Artificial Intelligence

Course code : CS451

Lobna Abd Alaziz

Data and Preprocessing

What Is Data Preprocessing for Machine Learning?

Data preprocessing in machine learning is the critical initial process of transforming raw data into a clean, structured, and usable format that a machine learning algorithm can understand and learn from

Why Preprocessing is Required?

- Preprocessing refers to the transformations applied to data before feeding it into a machine learning model.
- The quality of data is crucial for effective training.
- It's important to note that data is rarely in the format needed for analysis.
- Data cleaning ensures that the dataset is free from errors, missing values, or inconsistencies.

Missing Value

- Missing values occur when there is no data or value stored for the variable in an observation
- Missing data are a common occurrence and can have a significant effect on the conclusions drawn from the data
- Most statistical procedures require a value for each variable

Imputing the Missing Values

- Statistical technique such as imputing the missing values with **average** values
- Negative values such as **-1** or **-99**

Imputing Missing Values using Mean, Median and Mode

❑ Mode

- Having some missing values in a categorical variable called Gender
- Two types of values: **Males** and **Females**
- To impute or replace the missing values from such a column we can use the **Mode** function
- It returns the **maximum occurring** value, which in turn can be use to replace the missing values

Imputing Missing Values using Mean, Median and Mode

❑ Median

- We should impute the missing values in numerical variables using the **Median** function
- If there are **outliers** present in the data
- As the median function is not sensitive towards outliers

Imputing Missing Values using Mean, Median and Mode

❑ Mean

- We should use the **Mean** function when the data does not contain any outliers
- **Mean** function is very sensitive towards outliers

Import the Necessary Libraries

Pandas is an open-source software library for the Python programming language, primarily used for data manipulation and analysis.

Import the **Pandas** library

```
import pandas as pd
```

How to Load the Dataset

- Start by loading your dataset into a Pandas **DataFrame**.
- We'll use a hypothetical dataset named **your_dataset.csv**.
- We will load the dataset into a variable called **df**.

```
# Reading the dataset

#Replace 'your_dataset.csv' with the actual dataset name or file path
df = pd.read_csv('your_dataset.csv')
```

Exploratory Data Analysis (EDA)

- **EDA** helps you understand the structure and characteristics of your dataset. Some Pandas functions help us gain insights into our dataset.

  For example:

  - **df.head( )**
  - **df.describe( )**
  - **df.info( )**

Exploratory Data Analysis (EDA)

df.head() will call the first **5 rows** of the dataset. You can specify the number of rows to be displayed in parentheses.

```
#Display the first few rows of the dataset
print(df.head())
```

Exploratory Data Analysis (EDA)

df.describe() gives some statistical data like percentile, mean, and standard deviation of the numerical values of the **DataFrame**.

```
#Summary statistics
print(df.describe())
```

Exploratory Data Analysis (EDA)

df.info() gives the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (**non-null values**)

```
#Information about the dataset
print(df.info())
```

How to Handle Missing Values

- **Missing values** pose a significant stress as they come in different formats and can adversely impact your analysis or model.

- **Machine learning models** cannot be trained with data that has missing values, as this can alter your end results during analysis.

How to Handle Missing Values

- One way to do this is by removing the missing values altogether.

```
#Check for missing values
print(df.isnull().sum())

#Drop rows with missing valiues and place it in a new variable "df_cleaned"
df_cleaned = df.dropna()
```

How to Handle Missing Values

- But if the number of rows that have missing values is large, then this method will be inadequate.

- For numerical data, you can simply compute the mean and input it into the rows that have missing values

```
#Replace missing values with the mean of each column
df.fillna(df.mean(), inplace=True)

#If you want to replace missing values in a specific column, you can do it this way:
#Replace 'column_name' with the actual column name
df['column_name'].fillna(df['column_name'].mean(), inplace=True)
```

How to Remove Duplicate Records

- Duplicate records can distort your analysis by influencing the results in ways that do not accurately show trends and underlying patterns.
- Pandas helps to identify and remove duplicate values.

```
#Identify duplicates
print(df.duplicated().sum())

#Remove duplicates
df_no_duplicates = df.drop_duplicates()
```

Outliers

- **An outlier** is a data point that differs substantially from the rest of the data. In other words, it's an observation that lies an abnormal distance from other values.

Why are Outliers Problematic for Building Predictive Models?

Outliers can negatively impact predictive models for several reasons:

1. Distortion of Statistical Measures: Outliers can skew the mean and standard deviation, leading to incorrect conclusions about data distribution.
2. Model Training: They may cause the model to learn incorrect patterns, resulting in biased predictions.

How Should We Handle Outliers in Our Data?

There are several methods to handle outliers, namely:

1. **Removal**, if you believe the outliers are caused by errors or are irrelevant to your analysis, then this is the right method. but you should also consider how much data to remove and how much it affects your data.

2. **Imputation** is a technique used to fill in values that are considered outliers with other values. Usually, the outlier value is replaced with the **median**, **mean**, or **mode value**.

Outlier Detection using Interquartile Range (IQR)

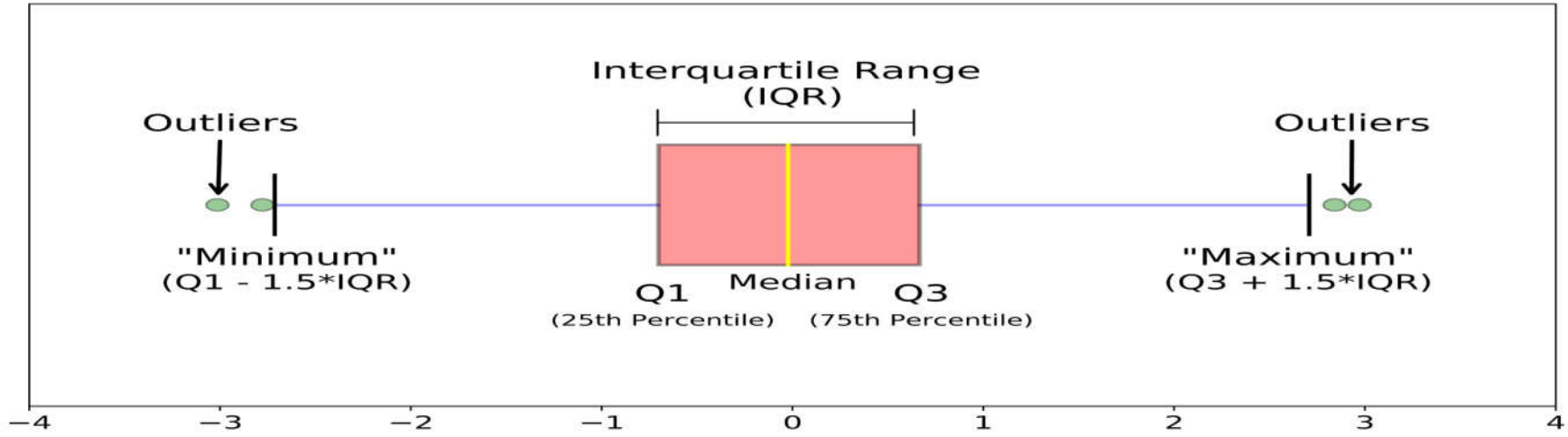
- The interquartile range (**IQR**) is a measure of statistical dispersion that is equal to the difference between the **first** and **third quartiles**.
- It is calculated by subtracting the **first** quartile from the **third quartile**

$$IQR = Q_3 - Q_1$$

Outlier Detection using Interquartile Range (IQR)

How to detect outliers now?

- All the values above **Q3 + 1.5*IQR** and the values below **Q1 – 1.5*IQR** are outliers.
- That's basically all the points outside the whiskers.



Steps to perform Outlier Detection

1. Arrange your data in ascending order
2. Find **Q2** (Median)
3. Calculate **Q1** (**the first Quarter**)
4. Calculate **Q3** (**the third Quartile**)
5. Find **IQR** = (**Q3** – **Q1**)
6. Find the **lower Range** = **Q1** -(**1.5** * **IQR**)
7. Find the **upper Range** = **Q3** + (**1.5** * **IQR**)

Question:

Given the following set of values: **10**, **12**, **15**, **16**, **18**, **20**, **22**, **25**, **50**,

calculate **the outliers** using **the Interquartile Range** (**IQR**) method.

Answer:

To find outliers using the **IQR** method manually, follow these steps:

1 - Sorted Data: [10, 12, 15, 16, 18, 20, 22, 25, 50]

2 - Q2 (Median): Entire dataset: **18** (5th value in sorted list)

3 - Calculate Q1 (25th Percentile)

- First half of the data: [10, 12, 15, 16]
- Median of the first half: $\boldsymbol{Q1} = \frac{12+15}{2} = 13.5$

Answer:

To find outliers using the **IQR** method manually, follow these steps:

4 - Calculate Q3 (75th Percentile)

- Second half of the data: [20, 22, 25, 50]
- Median of the Second half: $Q3 = \frac{22+25}{2} = 23.5$

5 - Calculate the IQR

$$IQR = Q3 - Q1 = 23.5 - 13.5 = 10$$

Answer:

To find outliers using the **IQR** method manually, follow these steps:

6 - Determine Outlier Boundaries

Lower Bound = **Q1** − 1.5 × **IQR** = 13.5 − 1.5 × 10 = 13.5 − 15 = **−1.5**

Upper Bound = **Q3** + 1.5 × **IQR** = 23.5 + 1.5 × 10 = 23.5 + 15 = **38.5**

Answer:

To find outliers using the **IQR** method manually, follow these steps:

7 - Check Each Data Point

Data points: [10, 12, 15, 16, 18, 20, 22, 25, 50]

Outlier condition:

Values below **-1.5**: **None**

Values above **38.5**: **50**

Outliers: The only outlier in this dataset is **50**.

Python Code:

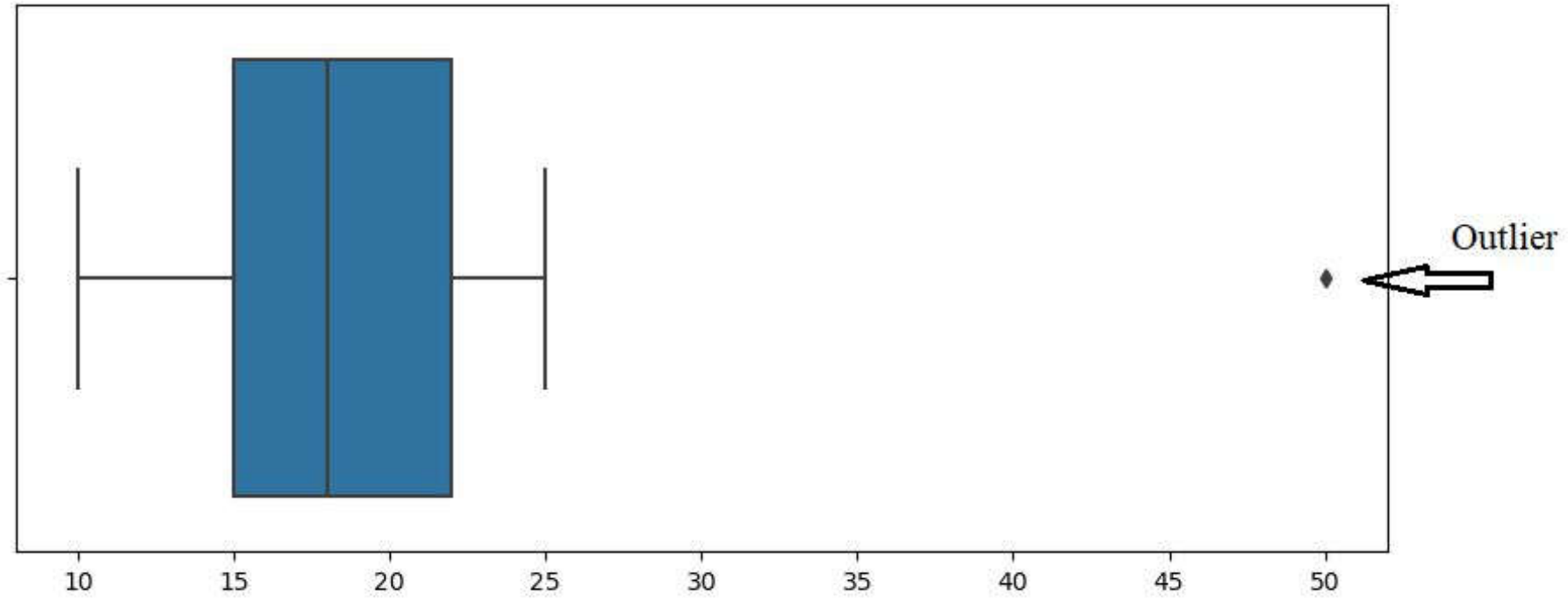
```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Sample data with a clear outlier
data = [10, 12, 15, 16, 18, 20, 22, 25, 50]

# Create a single figure
plt.figure(figsize=(10, 4))

# Boxplot
sns.boxplot(data)
```

Python Code:



Python Code:

```
# Demonstrate outlier detection calculation (optional, for understanding)
q1 = np.percentile(data, 25)
q3 = np.percentile(data, 75)
iqr = q3 - q1
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr

print('q1 = ', q1)
print('q3 = ', q3)
print('IQR = ', iqr)
print('Lower Bound = ', lower_bound)
print('Upper Bound = ', upper_bound)

# Identify outliers programmatically
outliers = [x for x in data if x < lower_bound or x > upper_bound]
print(f"Identified Outliers: {outliers}")
```

```
q1 =  15.0
q3 =  22.0
IQR =  7.0
Lower Bound =  4.5
Upper Bound =  32.5
Identified Outliers: [50]
```

Normalization

- **Normalization** is a preprocessing technique used to **scale the features** of your dataset to a common range, typically **[0, 1]** or **[-1, 1]**.
- This is important in machine learning, especially for algorithms that are **sensitive to the scale of the data**, such as **neural networks**, **k-nearest neighbors**, and **support vector machines**.

Normalization

Why Normalize the Data?

- **Improves Convergence Speed**: Normalizing can speed up the convergence of gradient descent algorithms used in training models like neural networks.
- **Prevents Dominance of Features**: If one feature has a much larger range than others, it can dominate the distance calculations in algorithms, leading to poor model performance.
- **Ensures Fairness**: Each feature contributes equally to the distance computations, making the model more robust.

Normalization

Common Normalization Techniques

1 - Min-Max Normalization: Scales the data to a fixed range, usually **[0, 1]**.

The formula is:

$$\grave{X} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Normalization

Common Normalization Techniques

2 - Z-score Normalization (Standardization): Centers the data around the mean with a unit standard deviation.

The formula is:

$$\grave{X} = \frac{X - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation

Normalization

Example of Normalization

Suppose you have a dataset with the following features:

Feature B	Feature A
200	1
300	2
400	3

Normalization

Answer: Min-Max Normalization

For Feature A:

$$\grave{X} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- Min = 1, Max = 3
- Normalized values:
  - **For 1**: (1−1) / (3−1)= 0
  - **For 2**: (2−1) / (3−1)= 0.5
  - **For 3**: (3−1) / (3−1)= 1

Feature B	Feature A
200	1
300	2
400	3

Normalization

Answer: Min-Max Normalization

For Feature A:

$$\grave{X} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- Min = 200, Max = 400
- Normalized values:
  - **For 200**: (200−200) / (400-200)= 0
  - **For 300**: (300−200) / (400-200)= 0.5
  - **For 400**: (400−200) / (400-200)= 1

Feature B	Feature A
200	1
300	2
400	3

Normalization

Answer: **Min-Max Normalization**

Feature B	Feature A
200	1
300	2
400	3

The normalized dataset will look like this:

Feature B	Feature A
0	0
0.5	0.5
1	1

Python Code:

Using **MinMaxScaler** from **sklearn** for **Min-Max normalization**

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_normalized = scaler.fit_transform(X)  # X is your feature matrix
```

Python Code:

Using **StandardScaler** from **sklearn** for **Z-score normalization**

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)  # X is your feature matrix
```