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

->**Artificial Intelligence (AI)**

AI is the broadest field. It is the science of creating machines that can **simulate human intelligence** to perform tasks like reasoning, problem-solving, and decision-making. AI can be simple, rule-based systems or complex, learning-based systems.

Machine Learning (ML)

ML is a **subset of AI**. It focuses on the ability of machines to **learn from data** and improve their performance on a specific task without being explicitly programmed for every scenario. Instead of being given rigid rules, an ML model is trained on data to find patterns and make predictions.

Deep Learning (DL)

DL is a **subset of ML**. It uses multi-layered neural networks, inspired by the human brain, to **learn complex patterns** from large amounts of data. The "deep" refers to having many layers, which allows the model to automatically extract features from raw data, unlike traditional ML that often requires manual feature engineering.

Data Science (DS)

DS is a multidisciplinary field that uses scientific methods, processes, and algorithms to **extract knowledge and insights from data**. It is a complete process that often includes aspects of statistics, data visualization, and ML to analyze data and solve problems. You can think of it as the overarching practice of using data to gain insights, and it often employs ML and DL as tools.

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

->There are three main types of machine learning: supervised, unsupervised, and reinforcement learning. A fourth type, semi-supervised learning, is a hybrid of the first two.

1. Supervised Learning

In supervised learning, the model is trained on a **labeled dataset**, which means each input has a corresponding correct output. The goal is for the model to learn the mapping from input to output so it can make accurate predictions on new, unseen data. Think of

it like a student with a teacher, where the teacher provides the correct answers during training.

Example:

- **Spam detection**: An email is labeled as either "spam" or "not spam." The model learns to identify patterns in the content and headers of emails to classify new emails correctly.

2. Unsupervised Learning

Unsupervised learning uses an **unlabeled dataset**, where there are no correct outputs. The model's task is to find hidden patterns, structures, or relationships within the data on its own. This is like a student learning by observation without a teacher.

Example:

- **Customer segmentation**: A retail store uses customer purchase data (without any pre-defined groups) to cluster customers with similar buying habits. This helps them create targeted marketing campaigns.

3. Reinforcement Learning

Reinforcement learning involves an **agent** learning to make decisions by interacting with an environment. The agent receives **rewards** for good actions and **penalties** for bad ones. Its goal is to maximize the cumulative reward over time through trial and error.

Example:

- **Training a robotic arm**: A robotic arm learns to pick up an object by trying different movements. It receives a positive reward when it successfully grabs the object and a negative penalty if it fails or drops it.

4. Semi-Supervised Learning

Semi-supervised learning is a combination of supervised and unsupervised learning. It's used when you have a **small amount of labeled data** and a **large amount of unlabeled data**. The model uses the labeled data to start its learning, then leverages the unlabeled data to improve its performance and find deeper patterns.

Example:

- **Web page classification**: A search engine has a few web pages manually labeled by category (e.g., "sports," "news"), but billions of unlabeled pages. The model can use the small labeled set to begin learning and then use the structure and content of the massive unlabeled set to improve its classification accuracy.

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

->**Overfitting**

Overfitting occurs when a model learns the training data and its noise too well. It's like a student who has memorized every answer from a practice test but doesn't understand the underlying concepts. As a result, the model performs exceptionally well on the training data but fails to generalize and makes poor predictions on new, unseen data.

- **Example:** A model trained to classify dog breeds uses a very complex decision boundary to perfectly separate every training image. It might memorize that every image with a leash is a dog, but then fail when it sees a new dog image without a leash.

Underfitting

Underfitting is the opposite problem. It happens when a model is too simple to capture the underlying patterns in the training data. The model hasn't learned enough from the data to make accurate predictions. This is like a student who hasn't studied at all and performs poorly on both the practice test and the real exam. An underfit model has a high error rate on both the training and new data.

- **Example:** Using a straight line to fit data that clearly follows a curved, non-linear pattern. The simple linear model can't capture the complexity of the data, so it performs poorly.

Bias-Variance Tradeoff

The bias-variance tradeoff is a core concept that explains the relationship between model complexity, overfitting, and underfitting. It states that as you try to decrease one type of error, you often increase the other.

- **Bias** is the error from overly simple assumptions in a model. A **high-bias** model is an **underfitted** model. It fails to capture the true patterns in the data and has high error on both training and test sets.
- **Variance** is the error from a model's sensitivity to small fluctuations in the training data. A **high-variance** model is an **overfitted** model. It performs well on the training data but has a high error rate on new data because it has learned noise instead of the signal.

The goal is to find the "sweet spot" of model complexity, where you achieve a balance between bias and variance, minimizing the total error on unseen data. You want a model that is complex enough to capture the true patterns (low bias) but not so complex that it memorizes the noise (low variance).

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

->Outliers are data points that are significantly different from the rest of the observations in a dataset. They can be due to natural variability, errors in data entry, or measurement issues. Outliers can heavily influence statistical analyses and the performance of machine learning models, often skewing results and leading to inaccurate conclusions.

Techniques for Handling Outliers

1. **Removal:** The simplest method is to remove the outlier data points from the dataset. This is a good approach when you're confident that the outliers are the result of data entry errors or are not relevant to your analysis, especially if you have a large dataset where a few removed points won't impact the overall model.
2. **Transformation:** You can apply mathematical transformations to the data to reduce the impact of the outliers. Common transformations include using a **logarithmic, square root, or reciprocal function**. This "squeezes" the extreme values, bringing them closer to the rest of the data and making the distribution more symmetrical. This approach is useful when the outliers are genuine, but you need to minimize their influence on the model.
3. **Winsorization/Capping:** This method involves replacing the outlier values with a less extreme value from the dataset. For example, you could set an upper and lower limit (e.g., the 95th and 5th percentiles) and replace any values outside this range with the values at those limits. This is less drastic than removing the data entirely and helps to preserve the overall size of the dataset while reducing the impact of the outliers.

The video "The A to Z of dealing with Outliers" gives a comprehensive overview of how to detect and handle outliers in a dataset.

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

->Handling missing values is a crucial step in data preprocessing. The process involves first identifying the missing data and then deciding on an appropriate method to either remove it or, more commonly, replace it.

The Process

The general process goes like this:

1. **Identify Missing Values**: You first need to find out which features (columns) and how many data points are missing. This is often done by checking for NaN (Not a Number), None, or other specific placeholder values.

2. **Determine the Cause**: It's helpful to understand why the data is missing. Is it a data entry error, a technical issue, or does the missing value have a meaning in itself (e.g., a "not applicable" value)? This can help you choose the right strategy.
3. **Choose a Strategy**: You have a couple of options.
    - **Deletion**: You can simply remove rows or columns with missing data. This is suitable if a very small percentage of the data is missing and removing it won't cause a significant loss of information or bias.
    - **Imputation**: This is the most common approach. It involves replacing the missing values with a substitute. The choice of substitute depends on the data type and the distribution of the data.

Imputation Techniques

For Numerical Data: Median Imputation

Median imputation is a robust technique for numerical data. It involves replacing missing values in a column with the **median** of the non-missing values in that same column. The median is the middle value in a sorted list of numbers and is less sensitive to **outliers** than the mean. This makes it a great choice when the data is skewed or contains extreme values that could distort the average.

Example: If you have a column of housing prices with some missing values and a few very expensive outlier houses, using the median price would be a better representation of the typical value than using the mean, which would be inflated by the outliers.

For Categorical Data: Mode Imputation

Mode imputation, also known as most frequent value imputation, is a standard technique for categorical data. It involves replacing missing values with the **mode**—the most frequently occurring category—in that column.

Example: In a survey dataset, if a column for "Favorite Color" has missing values, you would fill those missing spots with the color that was most commonly chosen by other respondents (e.g., 'Blue'). This helps maintain the overall distribution of the categorical data.

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.)

->

```
from sklearn.datasets import make_classification

from collections import Counter
```

```
# Create a synthetic imbalanced dataset
X, y = make_classification(n_classes=2, class_sep=2,
                           weights=[0.9, 0.1], n_informative=3,
                           n_redundant=1, flip_y=0,
                           n_features=5, n_clusters_per_class=1,
                           n_samples=1000, random_state=42)

# Print the class distribution
class_distribution = Counter(y)
print("Class distribution:", class_distribution)
```

Output:

Class distribution: Counter({0: 900, 1: 100})

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
# List of colors
colors = ['Red', 'Green', 'Blue', 'Green', 'Red']
# Create a DataFrame
df = pd.DataFrame({'Color': colors})
# Apply one-hot encoding
one_hot_encoded_df = pd.get_dummies(df, columns=['Color'])
# Print the resulting DataFrame
print(one_hot_encoded_df)
```

Output:

Color_Blue	Color_Green	Color_Red
0	0	1
1	0	1	0
2	1	0	0
3	0	1	0
4	0	0	1

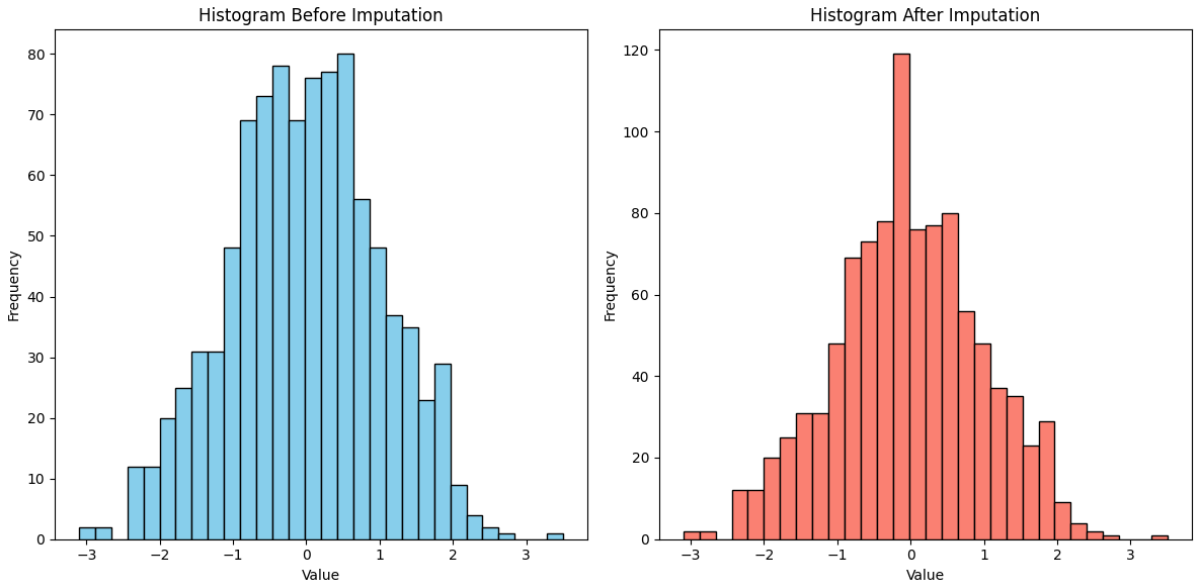
8)Write a Python script to: ● Generate 1000 samples from a normal distribution. ● Introduce 50 random missing values. ● Fill missing values with the column mean. ● Plot a histogram before and after imputation. (Include your Python code and output in the code box below.)

->

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt


# Step 1: Generate 1000 samples from a normal distribution

data = np.random.normal(loc=0, scale=1, size=1000)


# Step 2: Introduce 50 random missing values

data_with_nan = data.copy()

missing_indices = np.random.choice(len(data_with_nan), size=50, replace=False)

data_with_nan[missing_indices] = np.nan


# Step 3: Fill missing values with the column mean

mean_value = np.nanmean(data_with_nan)

data_imputed = np.where(np.isnan(data_with_nan), mean_value, data_with_nan)
```

```
# Step 4: Plot histograms before and after imputation
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(data_with_nan[~np.isnan(data_with_nan)], bins=30, color='skyblue', edgecolor='black')
plt.title('Histogram Before Imputation')
plt.xlabel('Value')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
plt.hist(data_imputed, bins=30, color='salmon', edgecolor='black')
plt.title('Histogram After Imputation')
plt.xlabel('Value')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

Output:



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.)

->

```
from sklearn.preprocessing import MinMaxScaler

import numpy as np

# Original data

data = np.array([[2], [5], [10], [15], [20]])

# Initialize the MinMaxScaler

scaler = MinMaxScaler()

# Fit and transform the data

scaled_data = scaler.fit_transform(data)

# Print the scaled array

print("Scaled array:")

print(scaled_data)
```

Output:

Scaled array:

[[0.   ]

 [0.16666667]

 [0.44444444]

 [0.72222222]

 [1.   ]]

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.)

->

Step-by-Step Data Preparation Plan

1. Handle Missing Ages

Use mean imputation to fill missing values in the Age column.

2. Treat Outliers in Transaction Amount

Apply the IQR method to cap extreme values beyond the upper and lower bounds.

3. Encode Categorical Variables

Use OneHotEncoder from sklearn.preprocessing (with sparse_output=False for newer versions).

4. Handle Imbalanced Target

Use upsampling to balance the IsFraud classes.

5. Scale Numerical Features

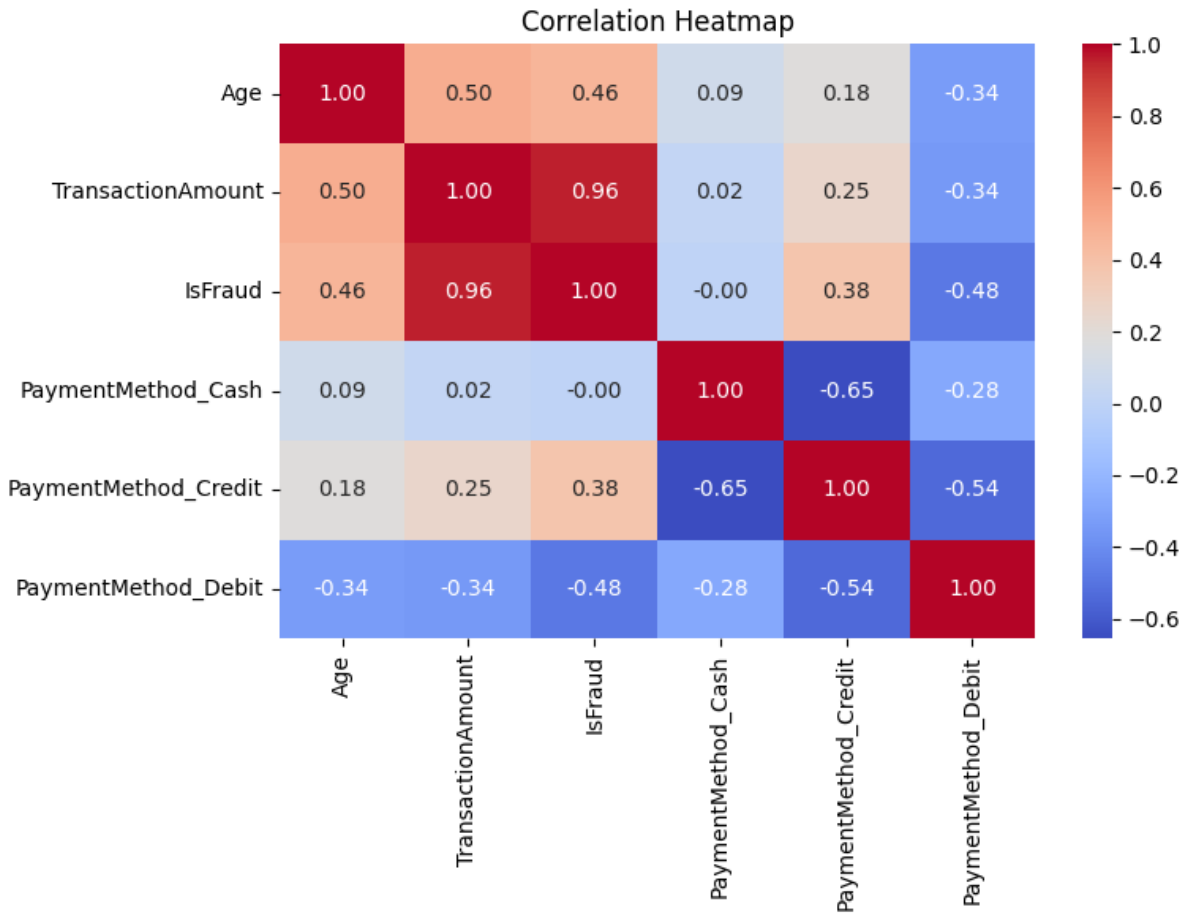
Apply Min-Max Scaling to normalize Age and TransactionAmount.

Code:

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.utils import resample
# Simulate dataset
data = {
   'Age': [25, np.nan, 35, 45, np.nan, 30, 40, 50, np.nan, 60],
   'TransactionAmount': [100, 200, 150, 100000, 250, 300, 120, 50000, 180, 220],
   'PaymentMethod': ['Credit', 'Debit', 'Cash', 'Credit', 'Cash', 'Debit', 'Credit', 'Cash',
'Debit', 'Credit'],
   'IsFraud': [0, 0, 0, 1, 0, 0, 0, 1, 0, 0]
}
df = pd.DataFrame(data)
# Step 1: Impute missing ages
imputer = SimpleImputer(strategy='mean')
df['Age'] = imputer.fit_transform(df[['Age']])
# Step 2: Treat outliers in TransactionAmount
Q1 = df['TransactionAmount'].quantile(0.25)
Q3 = df['TransactionAmount'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df['TransactionAmount'] = np.where(df['TransactionAmount'] > upper_bound,
upper_bound,
```

```
                    np.where(df['TransactionAmount'] < lower_bound, lower_bound,
df['TransactionAmount']))

# Step 3: One-hot encode PaymentMethod

encoder = OneHotEncoder(sparse_output=False)

encoded = encoder.fit_transform(df[['PaymentMethod']])

encoded_df = pd.DataFrame(encoded,
columns=encoder.get_feature_names_out(['PaymentMethod']))

df = pd.concat([df.drop('PaymentMethod', axis=1), encoded_df], axis=1)

# Step 4: Handle class imbalance

df_majority = df[df.IsFraud == 0]

df_minority = df[df.IsFraud == 1]

df_minority_upsampled = resample(df_minority, replace=True,
n_samples=len(df_majority), random_state=42)

df_balanced = pd.concat([df_majority, df_minority_upsampled])

# Step 5: Scale numerical features

scaler = MinMaxScaler()

df_balanced[['Age', 'TransactionAmount']] = scaler.fit_transform(df_balanced[['Age',
'TransactionAmount']])

# Final output

print("Prepared DataFrame:")

print(df_balanced)
```

Output:


Correlation Heatmap
Age
TransactionAmount
IsFraud
PaymentMethod_Cash
PaymentMethod_Credit
PaymentMethod_Debit
1.00 0.50 0.46 0.09 0.18 -0.34
0.50 1.00 0.96 0.02 0.25 -0.34
0.46 0.96 1.00 -0.00 0.38 -0.48
0.09 0.02 -0.00 1.00 -0.65 -0.28
0.18 0.25 0.38 -0.65 1.00 -0.54
-0.34 -0.34 -0.48 -0.28 -0.54 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6