**📌 Q: What is Data Preprocessing?**

**Data Preprocessing** is the technique of **cleaning and transforming raw data** into a usable format so that Machine Learning models or data analytics techniques can give accurate, meaningful results.

📉 **Why?**  
Raw data often has:

* Missing values
* Inconsistent formats
* Duplicates
* Noise and outliers
* Irrelevant or redundant features

🛠️ Goal: Improve **data quality** and ensure better **model performance.**

**✅ Main Steps in Data Preprocessing:**

Raw Data ──► Cleaning ──► Integration ──► Transformation ──► Reduction ──► Discretization ──► Final Dataset

**🔹1. Data Cleaning**

**Purpose: Fix incorrect, inconsistent, incomplete, or noisy data.**

**A. Handling Missing Data**

📌 **Reasons for Missing Data:**

* Human error
* Sensor malfunction
* Data corruption

📊 **Types:**

* MCAR (Missing Completely At Random)
* MAR (Missing At Random)
* MNAR (Missing Not At Random)

**📌 Techniques to Handle:**

| **Method** | **Use Case** | **Example** |
| --- | --- | --- |
| Drop rows/columns | Large dataset with few missing | df.dropna() |
| Fill with mean/median | Numeric columns | df.fillna(df.mean()) |
| Fill with mode | Categorical columns | df['col'].fillna(df['col'].mode()[0]) |
| Predictive Imputation | Use ML model to predict missing values | Advanced use case |

📦 **Python Example:**

import pandas as pd

df = pd.read\_csv('data.csv')

# Drop rows with any missing values

df\_cleaned = df.dropna()

# Fill missing numeric values with mean

df['Age'].fillna(df['Age'].mean(), inplace=True)

**🔹2. Removing Duplicates**

Duplicates can bias analysis and model training.

📦 **Python Example:**

# Check for duplicates

df.duplicated().sum()

# Remove duplicates

df = df.drop\_duplicates()

**🔹3. Data Transformation**

Transforming data into a consistent and useful format.

**A. Standardization (Z-score Normalization)**

Formula:

z = (x - mean) / std\_dev

Used when features have different units (e.g., kg vs cm)

**B. Min-Max Normalization**

Brings data into range [0, 1]

x' = (x - min) / (max - min)

**C. Encoding Categorical Data**

| **Type** | **Method** | **Code** |
| --- | --- | --- |
| Nominal | One-Hot Encoding | pd.get\_dummies(df) |
| Ordinal | Label Encoding | LabelEncoder().fit\_transform() |

📦 **Python Code:**

from sklearn.preprocessing import MinMaxScaler, LabelEncoder

# Min-Max Scaling

scaler = MinMaxScaler()

df[['Salary']] = scaler.fit\_transform(df[['Salary']])

# Label Encoding

le = LabelEncoder()

df['Gender'] = le.fit\_transform(df['Gender']) # M:1, F:0

**🔹4. Data Integration**

Combining data from multiple sources into a single dataset.

📌 Examples:

* Merging multiple CSV files
* Joining SQL tables

df1 = pd.read\_csv('sales.csv')

df2 = pd.read\_csv('employees.csv')

df\_merged = pd.merge(df1, df2, on='Employee\_ID')

**🔹5. Data Reduction**

Reducing volume but preserving integrity.

**Techniques:**

* Dimensionality Reduction (e.g., PCA)
* Aggregation (e.g., average weekly sales)
* Sampling

**🔹6. Discretization**

Transforming continuous data into categorical.

📌 Example:

df['Age\_Group'] = pd.cut(df['Age'], bins=[0, 18, 35, 60, 100], labels=['Teen', 'Young', 'Adult', 'Senior'])

**🔧 Essential Python Libraries for Preprocessing**

| **Library** | **Purpose** |
| --- | --- |
| pandas | Data manipulation and preprocessing |
| numpy | Numeric computations |
| sklearn.preprocessing | Scaling, encoding, imputation |
| scipy.stats | Statistical handling |
| missingno | Visualizing missing data |
| matplotlib / seaborn | Visual analysis for preprocessing |

**📊 DIAGRAM – Example Workflow of Preprocessing**

┌────────────┐

│ Raw Dataset│

└─────┬──────┘

│

┌─────▼─────┐

│ Drop Duplicates │

└─────┬──────┘

│

┌─────▼─────┐

│ Handle Missing │

└─────┬──────┘

│

┌─────▼─────┐

│ Transform / Scale│

└─────┬──────┘

│

┌─────▼─────┐

│ Encode Categorical │

└─────┬──────┘

│

┌─────▼─────┐

│ Final Clean Data │

└────────────┘

**✅ Final Notes:**

* Preprocessing ensures cleaner, more usable data.
* Different methods suit different data types.
* Python libraries like Pandas and Scikit-learn simplify the task.

Here’s a **very detailed yet simple answer** for:

**✅ Q4 b) Explain scikit-learn library for matplotlib with very simple example.**

**🔹 What is Scikit-learn?**

📌 **Scikit-learn** is a powerful Python library used for:

* Machine Learning
* Data Mining
* Data Preprocessing
* Model Evaluation

It works **with** libraries like:

* **NumPy** (for arrays)
* **Pandas** (for data)
* **Matplotlib** (for plotting results)

**🔹 What is Matplotlib?**

📌 **Matplotlib** is a visualization library in Python used to create:

* Line plots
* Bar graphs
* Scatter plots
* Histograms  
  and more.

**🔄 How do Scikit-learn & Matplotlib work together?**

* Scikit-learn is used to **train a model**.
* Matplotlib is used to **visualize** results of that model.

✅ Example: Visualizing the **decision boundary** of a classifier trained using scikit-learn.

**✅ Very Simple Example: Classification with Plot**

**🎯 Problem: Train a classifier using Scikit-learn and visualize using Matplotlib.**

# Import Libraries

import matplotlib.pyplot as plt

from sklearn.datasets import make\_classification

from sklearn.linear\_model import LogisticRegression

# Generate simple synthetic data

from sklearn.model\_selection import train\_test\_split

X, y = make\_classification(n\_samples=100, n\_features=2,

n\_informative=2, n\_redundant=0,

random\_state=1)

# Train-Test Split

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

# Train Logistic Regression model

model = LogisticRegression()

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

# Visualize using matplotlib

plt.scatter(X\_test[:, 0], X\_test[:, 1], c=y\_test, cmap='coolwarm')

plt.title("Test Data - True Labels")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()

**🔍 Output Explanation:**

* 🧠 make\_classification() creates a fake dataset (2 features).
* 📊 plt.scatter() creates a **colored scatter plot** based on labels.
* 🎯 Classifier is trained using **LogisticRegression()** from Scikit-learn.
* 📈 Matplotlib shows test points with their actual classes.

**📌 Summary:**

| **Aspect** | **Description** |
| --- | --- |
| Scikit-learn | Trains the model |
| Matplotlib | Plots or visualizes the results |
| Integration | You train with sklearn, then show results with matplotlib |

**📊 Diagram (ASCII style)**

Scikit-learn Model

(LogisticRegression)

│

Trained on Data

│

▼ Predictions ▼

┌────────────┐ ┌────────────┐

│ Features │ │ Labels │

└────────────┘ └────────────┘

│

▼

matplotlib.pyplot

▶ Scatter Plot

Here is a **very very very detailed answer** to **Regression (Linear and Logistic)** based on the previous-year DSBDA questions from **2022, 2023 Jan, and 2024 May**. It includes:

✅ Complete theory  
✅ Mathematical equations  
✅ Python code  
✅ Diagrams  
✅ Detailed explanation of the **sigmoid function** and its **role in logistic regression**  
✅ Differences between Linear and Logistic Regression

**🔵 What is Regression?**

Regression is a **supervised learning technique** used to model the relationship between **input (independent)** and **output (dependent)** variables.

It predicts continuous or categorical values based on input features.

**📘 1. LINEAR REGRESSION**

**✅ Definition**

Linear Regression is a technique to model the **linear relationship** between **dependent variable (y)** and **independent variable(s) (x)**.

It predicts a **continuous value** (e.g., salary, price, age).

**📐 Mathematical Equation**

For **Simple Linear Regression (1 variable)**:

y=β0+β1x+ϵy = \beta\_0 + \beta\_1 x + \epsilon

Where:

* yy = dependent variable (target)
* xx = independent variable (feature)
* β0\beta\_0 = intercept
* β1\beta\_1 = slope of the line
* ϵ\epsilon = error term

**📊 Diagram**

y-axis

│ ●

│ ●

│ ●

│ ●

│ ●

│ ●

│ ●

│ ●\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ x-axis

**Best fit line** minimizes the **Sum of Squared Errors (SSE)**

**⚙️ How it Works**

* Tries to find the best-fitting line through the data points
* Uses **least squares method** to minimize the error between predicted and actual values

**✅ Example Code (Linear Regression)**

from sklearn.linear\_model import LinearRegression

import pandas as pd

# Sample Data

data = pd.DataFrame({

'Experience': [1, 2, 3, 4, 5],

'Salary': [30000, 35000, 40000, 45000, 50000]

})

X = data[['Experience']]

y = data['Salary']

# Model

model = LinearRegression()

model.fit(X, y)

# Prediction

print("Predicted salary for 6 years:", model.predict([[6]]))

**📘 2. LOGISTIC REGRESSION**

**✅ Definition**

Logistic Regression is used for **classification** problems (not regression!) – it predicts **probability of belonging to a class (0 or 1)**.

Despite its name, **logistic regression is a classification algorithm**, not regression.

**🧪 Use Case Example:**

* Email spam detection (Spam or Not Spam)
* Tumor prediction (Benign or Malignant)
* Customer churn (Yes or No)

**📐 Mathematical Model**

Linear regression gives:

y=β0+β1xy = \beta\_0 + \beta\_1 x

Logistic Regression applies **sigmoid function** to convert it into probability:

p=11+e−(β0+β1x)p = \frac{1}{1 + e^{-(\beta\_0 + \beta\_1 x)}}

This gives **p between 0 and 1**, which is interpreted as probability of class 1.

**📈 Sigmoid Function (Logistic Function)**

**✅ Equation:**

σ(z)=11+e−z\sigma(z) = \frac{1}{1 + e^{-z}}

Where:

* z=β0+β1xz = \beta\_0 + \beta\_1 x
* Output is always between **0 and 1**

**📊 Diagram of Sigmoid Function:**

1.0 ───────────────●───────────────

●

●

●

●

●

●

●

0 ─●───────────────────────────────

-6 -3 0 3 6 → z

**🧠 Role of Sigmoid in Logistic Regression**

* Converts **linear output to probability**
* If p≥0.5p \geq 0.5, class = 1
* If p<0.5p < 0.5, class = 0

It acts as a **threshold-based classifier** using probability.

**✅ Example Code (Logistic Regression)**

from sklearn.linear\_model import LogisticRegression

import pandas as pd

# Dataset

data = pd.DataFrame({

'Hours\_Studied': [1, 2, 3, 4, 5, 6],

'Passed': [0, 0, 0, 1, 1, 1]

})

X = data[['Hours\_Studied']]

y = data['Passed']

# Logistic Regression Model

model = LogisticRegression()

model.fit(X, y)

# Predicting

print("Pass probability if studied 3.5 hours:", model.predict\_proba([[3.5]]))

print("Predicted Class:", model.predict([[3.5]]))

**🔁 Differences: Linear Regression vs Logistic Regression**

| **Feature** | **Linear Regression** | **Logistic Regression** |
| --- | --- | --- |
| Purpose | Predict continuous values | Predict categorical classes |
| Output | Any real number | Value between 0 and 1 (probability) |
| Function | Linear Equation y=β0+β1xy = \beta\_0 + \beta\_1 x | Sigmoid function p=11+e−zp = \frac{1}{1 + e^{-z}} |
| Use Case | Predicting price, age, income | Classifying spam, disease, churn |
| Algorithm Type | Regression | Classification |
| Curve Type | Straight line | S-curve (sigmoid) |
| Loss Function | Mean Squared Error (MSE) | Log Loss / Cross Entropy |
| Decision Boundary | None | Based on threshold (e.g. 0.5) |
| Example | Predict house price | Predict if customer will buy |

**✏️ Summary Theory Points (useful for exam):**

* Linear Regression is used for **continuous output prediction**
* Logistic Regression is used for **binary classification**
* Logistic Regression applies the **sigmoid function** to map outputs to **probabilities**
* The sigmoid function ensures values lie between **0 and 1**
* If sigmoid output ≥ 0.5 → class = 1, else → class = 0
* Both are types of **supervised learning**
* Logistic regression uses **log-loss (cross-entropy)** instead of MSE
* Logistic regression is **linear in parameters**, but the output is **non-linear** due to sigmoid

**✅ Common Python Libraries Used**

* pandas → for data manipulation
* scikit-learn (sklearn) → for modeling (LinearRegression, LogisticRegression)
* numpy → for math operations
* matplotlib, seaborn → for visualization

Here’s a **very detailed answer** for:

📌 **Naive Bayes Classifier Theory + Applications**  
📌 **Q3 b) Email classification using Naive Bayes with "offer"=1 and "free"=1 (with full step-by-step working)**

Includes:  
✅ Detailed theory  
✅ Bayes Theorem explained  
✅ Types of Naive Bayes  
✅ Real-life applications  
✅ Diagram  
✅ Python code example  
✅ Solved numerical example from Q3b

**📘 Naïve Bayes Classifier**

**✅ Definition**

Naïve Bayes is a **probabilistic classifier** based on **Bayes’ Theorem** with a **naive assumption** that features are **independent** of each other given the class.

Despite the simplicity, it works well for text classification problems like spam detection.

**📐 Bayes’ Theorem**

P(C∣X)=P(X∣C)⋅P(C)P(X)P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}

Where:

* P(C∣X)P(C|X) = Posterior: Probability of class CC given data XX
* P(X∣C)P(X|C) = Likelihood: Probability of data XX given class CC
* P(C)P(C) = Prior probability of class
* P(X)P(X) = Evidence (can be ignored during comparison)

**🔁 Naïve Assumption**

P(X∣C)=P(x1∣C)⋅P(x2∣C)⋅⋯⋅P(xn∣C)P(X|C) = P(x\_1|C) \cdot P(x\_2|C) \cdot \dots \cdot P(x\_n|C)

That is, all features xix\_i are **conditionally independent**.

**🧠 Types of Naïve Bayes**

| **Type** | **Use Case** |
| --- | --- |
| **Gaussian** | Continuous data (e.g., age, height) |
| **Multinomial** | Discrete count data (e.g., text frequency) |
| **Bernoulli** | Binary data (e.g., word present/absent) |

**🔎 Applications of Naïve Bayes**

1. **Email spam detection**
2. **Text sentiment classification**
3. **Medical diagnosis**
4. **News categorization**
5. **Document classification**
6. **Recommendation systems**
7. **Credit scoring**
8. **Face recognition**
9. **Weather prediction**

**🧮 Numerical Example: Q3b (2023)**

**Question:**

Use Naive Bayes to classify an email with features **"offer"=1** and **"free"=1**

**📋 Training Data:**

| **Email** | **Offer** | **Free** | **Class** |
| --- | --- | --- | --- |
| 1 | 1 | 1 | Spam |
| 2 | 1 | 0 | Spam |
| 3 | 0 | 1 | Spam |
| 4 | 1 | 1 | Ham |
| 5 | 0 | 0 | Ham |

We need to classify:  
X = {"offer": 1, "free": 1}

**✅ Step 1: Count class priors**

| **Class** | **Count** | **P(Class)** |
| --- | --- | --- |
| Spam | 3 | 3/5 = 0.6 |
| Ham | 2 | 2/5 = 0.4 |

**✅ Step 2: Likelihood Probabilities**

Calculate for each feature given class:

**For class = Spam**

* P(offer=1 | Spam) = 2/3
* P(free=1 | Spam) = 2/3

**For class = Ham**

* P(offer=1 | Ham) = 1/2
* P(free=1 | Ham) = 1/2

**✅ Step 3: Apply Bayes Rule (ignore denominator)**

**For Spam:**

P(Spam∣X)∝P(offer=1∣Spam)⋅P(free=1∣Spam)⋅P(Spam)=23⋅23⋅0.6=0.2667P(Spam|X) \propto P(offer=1|Spam) \cdot P(free=1|Spam) \cdot P(Spam) \\ = \frac{2}{3} \cdot \frac{2}{3} \cdot 0.6 = 0.2667

**For Ham:**

P(Ham∣X)∝P(offer=1∣Ham)⋅P(free=1∣Ham)⋅P(Ham)=12⋅12⋅0.4=0.1P(Ham|X) \propto P(offer=1|Ham) \cdot P(free=1|Ham) \cdot P(Ham) \\ = \frac{1}{2} \cdot \frac{1}{2} \cdot 0.4 = 0.1

**✅ Step 4: Compare Posterior Probabilities**

* Spam: 0.2667
* Ham: 0.1

🔚 **Prediction = Spam**

**🐍 Python Code Example**

from sklearn.naive\_bayes import BernoulliNB

import pandas as pd

# Dataset

data = pd.DataFrame({

'offer': [1, 1, 0, 1, 0],

'free': [1, 0, 1, 1, 0],

'label': ['spam', 'spam', 'spam', 'ham', 'ham']

})

X = data[['offer', 'free']]

y = data['label']

# Model

model = BernoulliNB()

model.fit(X, y)

# Predict email with offer=1, free=1

print("Predicted class:", model.predict([[1, 1]]))

print("Class probabilities:", model.predict\_proba([[1, 1]]))

**📊 Diagram: Naïve Bayes Flow**

+-------------+

Input ---> | Extract |

| Features |

+-------------+

|

V

+------------------+

| Compute P(Class) |

+------------------+

|

V

+--------------------------+

| Compute P(features|Class)|

+--------------------------+

|

V

+-------------------------+

| Apply Bayes' Theorem |

+-------------------------+

|

V

+--------------+

| Choose Class |

+--------------+

**✍️ Summary Theory (Use in Exam)**

* Naive Bayes is based on **Bayes Theorem** and **feature independence**
* It is widely used for **text classification**
* It is simple, fast, and works well with large datasets
* Three types: Gaussian, Multinomial, Bernoulli
* Posterior probability is calculated using:

P(C∣X)∝P(x1∣C)⋅P(x2∣C)⋅⋯⋅P(C)P(C|X) \propto P(x\_1|C) \cdot P(x\_2|C) \cdot \dots \cdot P(C)

* Choose the class with **highest posterior**
* Handles both **discrete and continuous** data

Let’s solve this Naïve Bayes **numerical step-by-step** for the given **exam-style question**:

**📘 Question Summary:**

You are given a training dataset:

| **Email** | **Offer** | **Free** | **Spam** |
| --- | --- | --- | --- |
| 1 | 1 | 0 | No |
| 2 | 0 | 1 | Yes |
| 3 | 1 | 1 | Yes |
| 4 | 0 | 1 | No |
| 5 | 1 | 1 | Yes |

🟢 New input to classify: offer = 1, free = 1

**✅ Step 1: Prepare Class Counts**

Total emails = 5

* **Spam (Yes)** = 3 emails → Emails 2, 3, 5
* **Not Spam (No)** = 2 emails → Emails 1, 4

So:

P(Spam=Yes)=35,P(Spam=No)=25P(Spam=Yes) = \frac{3}{5}, \quad P(Spam=No) = \frac{2}{5}

**✅ Step 2: Likelihoods**

We now compute conditional probabilities for both classes.

**🔷 For Spam = Yes (3 emails: 2, 3, 5):**

| **Feature** | **Value = 1** | **Count** | **Likelihood** |
| --- | --- | --- | --- |
| Offer=1 | Emails 3, 5 → 2/3 | (P(Offer=1 | Spam=Yes) = \frac{2}{3}) |
| Free=1 | Emails 2, 3, 5 → 3/3 | (P(Free=1 | Spam=Yes) = \frac{3}{3} = 1) |

**🔷 For Spam = No (2 emails: 1, 4):**

| **Feature** | **Value = 1** | **Count** | **Likelihood** |
| --- | --- | --- | --- |
| Offer=1 | Email 1 → 1/2 | (P(Offer=1 | Spam=No) = \frac{1}{2}) |
| Free=1 | Email 4 → 1/2 | (P(Free=1 | Spam=No) = \frac{1}{2}) |

**✅ Step 3: Naïve Bayes Formula (Unnormalized Posteriors)**

Use:

P(Class∣X)∝P(Class)⋅P(Offer=1∣Class)⋅P(Free=1∣Class)P(Class|X) \propto P(Class) \cdot P(Offer=1|Class) \cdot P(Free=1|Class)

**🔹 For Spam = Yes:**

P(Yes∣X)∝35⋅23⋅1=615=0.4P(Yes|X) \propto \frac{3}{5} \cdot \frac{2}{3} \cdot 1 = \frac{6}{15} = 0.4

**🔹 For Spam = No:**

P(No∣X)∝25⋅12⋅12=220=0.1P(No|X) \propto \frac{2}{5} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{2}{20} = 0.1

**✅ Step 4: Compare Posteriors**

| **Class** | **Probability** |
| --- | --- |
| Spam=Yes | 0.4 |
| Spam=No | 0.1 |

**🟢 Final Prediction:**

Since **0.4 > 0.1**, the new email is classified as **Spam**.

**📌 Final Answer:**

The new email with offer=1 and free=1 is **classified as Spam** using Naive Bayes.

**🧪 Optional: Python Code (for verification)**

from sklearn.naive\_bayes import BernoulliNB

import numpy as np

# Feature matrix (Offer, Free)

X = np.array([

[1, 0], # No

[0, 1], # Yes

[1, 1], # Yes

[0, 1], # No

[1, 1], # Yes

])

y = np.array(['No', 'Yes', 'Yes', 'No', 'Yes'])

# Train model

model = BernoulliNB()

model.fit(X, y)

# Predict for [Offer=1, Free=1]

pred = model.predict([[1, 1]])

print("Predicted class:", pred[0])

Would you like the **Laplace Smoothing** version or a **similar practice numerical** now?