Chapter 2
Data Preprocessing

Er. Shiva Ram Dam

Assistant Professor

Gandaki University

Content:

1. Data Objects and Attribute Types
2. Basic Statistical Descriptions of Data
3. Data Preprocessing
4. Data Cleaning
5. Data Integration
6. Data Reduction: Principle Component Analysis
7. Data Transformation and Data Discretization
8. Measures of Similarity and Dissimilarity

2.1 Data Objects and Attribute Types

Data Types and Attributes

Data:

- A data **is known fact** that can be recorded and have implicit meaning
  - For example: the names, eye color, telephone numbers, and addresses of students of a class
- So data is **collection of data objects** and their **attributes.**
- A collection of attributes describe an object.

Data Object:

- Data sets are made up of data objects.
- A **data object** represents an **entity**.
- Examples:
  - *sales database:* customers, store items, sales
  - *medical database:* patients, treatments
  - *university database:* students, professors, courses
- Data objects are described by **attributes**.

Attributes

Objects

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- database rows → data objects
- database columns → attributes

Attributes

Attributes:

- An attribute is a property or **characteristics of an object.**
  - For example, *id*, *age* are attributes of a student.
- **Attribute values** are numbers or symbols assigned to an attribute.
  - Example: attribute values of *ID* and *age* are integers.
- But properties of attribute values can be different.
  - *ID* has no limit but *age* has a maximum and minimum value.
- Also Known as:
  - Features (in Machine Learning)
  - Dimensions (in Data Mining)
  - Variables (in Statistics)

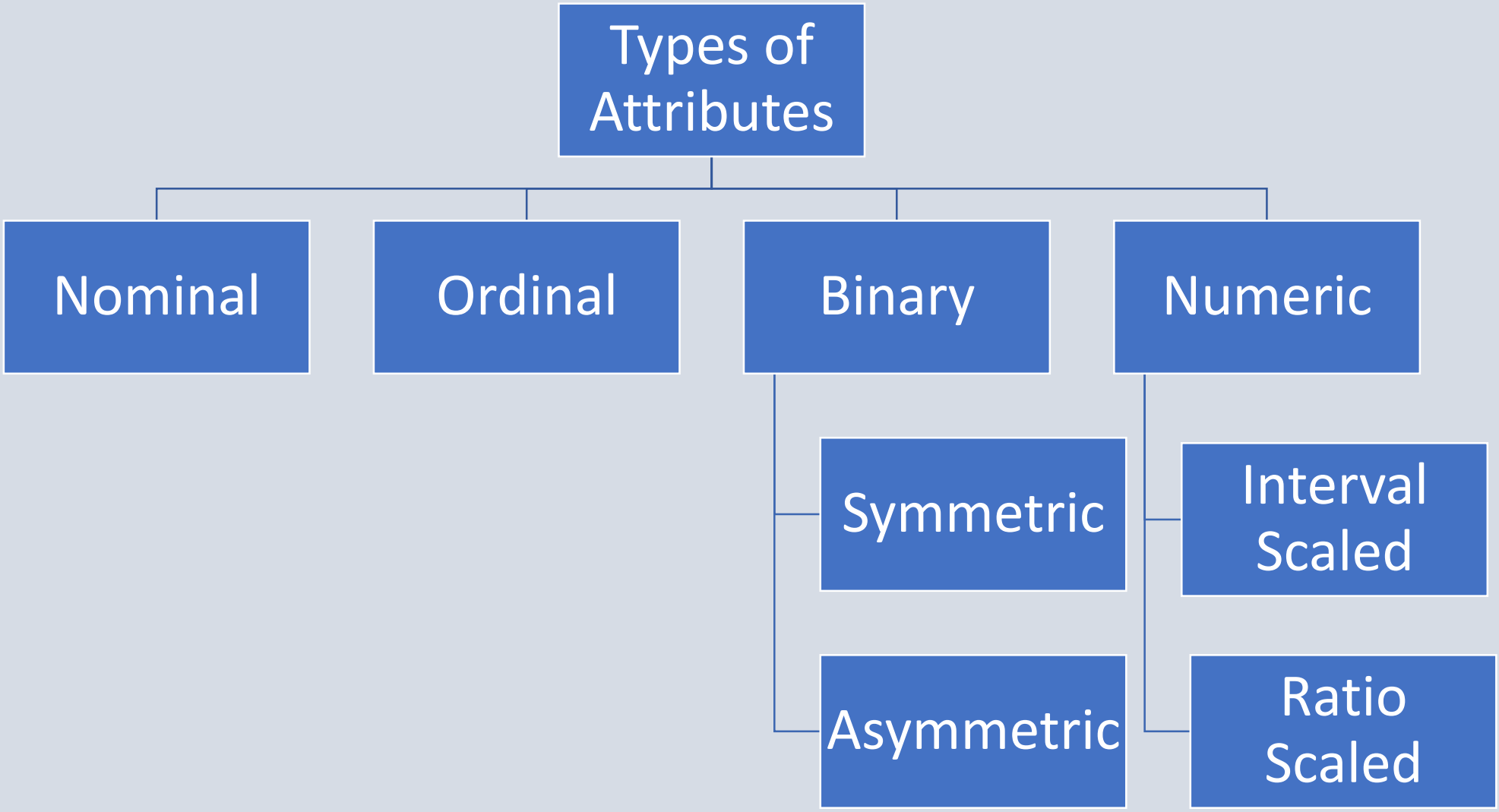
Attributes

Objects

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- database rows → data objects
- database columns → attributes

Types of Attributes



1. Nominal Attributes

- Nominal attributes are types of data variables that represent **categories or groups with no inherent order or ranking** between them.
- These attributes describe qualities or characteristics of objects that are non-numeric in nature.
- The values of nominal attributes **do not have any meaningful order.**
- Examples of nominal attributes include:
  - **Gender**: Categories might include "male," "female," or "other."
  - **Eye color**: Categories could be "blue," "brown," "green," etc.
  - **Marital status**: Categories might include "single," "married," "divorced," etc.
  - **Types of animals**: Categories could include "cat," "dog," "bird," etc.

2. Ordinal Attributes

- An ordinal attribute is an attribute with possible values that **have a meaningful order or ranking** among them, but the magnitude between successive values is not known.
- Examples of ordinal attributes include:
  - **Educational attainment**: Categories might include "high school diploma," "associate's degree," "bachelor's degree," "master's degree," and "doctorate," with a clear order from least to most advanced.
  - **Economic status**: Categories could be "low income," "middle income," and "high income," with a clear ordering based on income level.
  - **Customer satisfaction levels**: Categories might include "very dissatisfied," "dissatisfied," "neutral," "satisfied," and "very satisfied," with an implied order from least to most satisfied.

3. Binary Attributes

- A **binary attribute** is a special nominal **attribute with only two states: 0 or 1.** Example: Smoker, Gender, etc.
- Two types: **Symmetric and Asymmetric**
  - A binary attribute is **symmetric** if both of its states are equally valuable and carry the same weight.
    - Example: the attribute gender having the states male and female
  - A binary attribute is **asymmetric** if the outcomes of the states are not equally important.
    - Example: Positive and negative outcomes of a medical test for HIV.
    - By convention, we code the most important outcome, which is usually the rarest one, by 1 (e.g., HIV positive) and the other by 0 (e.g., HIV negative).

4. Numeric Attributes:

- It is quantitative, such that quantity can be **measured and represented in integer or real values.** Example: age, income, temperature, Height, etc.
- Two Types: Interval Scaled and Ratio Scaled
  - **Interval attributes** are measured on a scale of equal-size units.
    - No true zero-point
    - E.g., calendar dates , temperature, etc.
  - **Ratio attribute** is a numeric attribute with an inherent zero-point
    - e.g., height, weight, distance, etc.
    - Twice as tall, twice as far, ….

Other types of attribute

- **Discrete Attribute**
  - Has only a **finite set** of values
    - E.g., zip codes, profession, or the set of words in a collection of documents
  - Sometimes, represented as integer variables
  - Note: Binary attributes are a special case of discrete attributes
- **Continuous Attribute**
  - Has **real numbers** as attribute values
    - E.g., temperature, height, or weight
  - Practically, real values can only be measured and represented using a finite number of digits
  - Continuous attributes are typically represented as floating-point variables
- **Character**:
  - values are **represented in forms of character** or set of characters (string).
- **Number**:
  - values are **represented in forms of number.**
  - Number may be in form of whole number, decimal number.

Qualitative and Quantitative

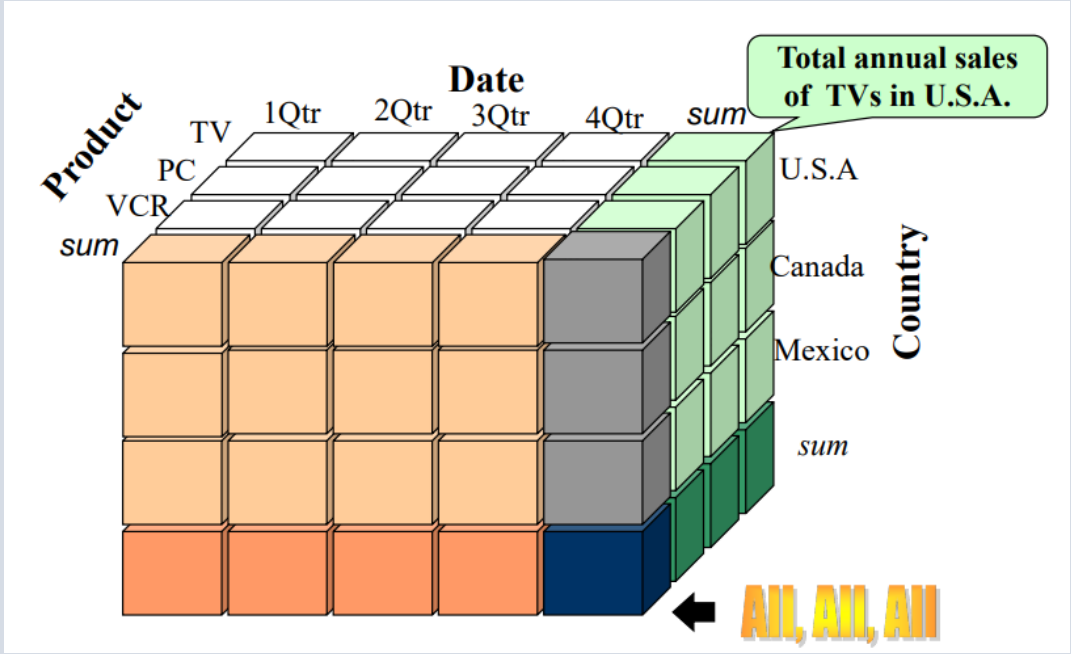
Qualitative attributes:

- Qualitative attributes are properties of an object or **entity that are descriptive** in nature and are not easily measured with numerical values.
- Examples of qualitative attributes include qualities such as color, texture, taste, smell, shape, emotion, attitude, opinion, and perception.
- Nominal and Ordinal attributes are categorical or qualitative attributes.

- **Quantitative attributes :**
  - Quantitative attributes are properties of an object or **entity that can be measured or expressed using numerical values.**
  - Examples of quantitative attributes include variables such as length, weight, temperature, time, age, income, and numerical ratings.
  - These attributes are typically analyzed using statistical methods and mathematical techniques

Data Cube

- A data cube is a fundamental concept which **represents data in a multidimensional format**, allowing for efficient and flexible analysis along multiple dimensions.
- It is defined by **dimensions** and **facts**.
  - **Dimensions** represent the categorical attributes or perspectives along which data can be analyzed. Examples of dimensions include time, geography, product, customer, and sales channel.
  - **Facts**, also known as measures, represent the numerical or quantitative data that are being analyzed. Examples of facts include sales revenue, quantity sold, profit, and expenses.



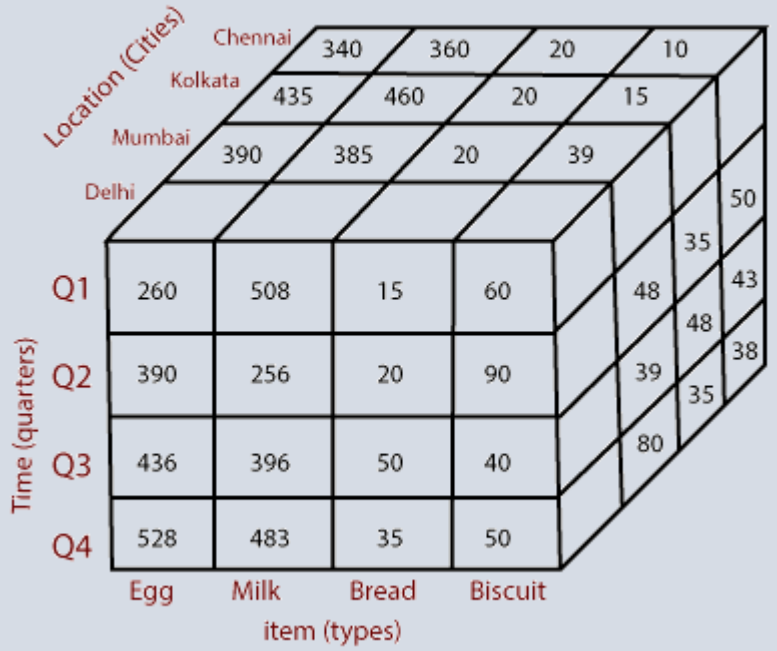
Data Cube Implementation

- For example, suppose the data according to time and item, as well as the location is considered for the cities Chennai, Kolkata, Mumbai, and Delhi. These 3D data are shown in the table. The 3D data of the table are represented as a series of 2D tables.

	Location="Chennai"				Location="Kolkata"				Location="Mumbai"				Location="Delhi"			
	item				item				item				item			
Time	Egg	Milk	Bread	Biscuit	Egg	Milk	Bread	Biscuit	Egg	Milk	Bread	Biscuit	Egg	Milk	Bread	Biscuit
Q1	340	360	20	10	435	460	20	15	390	385	20	39	260	508	15	60
Q2	490	490	16	50	389	385	45	35	463	366	25	48	390	256	20	90
Q3	680	583	46	43	684	490	39	48	568	594	36	39	436	396	50	40
Q4	535	694	39	38	335	365	83	35	338	484	48	80	528	483	35	50

	Location="Chennai"				Location="Kolkata"				Location="Mumbai"				Location="Delhi"			
	item				item				item				item			
Time	Egg	Milk	Bread	Biscuit	Egg	Milk	Bread	Biscuit	Egg	Milk	Bread	Biscuit	Egg	Milk	Bread	Biscuit
Q1	340	360	20	10	435	460	20	15	390	385	20	39	260	508	15	60
Q2	490	490	16	50	389	385	45	35	463	366	25	48	390	256	20	90
Q3	680	583	46	43	684	490	39	48	568	594	36	39	436	396	50	40
Q4	535	694	39	38	335	365	83	35	338	484	48	80	528	483	35	50

- Conceptually, it may also be represented by the same data in the form of a 3D data cube, as shown in fig:



Assignment:

- Implement a data cube for the below dataset:

Country	Nepal			India			Japan		
Year/Game	Gold	Silver	Bronze	Gold	Silver	Bronze	Gold	Silver	Bronze
2005	5	6	1	6	4	3	10	7	9
2010	10	7	2	12	8	6	20	14	6
2015	15	8	3	24	12	9	30	21	3
2020	20	9	4	36	16	12	40	28	1

2.2 Basic Statistical Descriptions of Data

Basic statistical descriptions of data

- Basic statistical descriptions of data are essential to understand and analyze datasets effectively.
- They provide summaries of the data distribution, variability, and relationships among attributes.
- These summaries are vital in data preprocessing and exploration stages in data mining.
- For data preprocessing to be successful, it is essential to have an overall picture of your data.
  - Basic statistical descriptions can be used to identify properties of the data and highlight which data values should be treated as noise or outliers.
- Applications in Data mining:
  1. **Data Cleaning:** Detecting and handling outliers and missing values.
  2. **Data Transformation:** Normalization and standardization of data for uniformity.
  3. **Feature Selection:** Identifying important attributes based on variability and correlations.
  4. **Model Evaluation:** Understanding the distribution to choose appropriate machine learning models.

1. Measuring the Central Tendency: Mean, Median, and Mode

- Central tendency describes the location or central value of the data distribution.
- Measures of central tendency include the mean, median, mode, and midrange.

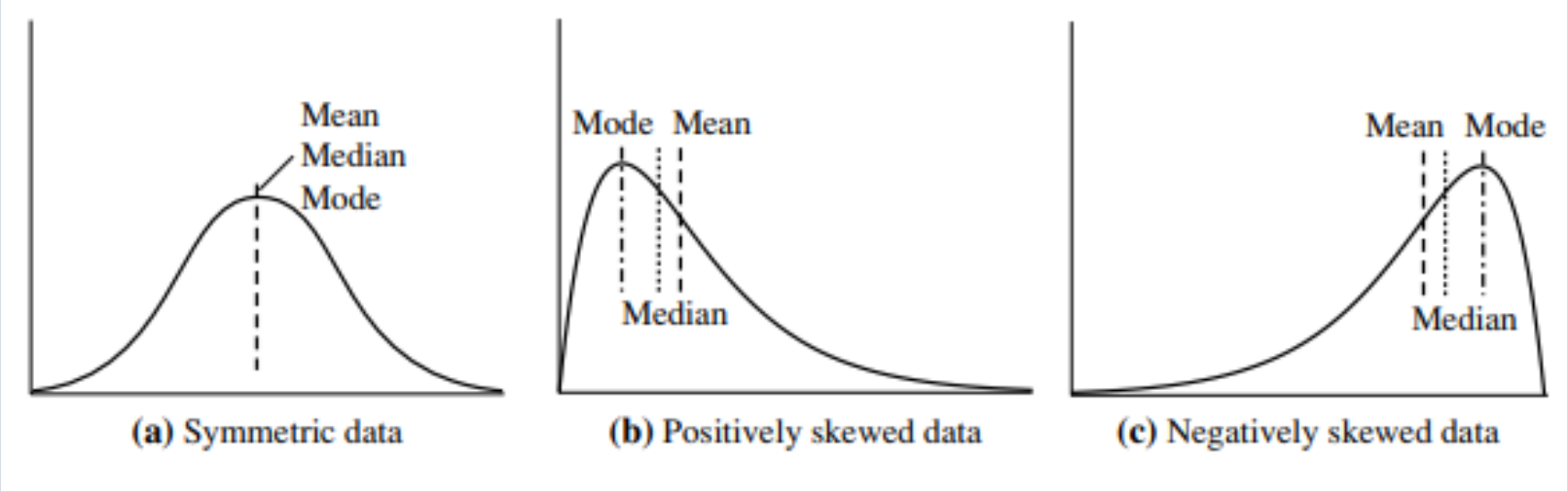


Figure: Mean, median and mode of symmetric versus positively and negatively skewed data

a) mean

- The most common and effective numeric measure of the “center” of a set of data is the (arithmetic) mean.
- A major problem with the mean is its **sensitivity to extreme** (e.g., outlier) values. Even a small number of extreme values can corrupt the mean. For example, the mean salary at a company may be substantially pushed up by that of a few highly paid managers.
- The mean is given by:

$$\text{Mean} = \frac{\sum_{i=1}^{n} x_i}{n}$$

- The weighted arithmetic mean or weighted average is given by:

$$\bar{x} = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i} = \frac{w_1 x_1 + w_2 x_2 + \cdots + w_N x_N}{w_1 + w_2 + \cdots + w_N}.$$

- Suppose we have the following values for salary (in thousands of dollars), shown in increasing order: 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110.

- Then :

$$\bar{x} = \frac{30+36+47+50+52+52+56+60+63+70+70+110}{12}$$

$$= \frac{696}{12} = 58.$$

Thus, the mean salary is \$58,000.

b) Median

- For skewed (asymmetric) data, a better measure of the center of data is the median, which is the middle value in a set of ordered data values.
- It is the value that separates the higher half of a data set from the lower half
- In probability and statistics, the median generally applies to numeric data; however, we may extend the concept to ordinal data.
- Suppose that a given data set of N values for an attribute X is sorted in increasing order.
  - If N is odd, then the median is the middle value of the ordered set.
  - If N is even, then the median is not unique; it is the two middlemost values and any value in between. If X is a numeric attribute in this case, by convention, the median is taken as the average of the two middlemost values.

- Suppose we have the following values for salary (in thousands of dollars), shown in increasing order: 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110.
- Then ,

$$\text{Median} = \frac{52+56}{2} = \frac{108}{2} = 54$$

c) Mode

- The mode is anther measure of central tendency.
- The mode for a set of data is the value that occurs most frequently in the set.
- Therefore, it can be determined for qualitative and quantitative attributes.
- It is possible for the greatest frequency to correspond to several different values, which results in more than one mode.
- Data sets with one, two, or three modes are respectively called unimodal, bimodal, and trimodal.
- In general, a data set with two or more modes is multimodal. At the other extreme, if each data value occurs only once, then there is no mode.
- Example: For the dataset: : 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110.

  The two modes are: 52 and 70

d) Midrange

- The midrange can also be used to assess the central tendency of a numeric data set.
- It is the average of the largest and smallest values in the set.
- This measure is easy to compute using the SQL aggregate functions, max() and min().
- Example:

  For the dataset: : 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110.

  The midrange = $\frac{30+110}{2} = 70$

Example with python code:

```
import statistics
# Sample dataset
data = [30, 36, 47, 50, 52, 52, 56, 60, 63,
70, 70, 110]
# Calculate Mean
mean = sum(data) / len(data)
print(f"Mean: {mean}")
# Calculate Median
median = statistics.median(data)
print(f"Median: {median}")
# Calculate Mode
mode = statistics.mode(data)
print(f"Mode: {mode}")
# Calculate Midrange
midrange = (min(data) + max(data)) / 2
print(f"Midrange: {midrange}")
```

```
Mean: 58.0
Median: 54.0
Mode: 52
Midrange: 70.0
```

2. Measuring the Dispersion of Data: Range, Quartiles, Variance, Standard Deviation, and Interquartile Range

- We now look at measures to assess the dispersion or spread of numeric data.
- Dispersion describes the spread of the data.
- The measures include range, quartiles, percentiles, and the interquartile range.

a) Range:

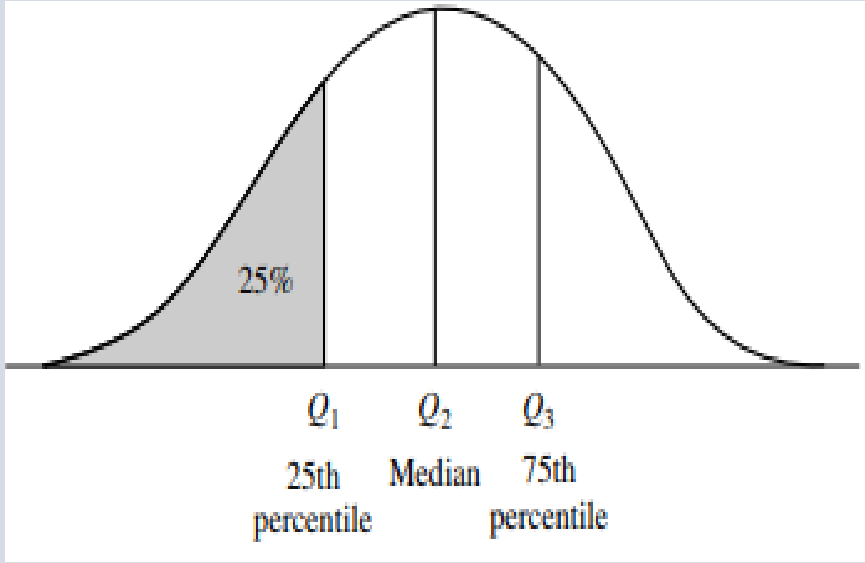
- The range of the set is the difference between the largest (max()) and smallest (min()) values.

b) Quantiles and Quartiles:

- Quantiles are points taken at regular intervals of a data distribution, dividing it into essentially equal size consecutive sets.
- The 2-quantile is the data point dividing the lower and upper halves of the data distribution. It corresponds to the median.
- The 4-quantiles are the three data points that split the data distribution into four equal parts; each part represents one-fourth of the data distribution. They are more commonly referred to as quartiles.

c) Percentile:

- The 100-quantiles are more commonly referred to as percentiles; they divide the data distribution into 100 equal-sized consecutive sets



d) Inter-quartile range:

- Measures the range of the middle 50% of the data.
- IQR = Q3 - Q1
- For the dataset containing 12 observations : 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110.
- The quartiles are the third, sixth and ninth values, respectively, in the sorted order list.
- Therefore, Q1=47, Q3=63 and the IQR= 63-47 = 16

Example with python code:

```
import numpy as np
# Data
data = [30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110]
# Calculate Range
data_range = max(data) - min(data)
print(f"Range: {data_range}")
# Calculate Quartiles
q1 = np.percentile(data, 25)  # First quartile (25th percentile)
q2 = np.percentile(data, 50)  # Second quartile (median, 50th percentile)
q3 = np.percentile(data, 75)  # Third quartile (75th percentile)
print(f"Q1 (25th percentile): {q1}")
print(f"Q2 (Median, 50th percentile): {q2}")
print(f"Q3 (75th percentile): {q3}")
# Calculate Inter-Quartile Range (IQR)
iqr = q3 - q1
print(f"IQR (Inter-Quartile Range): {iqr}")
```

```
Range: 80
Q1 (25th percentile): 49.25
Q2 (Median, 50th percentile): 54.0
Q3 (75th percentile): 64.75
IQR (Inter-Quartile Range): 15.5
```

e) Variance (σ^2):

- Measures average squared deviation from the mean.
- **Variance** in statistics measures how far individual data points in a dataset are spread out from the mean (average) of the data. It is a way to quantify the variability or dispersion of the dataset.

$$\text{Variance} = \frac{\sum_{i=1}^{n}(x_i - \text{Mean})^2}{n}$$

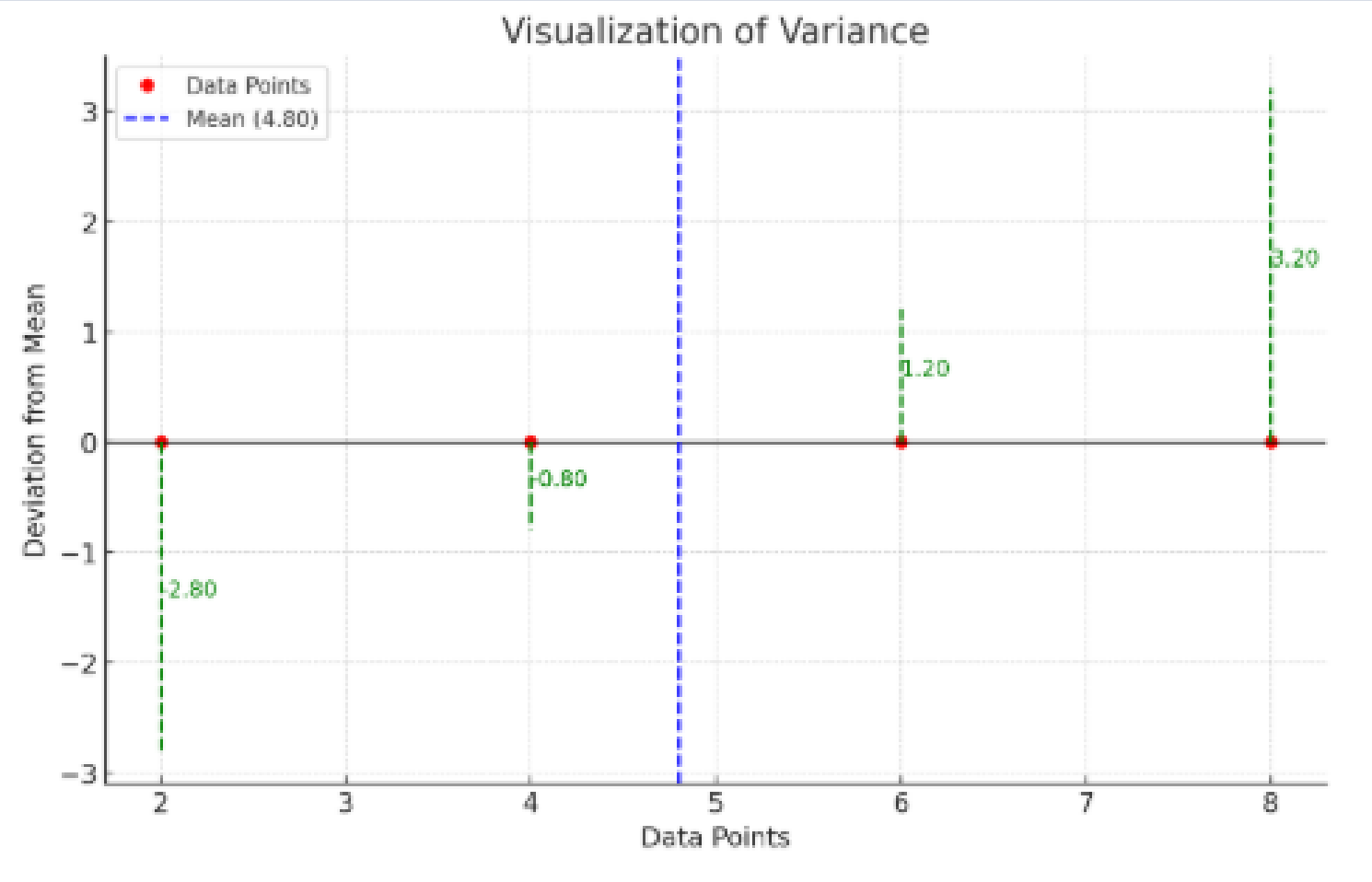
- Example: Imagine you have the following data representing the number of apples in five baskets: as 2, 4,4,6,8

$$\text{Mean} = \frac{\text{Sum of all data points}}{\text{Number of data points}} = \frac{2+4+4+6+8}{5} = \frac{24}{5} = 4.8$$

Data Point (Apples)	Deviation from Mean $(x - \bar{x})$	Squared Deviation $((x - \bar{x})^2)$
2	$2 - 4.8 = -2.8$	$(-2.8)^2 = 7.84$
4	$4 - 4.8 = -0.8$	$(-0.8)^2 = 0.64$
4	$4 - 4.8 = -0.8$	$(-0.8)^2 = 0.64$
6	$6 - 4.8 = 1.2$	$(1.2)^2 = 1.44$
8	$8 - 4.8 = 3.2$	$(3.2)^2 = 10.24$
Sum		$7.84 + 0.64 + 0.64 + 1.44 + 10.24 = 20.8$
Variance		$\frac{20.8}{5} = 4.16$

- The **mean** is 4.8, showing the central value. The **variance** is 4.16, representing the average squared deviation.
- The variance (4.16) indicates how spread out the number of apples in the baskets is around the mean. If the variance were smaller, the numbers would be closer to the mean. If the variance were larger, the numbers would be more spread out.

- Data points are: 2, 4,4,6,8
- Variance = 4.16



f) Standard Deviation (σ):

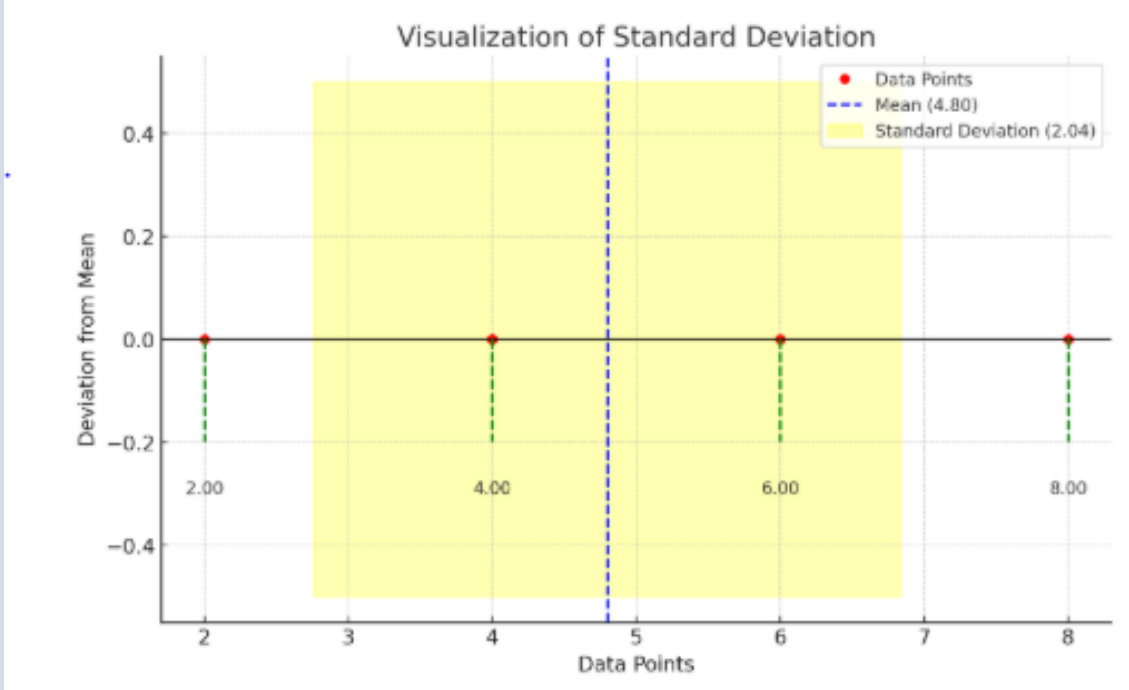
- Standard deviation is a measure of how spread out the data points in a dataset are around the mean.
- It is the square root of the variance and is expressed in the same units as the data, making it easier to interpret compared to variance.
- Provides deviation in the same unit as the data.

$$\text{Standard Deviation} = \sqrt{\text{Variance}}$$

Example: lets consider the same data points: 2,4,4,6,8

Here: Mean= 4.8, Variance= 4.16

Standard Deviation = $\sqrt{variance} = \sqrt{4.16} \approx 2.04$



Yellow Shaded Area: Shows the range within one standard deviation (2.04) from the mean, covering mean−σ to mean +σ (2.76 to 6.84).

Example with python code:

```python
import numpy as np
# Data
data = [2, 4, 4, 6, 8]

# Compute Mean
mean = np.mean(data)
print(f"Mean: {mean}")

# Compute Population Variance (no dividing by n-1)
squared_deviations = [(x - mean) ** 2 for x in data]
variance = np.sum(squared_deviations) / len(data)
print("Variance:" ,variance)

# Compute Population Standard Deviation
std_dev = np.sqrt(variance)
print("Standard Deviation:",std_dev)
```

```
Mean: 4.8
Variance: 4.16
Standard Deviation: 2.039607805437114
```

Graphics Display of Basic Statistical Descriptions of Data

- Quantile-Quantile Plot
- Histograms
- Scatter plots

2.3 Data Preprocessing

2.2 Data Pre-processing

- Data mining is a methodology in computer science for discovering meaningful patterns and knowledge from large amounts of data.
- However, before a data mining model can be applied, the raw data must be preprocessed to ensure that it is in a suitable format for analysis.
- Data preprocessing is an essential step in the data mining process and can greatly impact the accuracy and efficiency of the final results.
- ***Data preprocessing refers to a set of techniques and procedures applied to raw data to prepare it for analysis or to improve the performance of machine learning algorithms.***

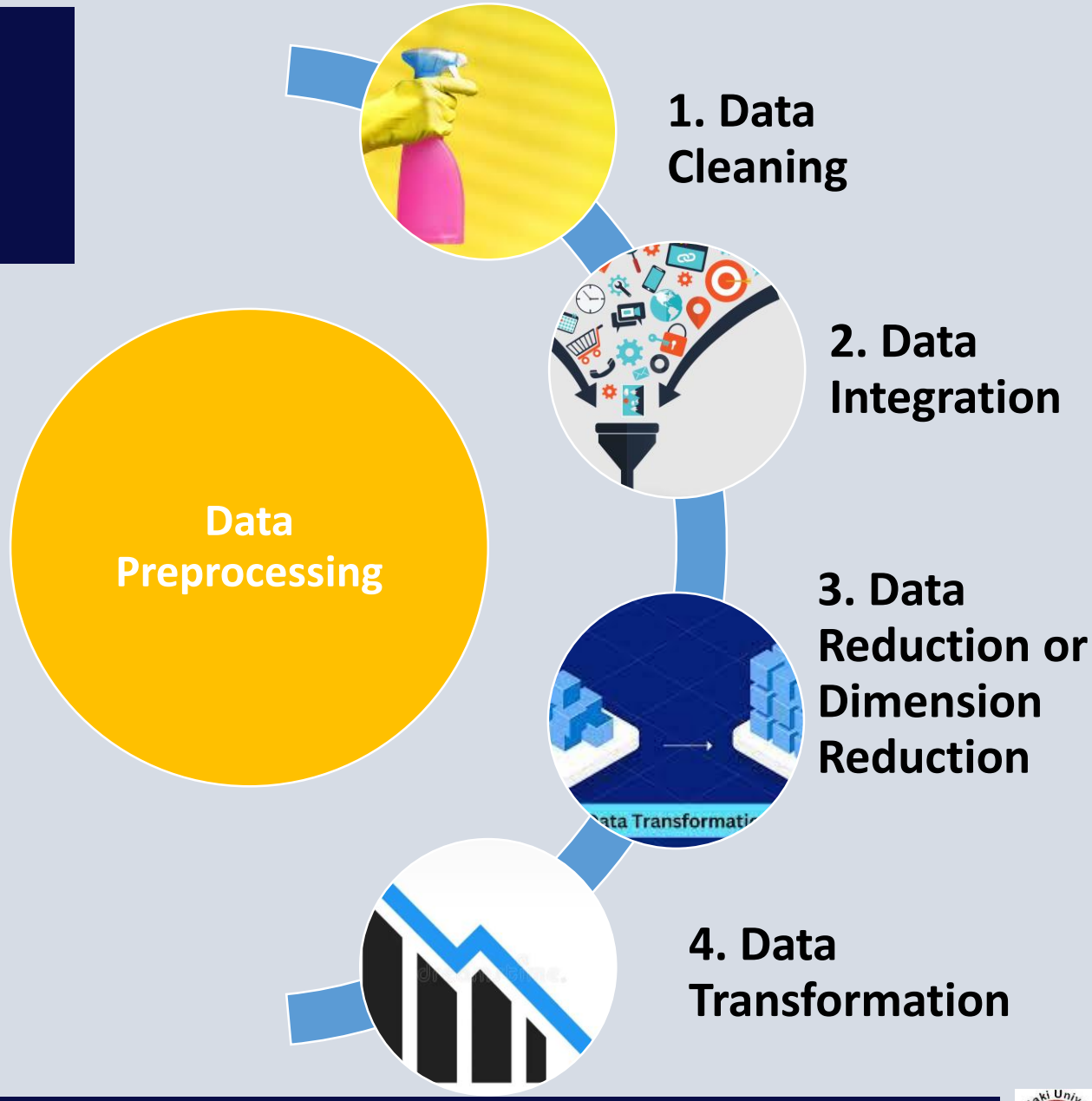
raw data → suitable format

- It involves **the cleaning, transforming, and integrating of data** in order to make it ready **for analysis.**
- The goal of data preprocessing is to improve the quality of the data and **to make it more suitable for the specific data mining task**.

Importance of Data Pre-processing

- Preprocessing of data is mainly to check the data quality. The quality can be checked by the following:
  1. **Accuracy**: To check whether the data entered is correct or not.
  2. **Completeness**: To check whether the data is available or not recorded.
  3. **Consistency:** To check whether the same data is kept in all the places that do or do not match.
  4. **Timeliness**: The data should be updated correctly.
  5. **Believability**: The data should be trustable.
  6. **Interpretability**: The understandability of the data

Major Tasks in Data Preprocessing:



- Data cleaning:
- Data Integration
- Data Reduction or Dimension Reduction
- Data Transformation

2.4 Data Cleaning

1. Data Cleaning

- Data in the Real World Is Dirty: Lots of potentially incorrect data due to faulty instrument, human or computer error, transmission error.
- These data can be **incomplete, inaccurate, inconsistent** and **noisy.**
- So, these needs to be cleaned.
- Data cleaning is the process of removing incorrect data, incomplete data, and inaccurate data from the datasets, and it also replaces the missing values.
- Some techniques for data cleaning are:
  i. Handling Missing Values
  ii. Removing duplicates
  iii. Removing inconsistencies
  iv. Handling outliers

i) Handling Missing Values

- Missing data **refers to the absence** of certain values in the dataset.
- Missing data is **a common challenge in data science**, affecting analysis accuracy and statistical power.
- **Missing data**, or missing values, occur when you don't have data stored for certain variables or participants.
- Data can go missing due to incomplete data entry, equipment malfunctions, lost files, and many other reasons.
- In any dataset, there are usually some missing data. In quantitative research, missing values appear as blank cells in your spreadsheet of csv file.
- Consequences of missing data:
  - Result can be biased
  - Incorrect accuracy and precision
  - A substantial loss in power
- Handling of missing data includes:
  - First identify the missing data
  - Remove or fill the missing data

a) Identifying Missing Data

- Identifying missing data is an essential first step in data cleaning.
- common methods to detect missing values in Python are:
  i. Using ***isna()*** or ***isnull()***
  ii. Using ***info()***
  iii. Using ***notna()***
  iv. Using ***any()*** and ***sum()*** with ***isna()***

i) Using *isna()* or *isnull()*

- The ***isna()*** and ***isnull()*** fucntions in Pandas detect **NaN** values.
- They return **True** for missing values and **False** for present values.

```
import pandas as pd
import numpy as np

# Sample data with missing values
data = pd.DataFrame({
    'Age': [25, np.nan, 35, 40, np.nan],
    'Income': [50000, 60000, np.nan, 70000, 80000]
})

# Check for missing values
print("Using isna():")
print(data.isna())

print("Using isnull():")
print(data.isnull())
```

```
Using isna():
     Age  Income
0  False   False
1   True   False
2  False    True
3  False   False
4   True   False
```

```
Using isnull():
     Age  Income
0  False   False
1   True   False
2  False    True
3  False   False
4   True   False
```

ii) Using *info()*

- The ***info()*** function in Pandas provides a quick summary of the dataset, including the count of non-null values per column.

```
import pandas as pd
import numpy as np

# Sample data with missing values
data = pd.DataFrame({
    'Age': [25, np.nan, 35, 40, np.nan],
    'Income': [50000, 60000, np.nan, 70000, 80000]
})

# Check for missing values
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Age     3 non-null      float64
 1   Income  4 non-null      float64
dtypes: float64(2)
memory usage: 208.0 bytes
None
```

iii) Using *notna()* or *notnull()*

- The ***notna()*** or ***notnull()*** functions
  - return **True** for non-missing values and
  - **False** for missing values.

```
import pandas as pd
import numpy as np

# Sample data with missing values
data = pd.DataFrame({
    'Age': [25, np.nan, 35, 40, np.nan],
    'Income': [50000, 60000, np.nan, 70000, 80000]
})

# Check for missing values
print("\nUsing notna():")
print(data.notna())
```

```
Using notna():
     Age  Income
0   True    True
1  False    True
2   True   False
3   True    True
4  False    True
```

iv) Using *any()* or *sum()* with *isna()*

- To quickly check if any column contains missing values, we can use any() or sum() with isna()

```
Any missing values in each column?
Age       True
Income    True
dtype: bool
```

```
Count of missing values in each column:
Age       2
Income    1
dtype: int64
```

```
import pandas as pd
import numpy as np

# Sample data with missing values
data = pd.DataFrame({
   'Age': [25, np.nan, 35, 40, np.nan],
   'Income': [50000, 60000, np.nan, 70000, 80000]
})
# Check if any value is missing in each column
print("\nAny missing values in each column?")
print(data.isna().any())

# Count the number of missing values per column
print("\nCount of missing values in each column:")
print(data.isna().sum())
```

b) Handling Missing Data

1. **Ignore the tuple:**
   usually done when class label is missing (when doing classification)—not effective when the % of missing values per attribute varies considerably
2. **Fill in the missing value manually**:
   tedious + infeasible? it is not recommended when that dataset is big.
3. **Imputation:** Replace missing values with:
   - Mean, median, or mode (for numerical data).
   - Most frequent value (for categorical data).
   - Predicted values using machine learning models.
4. **Forward/Backward Fill:** Use previous or next valid values (common in time-series data).
5. **Use the most probable value to fill in the missing value:** This may be determined with regression, inference-based tools using a Bayesian formalism, or decision tree induction.

Techniques for handing missing data with Python:

- Filtering out data or filling missing values in Python, especially with the **Pandas** library, is an essential task when cleaning data for analysis.
- You may want to remove rows or columns that do not meet specific conditions or that contain missing or unwanted values.
- The ***dropna()*** function can be used to remove the rows with missing values in the dataset.

i) Removing rows or with missing values

- You can remove rows or columns with missing values (i.e NaN) using dropna().

```
import pandas as pd
import numpy as np

# Sample data with missing values
data = pd.DataFrame({
    'Age': [25, np.nan, 35, 40, np.nan],
    'Income': [50000, 60000, np.nan,
70000, 80000]
})

#Remove rows with any missing values
filtered_data= data.dropna()
print(filtered_data)
```

	Age	Income
0	25.0	50000.0
3	40.0	70000.0

- You can also remove columns with missing values (i.e NaN) using dropna().

```
import pandas as pd
import numpy as np

# Sample data with missing values
data = pd.DataFrame({
    'Age': [25, np.nan, 35, 40, np.nan],
    'Income': [50000, 60000, 40000,
70000, 80000],
    'Gender': ['M', 'F', 'M', 'F', 'M']
})

# Drop columns with any missing values
data_cleaned = data.dropna(axis=1)
print(data_cleaned)
```

```
   Income Gender
0   50000      M
1   60000      F
2   40000      M
3   70000      F
4   80000      M
```

ii) Filling Missing Values

- Another approach is to **fill** missing values uidng ***fillna()*** with a specific value, such as the mean, median, or mode.
- This is commonly done for numerical columns.

```
import pandas as pd
import numpy as np

# Sample data with missing values
data = pd.DataFrame({
    'Age': [25, np.nan, 35, 40, np.nan],
    'Income': [50000, 60000, np.nan, 70000, 80000],
    'Gender': ['M', 'F', 'M', 'F', 'M']
})

# Fill missing numerical values with the mean
data['Age'] = data['Age'].fillna(data['Age'].mean()).astype(int)
data['Income'] =
data['Income'].fillna(data['Income'].mean()).astype(int)

print(data)
```

	Age	Income	Gender
0	25	50000	M
1	33	60000	F
2	35	65000	M
3	40	70000	F
4	33	80000	M

- Categorical data can be filled using the mode.

```python
import pandas as pd
import numpy as np

# Sample data with missing values
data = pd.DataFrame({
    'Age': [25, np.nan, 35, 40, np.nan],
    'Income': [50000, 60000, np.nan, 70000, 80000],
    'Gender': ['M', 'F', 'F', np.nan, 'M']
})

# Fill missing numerical values with the mean
# Fill missing categorical values with the mode
data['Age'] = data['Age'].fillna(data['Age'].mean()).astype(int)
data['Income'] =
data['Income'].fillna(data['Income'].mean()).astype(int)
data['Gender'] = data['Gender'].fillna(data['Gender'].mode()[0])
print(data)
```

	Age	Income	Gender
0	25	50000	M
1	33	60000	F
2	35	65000	F
3	40	70000	F
4	33	80000	M

iii) Using Forward Fill or Backward Fill

- This method is useful when the missing values are part of a time series or ordered data.
- You can fill the missing values by propagating the last valid value forward (ffill) or the next valid value backward (bfill)

```
import pandas as pd
import numpy as np

# Sample data with missing values
data = pd.DataFrame({
    'Age': [25, np.nan, 35, 40, np.nan],
    'Income': [50000, 60000, np.nan, 70000, 80000],
    'Gender': ['M', 'F', 'F', np.nan, 'M']
})

# Forward fill missing values
data_filled = data.fillna(method='ffill')
data_filled['Age'] = data_filled['Age'].astype(int)
data_filled['Income'] = data_filled['Income'].astype(int)
print(data_filled)
```

	Age	Income	Gender
0	25	50000	M
1	25	60000	F
2	35	60000	F
3	40	70000	F
4	40	80000	M

Backward Fill

- Sample with backward fill

```
import pandas as pd
import numpy as np

# Sample data with missing values
data = pd.DataFrame({
    'Age': [25, np.nan, 35, 40, 50],
    'Income': [50000, 60000, np.nan, 70000, 80000],
    'Gender': ['M', 'F', 'F', np.nan, 'M']
})

# Backward fill missing values
data_bfilled = data.fillna(method='bfill')
data_bfilled['Age'] = data_bfilled['Age'].astype(int)
data_bfilled['Income'] = data_bfilled['Income'].astype(int)
print(data_bfilled)
```

	Age	Income	Gender
0	25	50000	M
1	35	60000	F
2	35	70000	F
3	40	70000	M
4	50	80000	M

iv) Interpolating Missing Values

- Pandas also provides an interpolation method, which can be useful when the data is numeric and you want to estimate missing values based on surrounding data points.

```
import pandas as pd
import numpy as np

# Sample data with missing values
data = pd.DataFrame({
    'Age': [25, np.nan, 35, 40, 50],
    'Income': [50000, 60000, np.nan, 70000, 80000],
    'Gender': ['M', 'F', 'F', 'M', 'M']
})

# Backward fill missing values
data_filled = data.interpolate()
print(data_filled)
```

```
    Age   Income Gender
0  25.0  50000.0      M
1  30.0  60000.0      F
2  35.0  65000.0      F
3  40.0  70000.0      M
4  50.0  80000.0      M
```

v) Using a Model to predict Missing values

- In some advanced cases, you might want to predict the missing values using a model, such as a regression model or machine learning model.
- This is often done when the missing data is **Missing Not at Random (MNAR)**.

ii) Removing duplicates

- Duplicate rows may be found in a DataFrame for any number of reasons.
- Removing duplicate includes:
  - Data matching
  - Data Deduplication
- **Data matching** involves identifying and linking related records within or across datasets, especially when exact matches are difficult due to variations in data representation, such as spelling variations or different data formats. In python, the `duplicated()` method returns "True" for exactly duplicated rows.
- **Data deduplication** is the process of identifying and removing duplicate records from a dataset. In python, the `drop_duplicates()` can be used to filter out one of the duplicated rows.
- One of the techniques for removing duplicates is the ***Exact Matching.***
  - In exact matching, two records are considered a match if all selected fields are identical in both records.

Examples:

```
import pandas as pd
import numpy as np
data = pd.DataFrame({"k1": ["one",
       "two"] * 3 + ["two"],"k2": [1,
       1, 2, 3, 3, 4, 4]})
data
data.duplicated()
data.drop_duplicates()
```

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4
6	two	4

	0
0	False
1	False
2	False
3	False
4	False
5	False
6	True

dtype: bool

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4

- The DataFrame method ***duplicated()*** returns a Boolean Series indicating whether each row is a duplicate (its column values are exactly equal to those in an earlier row) or not.
- Relatedly, ***drop_duplicates()*** returns a DataFrame with rows where the duplicated array is False filtered out.

- Suppose, we have one more column “V1” and we need to filter out duplicates based only on the “K1” column:

```
import pandas as pd
import numpy as np
data = pd.DataFrame({
        "k1": ["one", "two"] * 3 + ["two"],
        "k2": [1, 1, 2, 3, 3, 4, 4]})

# Add one more column
data["v1"] = range(7)
data

# filter out data based on K1 column
data.drop_duplicates(subset=["k1"])
```

	k1	k2	v1
0	one	1	0
1	two	1	1
2	one	2	2
3	two	3	3
4	one	3	4
5	two	4	5
6	two	4	6

	k1	k2	v1
0	one	1	0
1	two	1	1

Examples:

```
import pandas as pd

# Sample dataset

data = {'Name': ['Alice Smith', 'Bob Johnson', 'Alice
      Smith', 'Carol White'],

      'Email': ['alice@example.com', 'bob@example.com',
      'alice@example.com', 'carol@example.com']}

df = pd.DataFrame(data)


# Identify duplicates based on the 'Name' and 'Email' fields

duplicates = df[df.duplicated(subset=['Name', 'Email'])]

print("Exact Matches:")

print(duplicates)
```

```
Exact Matches:
            Name                Email
2  Alice Smith  alice@example.com
```

iii) Removing Inconsistencies

- This technique ensures related data is logically consistent.
- Some of the techniques are:
  1. **Range Validation:** Ensure numerical values fall within a specified range.
  2. **Type Validation:** Check if data values match the expected data type.
  3. **Format Validation:** Ensure text data (e.g., dates, emails) follows a specific format.
  4. **Uniqueness Check:** Ensure values in specific columns (e.g., IDs) are unique.
  5. **Completeness Check**: Identify missing or null values in the data.
  6. **Business Rule validation:** Ensure the data complies with specific domain rules (eg, negative prices check)
  7. **Consistency Check:** Ensure related data is logically consistent.
  8. **Cross-field Validation:** Ensure consistency across related fields.

Example 1: Range Validation

Ensure numerical values fall within a specified range.

```
import pandas as pd

data = {'Age': [25, 40, -5, 100, 35]}
df = pd.DataFrame(data)
print(df)

# Validate that age is within 0 to 120
df['Valid_Age'] = df['Age'].apply(lambda x: 0 <= x <= 120)
print(df)

# Calculate the mean of valid ages
valid_mean = df.loc[df['Valid_Age'], 'Age'].mean()

# Replace invalid ages with the mean of valid ages
df.loc[~df['Valid_Age'], 'Age'] = valid_mean

# Drop the validation column for cleaner output
df = df.drop(columns=['Valid_Age'])

print(df)
```

```
   Age
0   25
1   40
2   -5
3  100
4   35
----------------
   Age  Valid_Age
0   25       True
1   40       True
2   -5      False
3  100       True
4   35       True
----------------
   Age
0   25
1   40
2   50
3  100
4   35
```

Example 2: Type Validation

This technique is used to check if data values match the expected data type.

```
import pandas as pd
data = {'ID': [1, 2, 'three', 4, 5]}
df = pd.DataFrame(data)

# Validate that all IDs are integers
df['Valid_ID'] = df['ID'].apply(lambda x:
isinstance(x, int))
invalid_ids = df[~df['Valid_ID']]
print(invalid_ids)
```

```
      ID  Valid_ID
2  three     False
```

Example 3: Format Validation

This technique ensures text data (eg. Dates, emails, time) follows a specific format.

```
from datetime import datetime

dates = ['2024-03-29', '06/24/2025', 'invalid_date']
date_format="%Y-%m-%d"

def is_valid_date(date, date_format):
    try:
        datetime.strptime(date, date_format)
        return True
    except ValueError:
        return False

valid_dates = [is_valid_date(d,date_format) for d in dates]
print(valid_dates)
```

Output:

```
[True,
False,
False]
```

```python
from datetime import datetime
import numpy as np
# Sample dates (with invalid ones included)
dates = ['2024-03-29','2023/02/20','04-29-2021','02/22/2020','invalid_date']
# Function to validate and convert date format
def validate_and_convert_date(date_string, valid_format="%Y-%m-%d"):
    # List of possible formats to try
    date_formats = ["%Y-%m-%d", "%Y/%m/%d","%m-%d-%Y","%m/%d/%Y"]
    for fmt in date_formats:
        try:
            # Try parsing the date with the current format
            date_obj = datetime.strptime(date_string, fmt)
            return date_obj.strftime(valid_format)  # Convert to the desired
format
        except ValueError:
            # If the date doesn't match this format, continue to the next one
            continue
    # If none of the formats match, return NaN
    return np.nan
# Apply the function to validate and convert dates
converted_dates = [validate_and_convert_date(d) for d in dates]
# Check the result with NaN
print(converted_dates)
```

Output:

```
['2024-03-29', '2023-02-20', '2021-04-29', '2020-02-22', nan]
```

Example 4: Uniqueness check

This technique ensures values in specific columns (e.g., IDs) are unique. The duplicated() method can be used to check the uniqueness of an attribute's values.

```
import pandas as pd
data = {'ID': [1, 2, 2, 4, 5]}
df = pd.DataFrame(data)

# Check for duplicate IDs
duplicates = df[df.duplicated(subset=['ID'], keep=False)]
print(duplicates)
```

	ID
1	2
2	2

Example 5: Completeness check

This technique is used to identify any missing or null values in the dataset. The isna() or isnull() method can be used.

```
import pandas as pd
data = {'Name': ['Alice', None, 'Charlie',
'Diana'], 'Age': [25, 30, None, 40]}
df = pd.DataFrame(data)

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

```
Name    1
Age     1
dtype: int64
```

Example 6: Consistency Check

This technique ensures related data is logically consistent.

```
import pandas as pd

data = {'Start_Date': ['2024-01-01', '2024-05-01'],
        'End_Date': ['2024-12-31', '2024-03-01']}
df = pd.DataFrame(data)
print(df)
print('-----------------')
# Check if Start_Date is before End_Date
df['Start_Date'] = pd.to_datetime(df['Start_Date'])
df['End_Date'] = pd.to_datetime(df['End_Date'])
df['Valid_Dates'] = df['Start_Date'] <= df['End_Date']
invalid_dates = df[~df['Valid_Dates']]
print("Displaying Invalid dates\n",invalid_dates)
```

```
   Start_Date    End_Date
0  2024-01-01  2024-12-31
1  2024-05-01  2024-03-01
-----------------
Displaying Invalid dates
   Start_Date   End_Date  Valid_Dates
1 2024-05-01 2024-03-01        False
```

Example 7: Cross-field Validation

This technique ensures consistency across related fields.

```
import pandas as pd

data = {'Quantity': [10, 1, 5], 'Price': [100, 50, 50],
'Total': [1000, 200, 250]}
df = pd.DataFrame(data)

# Validate that Total equals Quantity * Price
df['Valid_Total'] = df['Quantity'] * df['Price'] ==
df['Total']
invalid_totals = df[~df['Valid_Total']]
print("Displaying Invalid totals\n",invalid_totals)
```

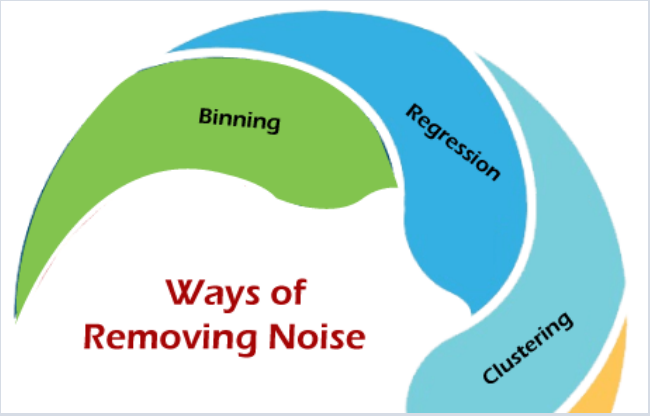
```
Displaying Invalid totals
   Quantity  Price  Total  Valid_Total
1         1     50    200        False
```

iv) Handling Noisy data

- Noise refers to values that are unusually high or low compared to others in a specific group or bin.
- Noise impact the data mining process, they need to be smoothened.
- Key techniques to handle noisy data are:
  - Binning
  - Regression
  - Clustering

How to handle Noisy data?

- Binning
- Regression
- Clustering



1. Binning

- Binning method is used to smoothing data or to handle noisy data.
- First, the data is sorted then and then the sorted values are separated and stored in the form of bins.
- Three methods: Bin mean, Bin median, Bin boundary
  - **Smoothing by bin mean method**: In this method, the values in the bin are replaced by the ***mean* value** of the bin;
  - **Smoothing by bin median**: In this method, the values in the bin are replaced by the *median* value;
  - **Smoothing by bin boundary**: In this method, the *minimum* and *maximum values* in a given bin are identified, and each bin value is then replaced by the closest boundary value.

Example:

Let us consider following data for price: 4 ,9, 15, 8, 21, 24, 21,26 ,34, 25, 28, 29

- **Approach:**
  - ➢Sort the array of a given data set.
  - ➢Divides the range into N intervals, each containing the approximately same number of samples(Equal-depth partitioning).
  - ➢Store mean/ median/ boundaries in each row.

Partition using equal frequency approach:

- Bin 1 : 4, 8, 9, 15
- Bin 2 : 21, 21, 24, 25
- Bin 3 : 26, 28, 29, 34

Smoothing by bin means:

- Bin 1: 9, 9, 9, 9
- Bin 2: 23, 23, 23, 23
- Bin 3: 29, 29, 29, 29

Smoothing by bin boundaries:

- Bin 1: 4, 4, 4, 15
- Bin 2: 21, 21, 25, 25
- Bin 3: 26, 26, 26, 34

Smoothing by bin median:

- Bin 1: 9 9, 9, 9
- Bin 2: 23, 23, 23, 23
- Bin 3: 29, 29, 29, 29

Take an example for a dataset: 50, 55, 57, 58, 60, 62, 65, 70

- The sorted data is : 50, 55,57,58,60, 62, 65, 70, 100
- Step 1: Divide the data into Bins:
  - Bin 1 (50-56): [50, 55]
  - Bin 2 (57-63): [57,58, 60, 62]
  - Bin 3 (64-70): [65, 70, 100]
- Calculate the mean of each Bin:
  - Bin 1 mean = 52.5
  - Bin 2 mean = 59.25
  - Bin 3 mean = 78.33
- Smoothed Dataset is:
  - 52.5, 52.5, 59.25 , 59.25 , 59.25 , 59.25, 78.33, 787.33, 78.33

```python
# Original dataset
data = [50, 55, 57, 58, 60, 100, 62, 65, 70]
# Step 1: Sort the data
data.sort()

# Step 2: Define the number of bins
num_bins = 3
bin_size = len(data) // num_bins
# Step 3: Create bins and replace values with the mean (rounded to 2 decimal places)
smoothed_data = []
for i in range(0, len(data), bin_size):
    bin_group = data[i:i + bin_size]  # Slice the bin
    bin_mean = round(sum(bin_group) / len(bin_group), 2)  # Calculate the mean and round it
    smoothed_data.extend([bin_mean] * len(bin_group))  # Fill the bin with the mean
# Step 4: Print results
print("Original Data:", data)
print("Smoothed Data:", smoothed_data)
```

```
Original Data: [50, 55, 57, 58, 60, 62, 65, 70, 100]
Smoothed Data: [54.0, 54.0, 54.0, 60.0, 60.0, 60.0, 78.33, 78.33, 78.33]
```

2. Regression

- This is used to smooth the data and help handle data when unnecessary data is present.
- For the analysis purpose, regression helps decide the suitable variable.
- ***Linear regression*** refers to finding the best line to fit between two variables so that one can be used to predict the other.
- ***Multiple linear regression*** involves more than two variables.

- Let’s assume we have the following dataset representing the relationship between study hours and test scores.

Study hours	1	2	3	4	5	6	7
Test Score	50	55	60	62	100	67	70

- In this dataset, we can see that the 100 score is an outlier compared to the trend. We can use linear regression to model the relationship and smooth out the data:
  - **Fit a regression model** to the data.
  - **Identify outliers** by calculating the residuals.
  - **Remove or adjust the outliers**.
  - **Recalculate the regression model** and observe the smoothed predictions.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Data
X = np.array([1, 2, 3, 4, 5, 6, 7]).reshape(-1, 1)  # Study Hours
y = np.array([50, 55, 60, 62, 100, 67, 70])  # Test Scores

# Step 1: Fit Linear Regression Model
model = LinearRegression()
model.fit(X, y)

# Step 2: Calculate predicted values and residuals
y_pred = model.predict(X)
residuals = y - y_pred

# Step 3: Identify outliers based on residuals (e.g., threshold > 15)
outliers = np.abs(residuals) > 15
```

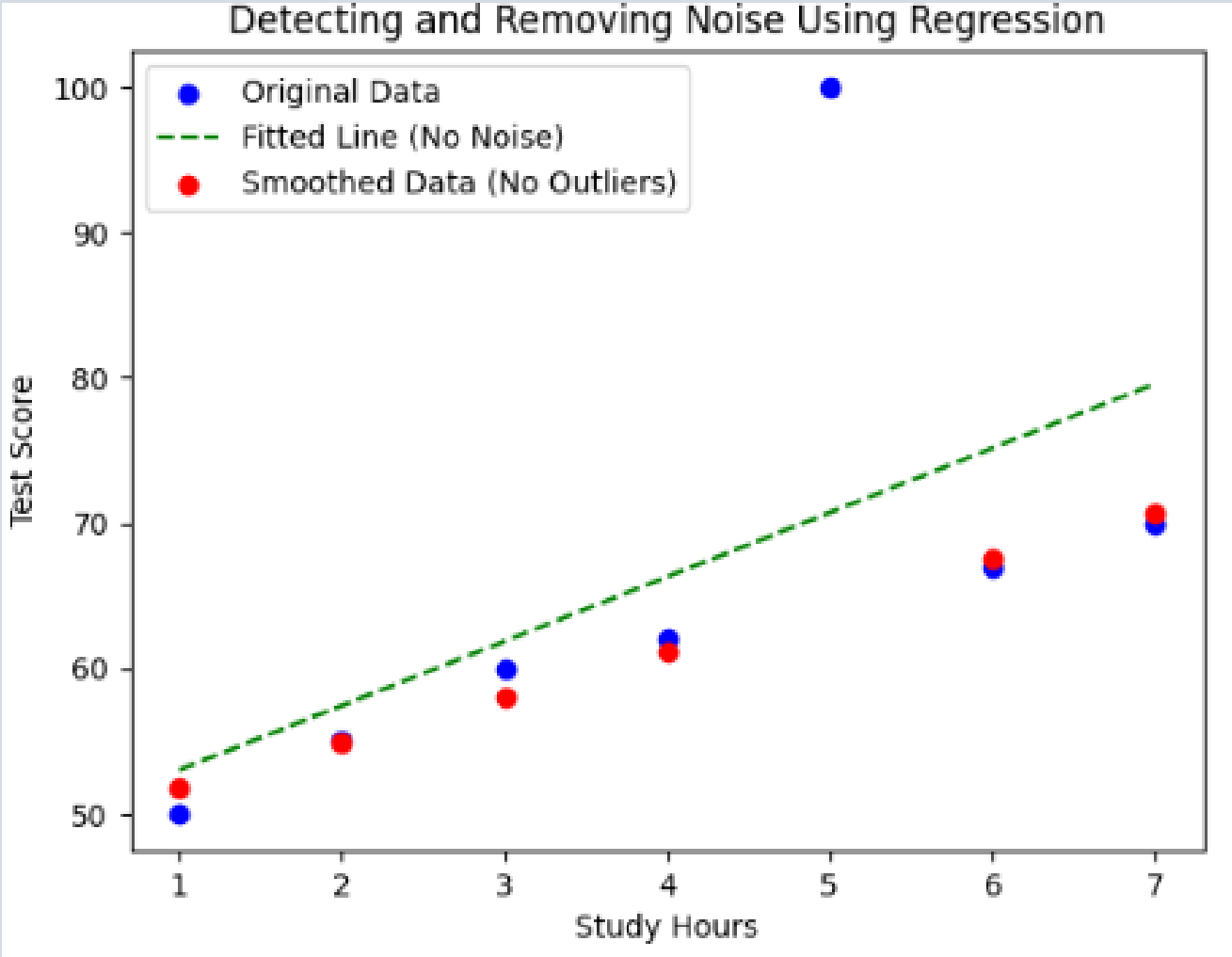
```
# Step 4: Remove outliers (adjust or ignore the noise)
cleaned_X = X[~outliers]
cleaned_y = y[~outliers]

# Re-fit the model without outliers
model.fit(cleaned_X, cleaned_y)
smoothed_y = model.predict(cleaned_X)

# Step 5: Plot the results
plt.scatter(X, y, color='blue', label='Original Data')
plt.plot(X, y_pred, color='green', linestyle='--', label='Fitted Line (No
Noise)')
plt.scatter(cleaned_X, smoothed_y, color='red', label='Smoothed Data (No
Outliers)')
plt.legend()
plt.xlabel('Study Hours')
plt.ylabel('Test Score')
plt.title('Detecting and Removing Noise Using Regression')
plt.show()
```

Output Explanation:

- **Original Data** (blue dots): The actual noisy data points, including the outlier.
- **Fitted Line** (green dashed line): The linear regression line that fits the data including the outlier.
- **Smoothed Data** (red dots): After removing the noisy outlier (100), the regression model is re-fitted, and the new line better reflects the data.



3. Clustering

- This is used for finding the outliers and also in grouping the data. Clustering is generally used in unsupervised learning.
- Outliers may be detected by clustering where similar values are organized into groups (or clusters).
- Values that fall outside of the set of clusters may be considered outliers.
- Clustering is generally used in unsupervised learning.

Note:

- We will study this in more detail later in the chapter “CLUSTERING”

2.5 Data Integration

2. Data Integration

- Data integration in preprocessing refers to the **process of combining data from multiple sources** into a unified dataset that can be used for analysis or modeling.
- The source may include multiple databases, data cubes, or flat files.
- This process involves identifying and accessing the different data sources, mapping the data to a common format, and reconciling any inconsistencies or discrepancies between the sources.
- Data integration can be challenging due to the variety of data formats, structures, and semantics used by different data sources.
- Different data sources may use different data types, naming conventions, and schemas, making it difficult to combine the data into a single view.

Challenges in Data Integration:

1. **Data Heterogeneity**: Different data sources may have inconsistent data formats, schemas, and semantics.
2. **Data Redundancy**: Multiple sources may contain overlapping or duplicate information, requiring techniques for de-duplication.
3. **Data Conflicts**: Conflicting data entries (e.g., different values for the same attribute) require resolving strategies that ensure data consistency.
4. **Data Quality**: Inconsistent, missing, or incorrect data from various sources need to be cleaned, validated, and transformed before integration.

Example: Suppose you are integrating data from two different hospitals. One hospital stores patient names and ages in one database, while the other stores patient names and birthdates in another.

1. **Schema Integration**:

   Hospital 1 uses columns name and age, while Hospital 2 uses name and birthdate. You need to standardize the schema, which could involve ***transforming the birthdate into an age*** column for Hospital 2, or vice versa.

2. **Entity Resolution**:

   There might be cases where a **patient appears in both databases but with slightly different information** (e.g., a typographical error in the name). You would apply ***entity resolution techniques*** to identify the same patient in both datasets.

3. **Redundancy Handling**:

   If a **patient appears multiple times in the merged dataset**, you will need to **remove duplicates** based on unique identifiers (e.g., patient ID).

4. **Conflict Resolution**:

   If the hospitals have **different values for the same attribute** (e.g., different ages for the same patient due to inaccurate data entry), you might decide to **keep the value from the more reliable source** or use a rule to combine the data, like averaging.

Techniques for Handling Data Integration Issues:

1. **Data Transformation**:

   Convert all data to a common format, such as using the same units of measurement, encoding schemes, or date formats.

2. **Matching and Merging**:

   Use methods like fuzzy matching, key matching, or machine learning models to match records referring to the same entity across data sources.

3. **Conflict Resolution**:

   Use business rules, heuristics, or more advanced techniques like probabilistic record linkage to resolve conflicts.

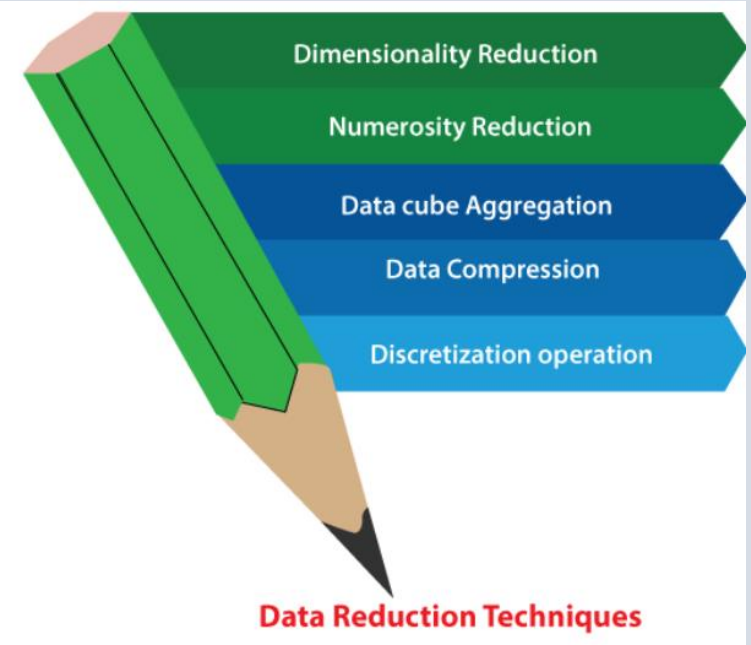
4. **Normalization**:

   Normalize data to bring everything to a common scale, particularly useful when different sources use different ranges or scales for numeric data.

2.6 Data Reduction

3. Data Reduction

- A database/data warehouse may store terabytes of data. Complex data analysis may take a very long time to run on the complete data set.
- This is the process to obtain a reduced representation of the data set that is much smaller in volume but yet produces the same (or almost the same) analytical results.
- ***It is a way of converting the higher dimensions dataset into lesser dimensions dataset ensuring that it provides similar information.***
- Data reduction strategies
  a) Dimensionality Reduction
  b) Numerosity Reduction
  c) Compression



a) Dimensionality Reduction

- Dimensionality reduction is a process used in machine learning and data analysis **to reduce the number of input variables** or features in a dataset while retaining the most important information.
- This process is necessary for real-world applications as the data size is big. In this process, the **reduction of random variables or attributes** is done so that the dimensionality of the data set can be reduced.
- This involves combining and merging the attributes of the data without losing its original characteristics.
- This also helps in the *reduction of storage space* and *computation time.*
- When the data is highly dimensional the problem called **"Curse of Dimensionality"** occurs.
  - When dimensionality increases, data becomes increasingly sparse
  - Density and distance between points, which is critical to clustering, outlier analysis, becomes less meaningful

RollNo	Name	Mobile Number	Mobile Network
T4Tutorials1	Sameed	+92 302 XX XXX XX	Mobilink
T4Tutorials1	Ali	+92 333 XX XXX XX	Ufone

Figure 1: Before Dimension reduction

If we know Mobile Number, then we can know the Mobile Network. So we need to reduce the one dimension.

RollNo	Name	Mobile Number
T4Tutorials1	Sameed	+92 302 XX XXX XX
T4Tutorials1	Ali	+92 333 XX XXX XX

Figure 2: After Dimension reduction

- Common techniques for Dimension Reduction:
  1. **Principal Component Analysis (PCA)**: A statistical method that transforms data into a smaller set of dimensions while maintaining most of the original variance in the data.
  2. **Linear Discriminant Analysis (LDA)**: Used primarily for classification problems, LDA finds a linear combination of features that separates two or more classes.

b) Numerosity Reduction

- Numerosity reduction is a technique used in data mining **to reduce the number of data points in a dataset** while still preserving the most important information.
- In this method, the **representation of the data is made smaller** by reducing the volume. There will not be any loss of data in this reduction.
- This can be beneficial in situations where the dataset is too large to be processed efficiently, or where the dataset contains a large amount of irrelevant or redundant data points.
- There are several different numerosity reduction techniques that can be used in data mining:
  i. Data Aggregation:
  ii. Data Sampling
  iii. Clustering
  iv. Data Generalization
  v. Data Compression

- Common techniques for Numerosity Reduction:
- **Aggregation**: Summarizing data by combining or grouping related data points into a single representative value (e.g., averaging or summing data over time intervals). For eg: daily sales data can be aggregated into monthly or quarterly totals, reducing the number of data points while preserving important trends.
- **Sampling**: Selecting a representative subset of the data (e.g., random sampling, stratified sampling).
- **Generalization:** Data generalization focuses on replacing individual data points or records with a more compact representation. For instance, an individual age of "23" could be generalized to the range "20-30".
- Data Compression:

Data Cube Aggregation:

- Aggregation operation is **performed to combine two or more objects.**
- For example, the data consist of the sales per quarter, for the years 2002 to 2004.
- If the interest of user is in the annual sales (total per year) than the total per quarter, the data can be aggregated so that the resulting data summarize the total sales per year instead of per quarter.

2002		2003		2004	
Quarter	Sales	Quarter	Sales	Quarter	Sales
Q1	50,000	Q1	45,000	Q1	50,000
Q2	25,000	Q2	40,000	Q2	45,000
Q3	15,000	Q3	15,000	Q3	40,000
Q4	20,000	Q4	25,000	Q4	35,000

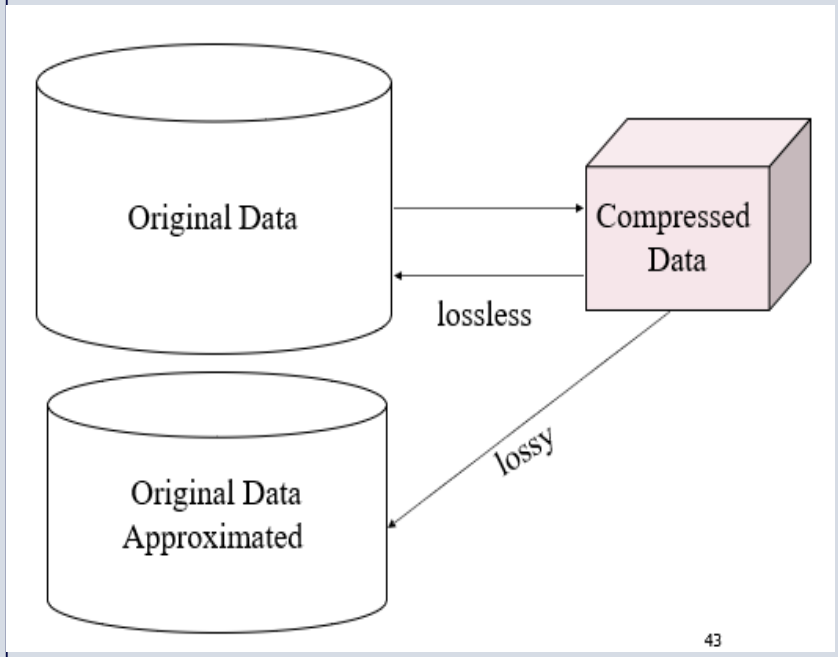
year	Sales
2002	110,000
2003	125,000
2004	170,000

- The resulting data set is smaller in volume, without loss of information necessary for the analysis task.

Data Compression:

Generally **done for space optimization** on mostly multimedia data

- **String compression**
  - There are extensive theories and well-tuned algorithms
  - Typically lossless, but only limited manipulation is possible without expansion
- **Audio/video compression**
  - Typically lossy compression, with progressive refinement
  - Sometimes small fragments of signal can be reconstructed without reconstructing the whole
- **Dimensionality and numerosity reduction** may also be considered as forms of data compression



2.7 Data Transformation

Data Transformation

- The **technique of changing the format or the structure of the data** is called data transformation.
- It is a function that maps the entire set of values of a given attribute to a new set of replacement values s.t. each old value can be identified with one of the new values.
- Some methods are:
  i. **Smoothing**: This works to **remove noise from the data**. Such techniques include binning, regression and clustering.
  ii. **Aggregation**: In this method, the data is stored and presented **in the form of a summary**. For example, the daily sales data may be aggregated so as to compute monthly and annual total amounts.
  iii. **Discretization**: The continuous data here is split into intervals. Discretization **reduces the data size**. For example, rather than specifying the class time, we can set an interval like (3 pm-5 pm, 6 pm-8 pm).
  iv. **Normalization:** It is the method of scaling the data so that it can be **represented in a smaller range.** Example ranging from -1.0 to 1.0.

Normalization techniques

i) Min-Max normalization

- This technique scales the values of a feature to a **range between 0 and 1.**
- This is done by subtracting the minimum value of the feature from each

$$v' = \frac{v - min_A}{max_A - min_A}$$

Eg. Let the income range is *12000* to *98000* be normalized to *[0.0, 1.0]*. Then *73600* is normalized to:

$$\frac{73600 - 12000}{98000 - 12000} = 0.716$$

ii) Z-score normalization:

- This technique normalizes the value of an attribute A **based on the mean and standard deviation of A.**
- This is done by subtracting the mean of the feature from each value, and then dividing by the standard deviation.

$$v' = \frac{v - mean_A}{\sigma_A}$$

Eg. Let *mean* = *54000* and *Standard Deviation* (σ) = *16000*. Then *73600* is mapped to

$$\frac{73600 - 54000}{16000} = 1.225$$

iii) Decimal Scale Normalization

- This technique scales the values of a feature by **dividing the values of a feature by a power of 10.**

$$v' = \frac{v}{10^j}$$

where j is the number of digits in the largest absolute value.

- Eg: Suppose that the recorded values of A ranges from -986 to 917. The maximum absolute value of A is 986. To normalize by decimal scaling, we therefore divide each value by 1000 (i.e j=3) so that -986 normalizes to -0.986 and 917 normalizes to 0.917.

Example 1: Min-Max normalization

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
# Example DataFrame
data = {
    "Feature1": [10, 20, 30, 40, 50],
    "Feature2": [100, 200, 300, 400, 500],
    "Feature3": [5, 15, 25, 35, 45],
}
df = pd.DataFrame(data)
# Initialize the MinMaxScaler
scaler = MinMaxScaler()
# Apply Min-Max Scaling
normalized_data = scaler.fit_transform(df)
# Convert the result back to a DataFrame
normalized_df = pd.DataFrame(normalized_data, columns=df.columns)
print("Original DataFrame:")
print(df)
print("\nNormalized DataFrame:")
print(normalized_df)
```

```
Original DataFrame:
   Feature1  Feature2  Feature3
0        10       100         5
1        20       200        15
2        30       300        25
3        40       400        35
4        50       500        45

Normalized DataFrame:
   Feature1  Feature2  Feature3
0      0.00      0.00      0.00
1      0.25      0.25      0.25
2      0.50      0.50      0.50
3      0.75      0.75      0.75
4      1.00      1.00      1.00
```

Example 2: Z-Score normalization

```
from sklearn.preprocessing import StandardScaler
import pandas as pd
# Example DataFrame
data = {
    "Feature1": [10, 20, 30, 40, 50],
    "Feature2": [100, 200, 300, 400, 500],
    "Feature3": [5, 15, 25, 35, 45],
}
df = pd.DataFrame(data)
print("Original dataframe:\n",df)
# Initialize the StandardScaler
scaler = StandardScaler()
# Apply Z-score Normalization
z_score_normalized_data = scaler.fit_transform(df)
# Convert the result back to a DataFrame
z_score_normalized_df = pd.DataFrame(z_score_normalized_data,
columns=df.columns)
print("\nZ-Score Normalized DataFrame (using sklearn):")
print(z_score_normalized_df)
```

```
Original DataFrame:
   Feature1  Feature2  Feature3
0        10       100         5
1        20       200        15
2        30       300        25
3        40       400        35
4        50       500        45

Z-Score Normalized DataFrame:
   Feature1  Feature2  Feature3
0 -1.264911 -1.264911 -1.264911
1 -0.632456 -0.632456 -0.632456
2  0.000000  0.000000  0.000000
3  0.632456  0.632456  0.632456
4  1.264911  1.264911  1.264911
```

Example 3: Decimal normalization

```
import pandas as pd
import numpy as np
# Example DataFrame
data = {
    "Feature1": [10, 20, 30, 40, 50],
    "Feature2": [100, 200, 300, 400, 500],
    "Feature3": [5, 15, 25, 35, 45],
}
df = pd.DataFrame(data)
# Decimal Normalization Function
def decimal_normalize(series):
    max_abs_val = series.abs().max()
    scaling_factor = 10 ** int(np.ceil(np.log10(max_abs_val)))
    return series / scaling_factor
# Apply the function
decimal_normalized_df = df.apply(decimal_normalize)
print("Decimal Normalized DataFrame:")
print(decimal_normalized_df)
```

```
Decimal Normalized DataFrame:
   Feature1  Feature2  Feature3
0       0.1       0.1      0.05
1       0.2       0.2      0.15
2       0.3       0.3      0.25
3       0.4       0.4      0.35
4       0.5       0.5      0.45
```

Assignment:

- Perform normalization for below data using:

a) Min-max normalization

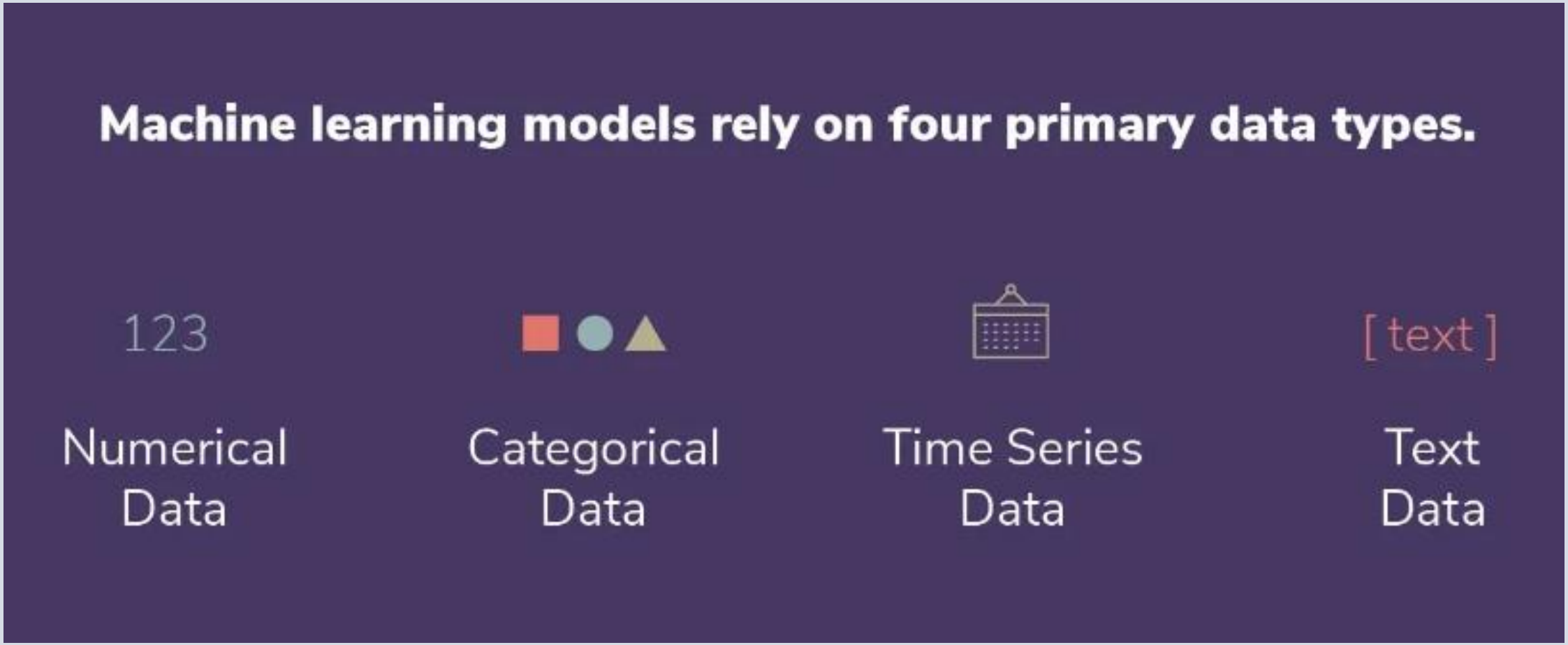
b) Decimal scale normalization

c) Z-score normalization

Q1	Q2	Q3	Q4
25000	12000	20000	6000
35000	14000	25000	12000
45000	16000	30000	24000
55000	18000	35000	28000

Handling Categorical Attributes

Primary data types



Categorical Attributes

- Categorical data refers to information that can be divided into groups or categories based on **qualitative** characteristics rather than quantitative values.
- Here are a few examples:
  - City: Delhi, Mumbai, Ahmedabad, Bangalore, etc.
  - Department: Finance, Human resources, IT, Production.
  - Degree: High school, Diploma, Bachelors, Masters, PhD.
  - Grades: A+, A, B+, B, B- etc.
- There are two kinds of categorical data-
  - **Ordinal Data:** The categories have an inherent order
  - **Nominal Data:** The categories do not have an inherent order

Handling Categorical Attributes

- Most machine learning algorithms cannot handle categorical variables unless we convert them to numerical values
- Since we are dealing with a mathematical model in machine learning, it is significant that we can **convert this category into numeric numbers** prior to utilizing it for training our model.

Different Approaches to Handle Categorical Data

1. Label Encoding
2. Ordinal Number Encoding
3. Binary Encoding
4. Count or Frequency Encoding
5. **One Hot Encoding**
6. One Hot Encoding with multiple categories
7. Target guided Ordinal Encoding
8. Mean Ordinal Encoding
9. Probability Ratio Encoding

i) Label Encoding

- One of the simplest and most common solutions advertised to transform categorical variables is **Label Encoding**.
- It consists of **substituting each group with a corresponding number** and keeping such numbering consistent throughout the feature.

Categorical Feature	Label Encoding
United States	1
United States	1
France	2
Germany	3
United Kingdom	4
France	2

State	Confirmed	Recovered	Deaths
Punjab	2000	1500	200
Haryana	2321	1222	345
Kerala	3455	2365	400

State
Punjab
Haryana
Kerala

State
0
1
2

ii) Ordinal Encoding

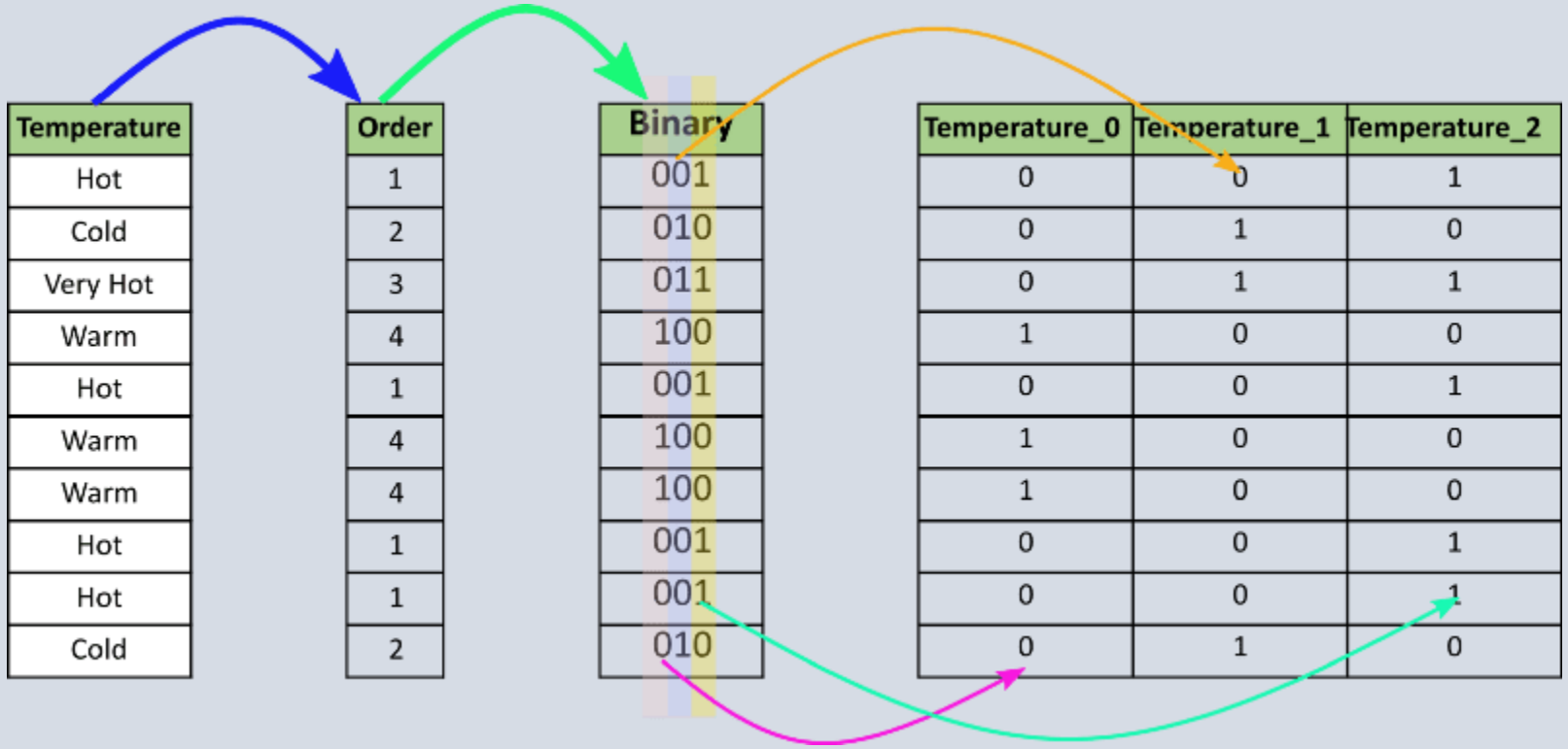
- Ordinal encoding's encoded variables retain the ordinal (ordered) nature of the variable.
- It looks **similar to label encoding**, the only difference being that ordinal encoding considers whether a variable is ordinal or not.

	temperature
0	very cold
1	cold
2	warm
3	hot
4	very hot

	temperature	temp_ordinal
0	very cold	1
1	cold	2
2	warm	3
3	hot	4
4	very hot	5

iii) Binary encoding

- Binary encoding **converts a category into binary digits.** Each binary digit creates one feature column.



Temperature
Hot
Cold
Very Hot
Warm
Hot
Warm
Warm
Hot
Hot
Cold

Order
1
2
3
4
1
4
4
1
1
2

Binary
001
010
011
100
001
100
100
001
001
010

Temperature_0	Temperature_1	Temperature_2
0	0	1
0	1	0
0	1	1
1	0	0
0	0	1
1	0	0
1	0	0
0	0	1
0	0	1
0	1	0

iv) Frequency encoding

- The category is assigned as **per the frequency of values** in its total lot.

	class	data_fe
0	A	0.27
1	B	0.13
2	C	0.27
3	D	0.13
4	A	0.27
5	B	0.13
6	E	0.20
7	E	0.20
8	D	0.13
9	C	0.27
10	C	0.27
11	C	0.27
12	E	0.20
13	A	0.27
14	A	0.27

v) One Hot Encoding

- **One-Hot Encoding** is the most common, correct way to deal with non-ordinal categorical data.
- This technique is applied for nominal categorical features.
- In one Hot Encoding method, each category value is converted into a new column and assigned a value as 1 or 0 to the column.

Index	Animal
0	Dog
1	Cat
2	Sheep
3	Horse
4	Lion

One-Hot code

Index	Dog	Cat	Sheep	Lion	Horse
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	0	1
4	0	0	0	1	0

End of Chapter