 if it’s an object  
     
     
     
   If it’s datetime64[ns], pandas already converted all values correctly — the *format is internally unified*, even if it looks different when printed
2. Remove whitespaces at the beginning and the ending of the text   
   
3. Similar categories that can be merged 🡪 you can do that by using value\_counts() or unique(), if the data is so big you can you use the process.extract function to be able to compare   
     
   A screen shot of a computer program

   AI-generated content may be incorrect.
4. Check for non ascii code and fix it  
    A screen shot of a computer code

   AI-generated content may be incorrect.

**Step 4: Handle Incorrect Data Types**

Sometimes data is read as the wrong type (e.g., numbers as text).

**Methods:**

1. **Convert to correct data types**
   * Strings to dates, numbers to floats/integers, etc.
   * Here we have a date column and the dates stored in string we need to convert the column and values into datetime
   * Another case we have an expected numeric column so we need to convert it into numeric values.
2. **Detect numeric-like strings**
   * Example: “10.0” stored as string → convert to float.

If a column is an object then there is some possibilities

1. First all of it’s values are string
2. Second it was a numeric column and you add a string value or no compatible value with it so pandas upcast the column into object here

**Step 5: Handle Categorical Data Issues**

Categorical columns can have typos or too many unique values.

**Methods:**

1. **Fix typos and unify labels**
   * Example: “NY”, “New York”, “N.Y.” → unify.
2. **Group rare categories**
   * **Too many unique categories**
     1. Example: Region has 50 unique values, but most of them only appear 1–2 times.
     2. This can cause problems for machine learning models, especially when encoding (like one-hot encoding), because it creates **many sparse features**.
   * **Rare categories**
     1. Categories that occur very infrequently (e.g., only 1 or 2 rows out of 50,000).
     2. Rare categories can be merged into an **“Other”** category to reduce noise and avoid overfitting.

**Step 6: Handle Noise and Inconsistent Entries**

Some entries may be random or contain junk.

**Methods:**

1. **Filter impossible values**
   * Example: Age = 300 or negative.
2. **Apply smoothing or aggregation**
   * For time-series, use rolling averages or median filters.
3. **Manual validation**
   * Spot check random samples for unrealistic entries.

A computer screen shot of a code

AI-generated content may be incorrect.

**Step 7: Handle Outliers**

Outliers can distort your model’s understanding.

**Methods:**

1. **Remove outliers**
   * Use statistical methods (Z-score, IQR) to detect and drop them.
2. **Cap or floor outliers**
   * Replace extreme values beyond certain limits with boundary values.
3. **Transform data**
   * Apply log or square root transformation to reduce skewness.

A screenshot of a black screen

AI-generated content may be incorrect.

**Things to Consider**

* **Domain knowledge is key:** Is the “outlier” actually an error, or just a rare but valid case?
* **Column type matters:** Outlier treatment for Age is different than for Revenue.
* **Impact on model:** Overly aggressive removal can distort true data distribution.