Question 1: Explain the differences between AI, ML, Deep Learning (DL), and Data Science (DS).

1. **Artificial Intelligence (AI):** Broad field aiming to create systems that perform tasks that normally require human intelligence (planning, perception, reasoning, language). AI includes rule-based systems, search, optimization, and machine learning.
2. **Machine Learning (ML):** A subset of AI where systems learn patterns from data to make predictions/decisions without being explicitly programmed for every rule. Examples: classification, regression, clustering.
3. **Deep Learning (DL):** A subset of ML using multi-layer neural networks (deep neural networks). Particularly effective on large datasets and tasks like image recognition, speech, and NLP. It automatically learns hierarchical features.
4. **Data Science (DS):** Interdisciplinary field that extracts insights from data using statistics, ML, data engineering, visualization, and domain knowledge. DS uses ML/DL as tools, but also covers data cleaning, EDA, storytelling, and deployment.

Question 2: What are the types of machine learning? Describe each with one real-world example.

1. **Supervised Learning:** Learn mapping from inputs to known outputs (labels).
   - Example: Email spam detection using labeled spam/ham emails.
2. **Unsupervised Learning:** Find structure/patterns in unlabeled data.
   - Example: Customer segmentation using clustering (k-means) on purchase behavior.
3. **Semi-supervised Learning:** Uses a small amount of labeled data + large unlabeled data to improve performance.
   - Example: Web image classification where only a few images are labeled; use unlabeled images to improve model.
4. **Reinforcement Learning:** Agent learns by interacting with an environment and receiving rewards.
   - Example: Training a recommendation agent that shows items and learns from user clicks/rewards (or game-playing agents).
5. **Self-supervised Learning:** A form of unsupervised learning where the data provides its own supervision (e.g., predicting masked parts).
   - Example: Pretraining language models (BERT) by predicting masked words, then fine-tuning on downstream tasks.

Question 3: Define overfitting, underfitting, and the bias-variance tradeoff in machine learning.

Answer:

- **Overfitting:** Model learns noise and fine-grained patterns of the training set that don't generalize to new data. Symptoms: very low training error, high validation/test error. Causes: overly complex model, too many parameters, insufficient data.
- **Underfitting:** Model is too simple to capture underlying patterns. Symptoms: high training and validation error. Causes: too-simple model, insufficient features, too much regularization.
- **Bias–Variance tradeoff:**
    - Bias = error from erroneous assumptions / underfitting (model too simple).
    - Variance = error from sensitivity to small fluctuations in training set / overfitting (model too complex).
    - We choose model complexity (and regularization) to balance bias and variance to minimize total expected error.

Question 4: What are outliers in a dataset, and list three common techniques for handling them.

Answer:

1.Outliers: Data points that differ significantly from other observations; they may result from measurement error, data-entry error, or true rare events.

2.**Common techniques to handle outliers:**

1. **Remove / discard** outliers (only if justified — e.g., measurement error).
2. **Cap / winsorize**: replace extreme values with a percentile (e.g., 1st/99th percentiles) to limit influence.
3. **Transform** data (log, Box–Cox) to reduce skew and lessen outlier influence.

Question 5: Explain the process of handling missing values and mention one imputation technique for numerical and one for categorical data.

Answer:

1.Process (general):

1. **Detect** missing values and quantify missingness pattern (MCAR / MAR / MNAR).
2. **Explore** — check if missingness correlates with other features/target.
3. **Decide** strategy per variable: drop rows, drop column, or impute.
4. **Impute** or model missingness appropriately; when imputing, consider uncertainty (e.g., multiple imputation).
5. **Validate**: check downstream effect on model and test performance.

2.One imputation technique for numerical: Mean imputation (replace missing numeric values with column mean) — simple, fast, but reduces variance and can bias distributions. Better alternatives: median (robust) or KNN / model-based imputation / multiple imputation.

3.One imputation technique for categorical: Mode (most frequent) imputation (replace missing categories with most frequent category). Alternatives: treat "Missing" as its own category, or use model-based prediction.

Question 6: Write a Python program that:

- **Creates a synthetic imbalanced dataset with make_classification() from sklearn.datasets.**
- **Prints the class distribution. (Include your Python code and output in the code box below.)**

Answer:

Input:

```
from sklearn.datasets import make_classification
from collections import Counter

# Generate imbalanced dataset
X, y = make_classification(n_samples=1000, n_features=5, n_informative=2,
                    n_redundant=0, n_clusters_per_class=1,
                    weights=[0.92, 0.08], random_state=42)

# Class distribution
dist = Counter(y)
print("Class distribution:", dist)
```

Output:

```
Class distribution: Counter({np.int64(0): 917, np.int64(1): 83})
```

Question 7: Implement one-hot encoding using pandas for the following list of colors: ['Red', 'Green', 'Blue', 'Green', 'Red']. Print the resulting dataframe. (Include your Python code and output in the code box below.)

```
import pandas as pd

# Original list
colors = ['Red', 'Green', 'Blue', 'Green', 'Red']

# Convert into DataFrame
df = pd.DataFrame({'color': colors})

# One-hot encode
df_encoded = pd.get_dummies(df, columns=['color'])
print(df_encoded)
```

Output:

```
   color_Blue  color_Green  color_Red
0     False       False      True
1     False        True     False
```

```
2     True      False     False
3     False      True     False
4     False     False      True
```

Question 8: Write a Python script to:

● **Generate 1000 samples from a normal distribution.**

● **Introduce 50 random missing values.**

● **Fill missing values with the column mean.**

● **Plot a histogram before and after imputation.**

(Include your Python code and output in the code box below.)

Answer:

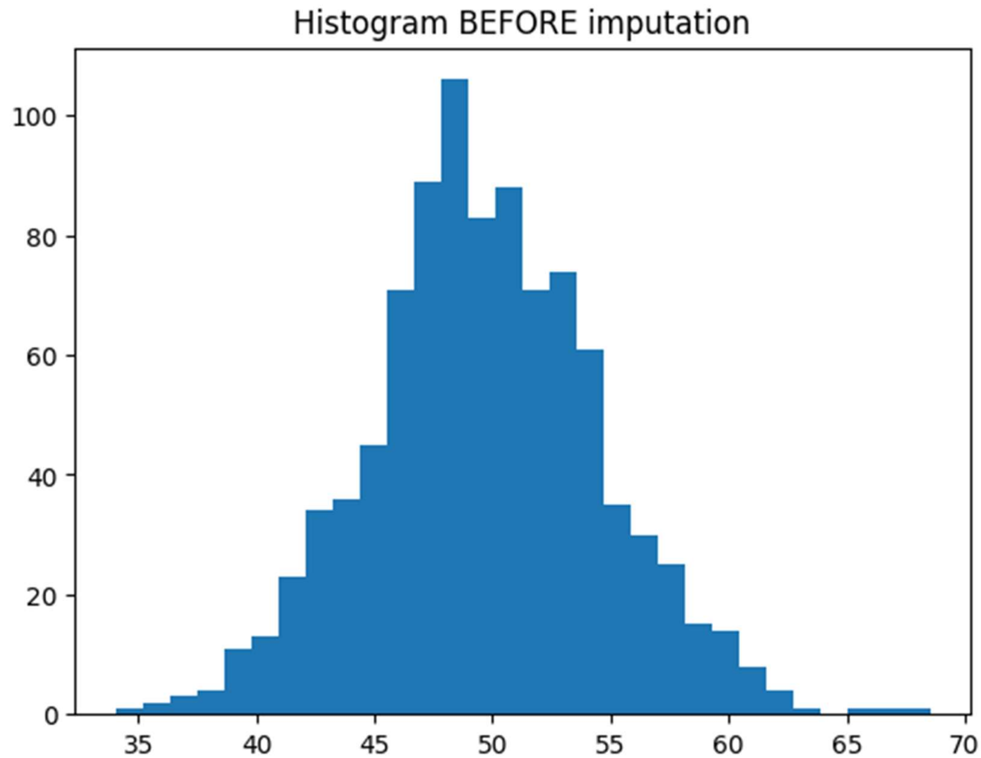
Input:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer

# Generate normal samples
data = np.random.normal(loc=50, scale=5, size=1000)

# Introduce 50 missing values
data_with_nan = data.copy()
missing_indices = np.random.choice(len(data), size=50, replace=False)
data_with_nan[missing_indices] = np.nan

print("Missing count BEFORE imputation:", np.isnan(data_with_nan).sum())

# Impute missing values with mean
imputer = SimpleImputer(strategy='mean')
data_imputed = imputer.fit_transform(data_with_nan.reshape(-1,1)).ravel()

print("Missing count AFTER imputation:", np.isnan(data_imputed).sum())

# Plot histograms
plt.hist(data_with_nan[~np.isnan(data_with_nan)], bins=30)
plt.title("Histogram BEFORE imputation")
plt.show()

plt.hist(data_imputed, bins=30)
plt.title("Histogram AFTER imputation")
plt.show()
```

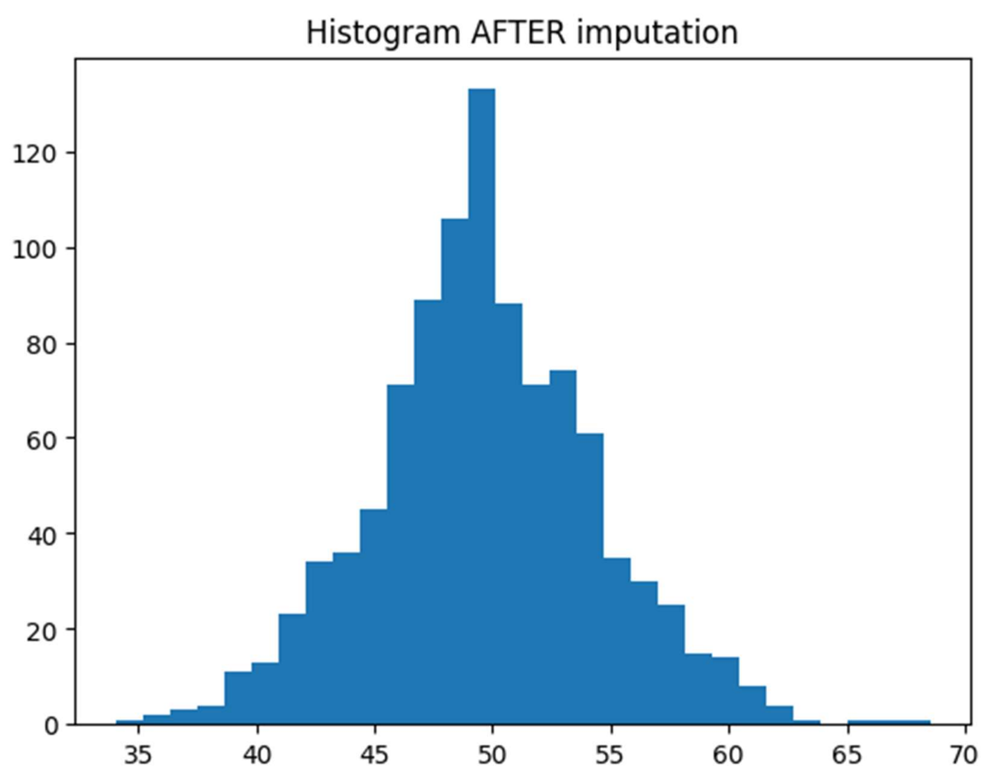
Output:

```
Missing count BEFORE imputation: 50
Missing count AFTER imputation: 0
```

Histogram before imputation



Histogram after imputation



Question 9: Implement Min-Max scaling on the following list of numbers [2, 5, 10, 15, 20] using sklearn.preprocessing.MinMaxScaler. Print the scaled array. (Include your Python code and output in the code box below.)

Answer:

Input:

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Values
values = np.array([2, 5, 10, 15, 20]).reshape(-1, 1)

# Min-Max scaling
scaler = MinMaxScaler()
scaled = scaler.fit_transform(values)

print("Original values:", values.ravel())
print("Scaled values:", scaled.ravel())
```

Output:

```
Original values: [ 2  5 10 15 20]
Scaled values: [0.         0.16666667 0.44444444 0.72222222 1.        ]
```

Question 10: You are working as a data scientist for a retail company. You receive a customer transaction dataset that contains:

- **Missing ages,**
- **Outliers in transaction amount.,**
- **A highly imbalanced target (fraud vs. non-fraud),**
- **Categorical variables like payment method.**

Explain the step-by-step data preparation plan you'd follow before training a machine learning model. Include how you'd address missing data, outliers, imbalance, and encoding. (Include your Python code and output in the code box below.)

Answer:(theory)

Step-by-step Data Preparation Plan

1. Handle Missing Ages
    - Check missing values in the age column.
    - Fill missing values using:
        - Median (robust against outliers).
        - Or predictive imputation using regression/KNN if age is important.
2. Handle Outliers in Transaction Amount
    - Detect outliers using IQR (Interquartile Range) or Z-score.
    - Options:
        - Cap extreme values (Winsorization).
        - Apply log transformation.
3. Handle Imbalanced Target (fraud vs. non-fraud)
    - Fraud detection datasets are often highly imbalanced.
    - Approaches:
        - SMOTE/ADASYN: Oversample minority class.
        - Undersampling: Reduce majority class.
        - Class weights: Penalize misclassification of minority class.
4. Encode Categorical Variables (Payment Method)
    - Use One-Hot Encoding if low cardinality.
    - Use Target Encoding / Frequency Encoding if high cardinality.
5. Final Preparation
    - Scale numerical features (age, transaction_amount) using StandardScaler/MinMaxScaler.
    - Split data into train-test sets.

Theory:

Step-by-step Data Preparation Plan

1. **Handle Missing Ages**
   - **Check missing values in the age column.**
   - **Fill missing values using:**
     - **Median (robust against outliers).**
     - **Or predictive imputation using regression/KNN if age is important.**
2. **Handle Outliers in Transaction Amount**
   - **Detect outliers using IQR (Interquartile Range) or Z-score.**
   - **Options:**
     - **Cap extreme values (Winsorization).**
     - **Apply log transformation.**
3. **Handle Imbalanced Target (fraud vs. non-fraud)**
   - **Fraud detection datasets are often highly imbalanced.**
   - **Approaches:**
     - **SMOTE/ADASYN: Oversample minority class.**
     - **Undersampling: Reduce majority class.**
     - **Class weights: Penalize misclassification of minority class.**
4. **Encode Categorical Variables (Payment Method)**
   - **Use One-Hot Encoding if low cardinality.**
   - **Use Target Encoding / Frequency Encoding if high cardinality.**
5. **Final Preparation**
   - **Scale numerical features (age, transaction_amount) using StandardScaler/MinMaxScaler.**
   - **Split data into train-test sets.**

Input:

```
# Input
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

```
from imblearn.over_sampling import SMOTE

# Example dataset
data = {
   'age': [25, np.nan, 40, 35, np.nan, 50],
   'transaction_amount': [200, 5000, 300, 250, 10000, 400],
   'payment_method': ['card', 'cash', 'upi', 'card', 'cash', 'upi'],
   'fraud': [0, 1, 0, 0, 1, 0]
}

df = pd.DataFrame(data)

# 1. Handle Missing Ages (Median Imputation)
imputer = SimpleImputer(strategy='median')
df['age'] = imputer.fit_transform(df[['age']])

# 2. Handle Outliers (Cap using IQR)
Q1 = df['transaction_amount'].quantile(0.25)
Q3 = df['transaction_amount'].quantile(0.75)
IQR = Q3 - Q1
lower_bound, upper_bound = Q1 - 1.5*IQR, Q3 + 1.5*IQR
df['transaction_amount'] = np.where(df['transaction_amount'] > upper_bound,
upper_bound,
                        np.where(df['transaction_amount'] < lower_bound, lower_bound,
                             df['transaction_amount']))

# 3. Encode Categorical Variables
encoder = OneHotEncoder(drop='first', sparse=False)
encoded = encoder.fit_transform(df[['payment_method']])
encoded_df = pd.DataFrame(encoded,
columns=encoder.get_feature_names_out(['payment_method']))
df = pd.concat([df.drop('payment_method', axis=1), encoded_df], axis=1)

# 4. Handle Imbalance (SMOTE)
X = df.drop('fraud', axis=1)
y = df['fraud']

smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)

# 5. Scale Features
scaler = StandardScaler()
X_res_scaled = scaler.fit_transform(X_res)

# Output
print("Processed Dataset (before SMOTE):")
print(df)
print("\nResampled target distribution (after SMOTE):")
print(y_res.value_counts())
```

Before SMOTE (original dataset):

	age	transaction_amount	fraud	payment_method_cash	payment_method_upi
0	25.0	200.0	0	1	0
1	37.5	2875.0	1	0	0
2	40.0	300.0	0	0	1
3	35.0	250.0	0	1	0
4	37.5	2875.0	1	0	0
5	50.0	400.0	0	0	1

After SMOTE (balanced target):

0 6

1 6

Name: fraud, dtype: int64