Data cleaning and Preprocessing:

data = {

'Age': [25, 30, np.nan, 22, 40, 25, -5, 200],

'Salary': [50000, 60000, 55000, np.nan, 70000, 50000, 60000, 1000000],

'Department': ['HR', 'Engineering', 'hr', 'Finance', np.nan, 'HR', 'HR', 'Finance'],

'Gender': ['Male', 'Female', 'Female', 'Male', 'Male', 'Male', 'Male', 'Female'],

'Review': ['Good job!', 'excellent', 'Bad!!', 'Great work', 'poor', 'Good job!', 'Nice!!', 'Bad!!'],

'Target': [1, 0, 1, 0, 1, 1, 0, 0]

}

df = pd.DataFrame(data)

**🔹 1. Handling Missing Data**

Missing values can occur due to data entry errors, collection issues, or merging datasets.

* **Detection**: isnull(), info(), describe() in pandas
* **Techniques to handle:**
  + **Removal**: Drop rows/columns with too many missing values.
  + **Imputation**:
    - Mean/median/mode imputation
    - Forward/backward fill (fill with previous/next value in the column)
    - Model-based imputation (e.g., KNN, regression)

df['Age'] = df['Age'].fillna(df['Age'].mean())

df['Salary'] = df['Salary'].fillna(df['Salary'].median())

df['Department'] = df['Department'].fillna(df['Department'].mode()[0])

**🔹 2. Removing Duplicates**

Duplicate rows can bias the model.

* **Detection**: df.duplicated()
* **Action**: df.drop\_duplicates()

**🔹 3. Data Type Corrections**

Ensuring each column has the appropriate data type. If it is true/false we can save the memory by changing datatype as “category”.

* E.g., Dates should be datetime, categories as category type, etc.
* Use df.astype() or pd.to\_datetime()

df['Department'] = df['Department'].astype('category')

**🔹 4. Handling Outliers**

Outliers can skew the results of the model, especially linear models. We can remove outliers with IQR capping.

* **Detection**:
  + IQR method
  + Z-score
  + Boxplots
* **Handling**:
  + Removal
  + Transformation (e.g., log)
  + Capping (e.g., winsorization)

for col in ['Age', 'Salary']:

Q1 = df[col].quantile(0.25)

Q3 = df[col].quantile(0.75)

IQR = Q3 - Q1

lower = Q1 - 1.5 \* IQR

upper = Q3 + 1.5 \* IQR

df[col] = np.where(df[col] < lower, lower, df[col])

df[col] = np.where(df[col] > upper, upper, df[col])

**🔹 5. Categorical Variable Cleaning**

Messy text, inconsistent categories, or mixed cases.

* **Standardization**:
  + Lowercase/uppercase
  + Stripping whitespace
  + Merging similar categories (map() or replace())
* **Encoding**: converting text into numeric
  + One-hot encoding (convert category(yes,no,male,female) into number(0, 1) )
  + Label encoding (0, 1, 2)
  + Target encoding

df['Department'] = df['Department'].str.strip().str.lower()

**🔹 6. Text Data Cleaning**

For NLP models.

* Lowercasing
* Removing punctuation, stopwords
* Lemmatization or stemming
* Removing HTML tags, special characters

**🔹 7. Normalization / Feature scaling**

Essential for models sensitive to feature magnitudes (e.g., KNN, SVM).

# from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler

* **MinMaxScaler (**Scales features to a fixed range, usually **0 to 1)**
* **StandardScaler** (Scales features so they have **mean = 0** and **standard deviation = 1**)
* **RobustScaler** (good for outliers)

| **Scaler** | **Handles Outliers?** | **Range** | **Use When...** |
| --- | --- | --- | --- |
| MinMaxScaler | ❌ No | 0 to 1 | Data is clean and bounded |
| StandardScaler | ❌ No | mean = 0 | Data is roughly normally distributed |
| RobustScaler | ✅ Yes | Around median | Data has **outliers** (like salary!) |

**🔹 8. Feature Consistency Checks**

* No contradictory values
* Validate ranges (e.g., age shouldn't be negative)
* Cross-field consistency (e.g., "Date of birth" + "Age" should match)

df = df[(df['Age'] > -3) & (df['Age'] < 3)]

If any value is **beyond -3 or +3**, it's probably:

* an outlier
* a data entry mistake
* or something inconsistent

**🔹 9. Handling Imbalanced Data**

If classes are imbalanced (in classification problems):

**✅ Resampling**

**Purpose:**  
To balance the dataset by adjusting the number of samples in each class.

**Types:**

* **Oversampling:** Increase the number of samples in the minority class by duplicating or generating new ones.
* **Undersampling:** Reduce the number of samples in the majority class by removing some data points.

**When to Use:**

* When the dataset is small or moderately imbalanced.
* Helps the model treat both classes more equally.

**✅ Class Weight Adjustment**

**Purpose:**  
To tell the model that the minority class is more important without changing the dataset.

**How it Works:**

* Assigns more weight to the minority class and less weight to the majority class.
* This makes the model pay more attention to the minority class during training.

**When to Use:**

* When you want a simple and effective solution without altering the data.
* Works well with most machine learning algorithms.

**✅ Anomaly Detection**

**Purpose:**  
To treat rare cases as anomalies or outliers.

**How it Works:**

* The model learns what is “normal” and identifies unusual patterns as possible rare events.
* Commonly used in fraud detection, rare disease identification, and defect detection.

**When to Use:**

* When the minority class is extremely small (less than 1–2% of the data).
* Useful when the rare class is very important to detect.

**🔹 10. Data Leakage Prevention**

Make sure future data doesn't influence the training set.

* Split data **before** any transformations like imputation or scaling that learn from data.

**What is Data Leakage?**

* Data leakage happens when information from **outside the training dataset** is used to create the model.
* This can cause the model to perform **unrealistically well** during training, but **fail badly in real-world testing**.

**Key Tip to Prevent Leakage:**

* **Always split your data** into training and testing sets **before applying any transformations** that learn from the data (like imputation, scaling, or encoding).
* This way, the test set remains completely unseen by the model during training.

**Example Scenario:**

* If you fill missing values using the mean of the full dataset **before splitting**, you are leaking test data info into training.

**When to Apply This:**

* Before scaling, imputing missing values, encoding categorical variables, etc.

Feature engineering:

Create new meaningful features from your existing data to help the model learn better patterns. **Example:** Extract year and month from a date column, or create interaction features, Create new feature: Salary in thousands

# df['Salary\_K'] = df['Salary'] / 1000

Train-test Split:

Split data into training and testing sets to evaluate model performance on unseen data.

X = df[['Join\_Year', 'Join\_Month', 'Salary\_K']]

y = df['Target']

# Split 70% train, 30% test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

Model selection and Training:

**1. Based on Problem Type**

* **Classification** → Predicting categories (e.g., fraud or not)
* **Regression** → Predicting continuous values (e.g., house price)

**2. Based on Data Size & Quality**

* Large dataset? → RandomForest, XGBoost
* Small dataset? → Logistic/Linear Regression, KNN
* Outliers? → Use tree-based models or Robust models

**3. Based on Accuracy Needs**

* Try multiple models → Compare using metrics like accuracy, F1-score, or RMSE

**Select the model:**

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

models = {

"Decision Tree": DecisionTreeClassifier(),

"Random Forest": RandomForestClassifier(),

"SVM": SVC()

}

**Train the model:**

for name, model in models.items():

model.fit(X\_train, y\_train)

preds = model.predict(X\_test)

print(f"{name} Accuracy: {accuracy\_score(y\_test, preds)}")

**make predections:**

predictions = model.predict(X\_test)

Model tuning(validation):

Optimize model parameters for better performance, Default values may not give the best accuracy, Proper tuning improves model performance, generalization, and reduces overfitting.

Common Tuning Techniques: Grid search, Random Search, Bayesian Optimization / Automated Tuning.

1. **Hyperparameter Tuning**

* Grid Search: Try all combinations
* Random Search: Try random combinations
* Bayesian Optimization, Optuna, Hyperopt: More intelligent search

2. **Feature selection**

Choosing only the most relevant input features for the model to reduce over fitting, speed up ttaining, improve model accuracy.

Wrapper methods: Recursive Feature Elimination (RFE)

# selector = RFE(RandomForestClassifier(), n\_features\_to\_select=5)

3. **model selection:**

Trying different algorithms to find the best performance on validation set.

| **Tuning Type** | **Goal** | **Applied On** |
| --- | --- | --- |
| Hyperparameter Tuning | Improve model behaviour | Validation Set |
| Feature Selection | Reduce input noise | Training + Validation Set |
| Model Selection | Choose best algorithm | Validation Set |

Evaluate/inference on Test set(metrics):

**Test set**: Final, untouched data used once to evaluate model performance

**Classification** →

| **Aspect** | **Details** |
| --- | --- |
| **Goal** | Predict discrete categories/labels |
| **Examples** | - Spam vs Not Spam (Binary classification)- Dog vs Cat vs Bird (Multiclass classification)- Disease present vs Not |
| **Output** | - A class label (e.g., 1, 0, "yes", "no")- Probabilities per class |
| **Evaluation Metrics** | - **Accuracy**: % of correct predictions- **Precision**: Out of predicted positives, how many were right?- **Recall**: Out of actual positives, how many did we catch?- **F1-Score**: Harmonic mean of precision and recall- **AUC-ROC**: How well the model distinguishes between classes- **Confusion Matrix**: TP, FP, FN, TN counts |
| **Visualization** | - Confusion Matrix- ROC Curve- Precision-Recall Curve |
| **Aspect** | **Details** |

**Regression** →

| **Aspect** | **Details** |
| --- | --- |
| **Goal** | Predict continuous numeric values |
| **Examples** | - Predict house price- Forecast sales or temperature- Estimate time until failure |
| **Output** | A real number like 189.32, 42.0, etc. |
| **Evaluation Metrics** | - **MSE (Mean Squared Error)**: Average of squared errors  - **RMSE (Root MSE)**: Square root of MSE (same units as target  **- MAE (Mean Absolute Error)**: Average of absolute differences  - **R² Score (R-squared)**: How much variance is explained by the model (1 is best) |
| **Visualization** | - Predicted vs Actual Plot- Residual Plot |

# from sklearn.metrics import accuracy\_score

# print(accuracy\_score(y\_test, predictions))

Model validation:

Check model’s stability and robustness by training/testing on different folds.

Model deployment:

Deployment is the process of making the trained model available in a production environment so it can serve predictions. Puts the model in an environment where users/applications can access it, Could be a REST API, stream processor, microservice, or even on-device.

* Model loading
* Monitoring
* Logging
* Scaling

Model Serving:

Model serving is a specific kind of deployment setup where the model is hosted as a service (usually via an API) and can handle multiple inference requests in real-time.

Enables real-time or near real-time inference, offen handles Versioning, Auto-scaling, Health checks, A/B testing.

Tools: Mlflow, Torchserve, TF serving, Kserve