Unit-1 Fundamental of Data Analytics

Exploratory Data Analysis (EDA) is the process of examining and summarizing a dataset to understand its main characteristics before applying modeling or advanced analytics.

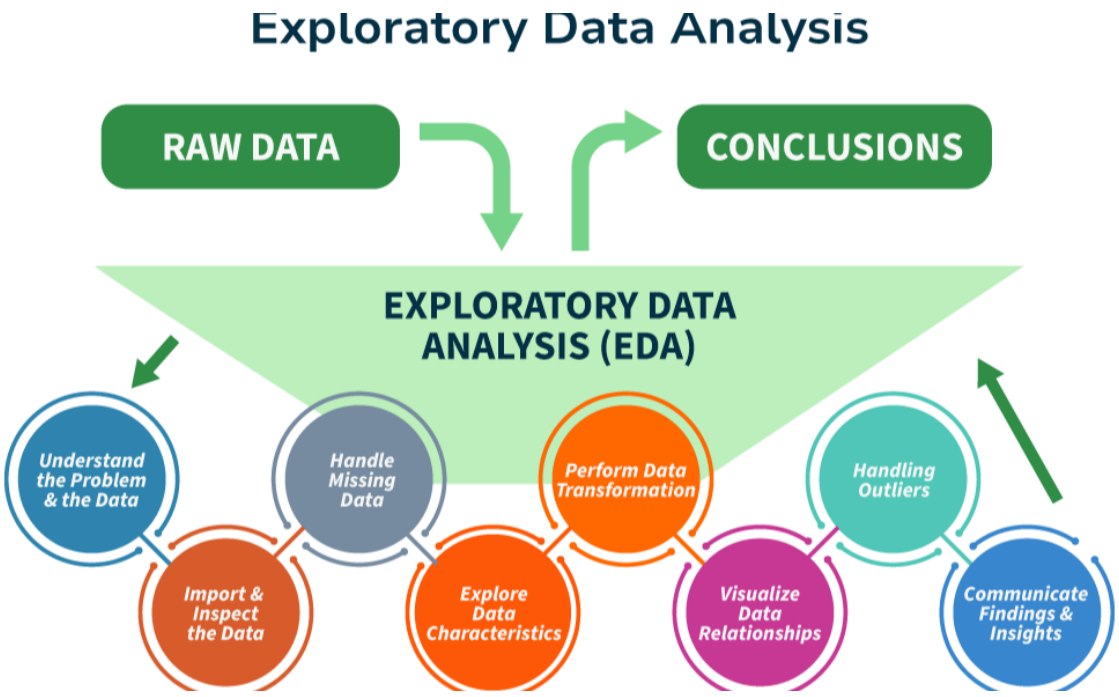
Exploratory Data Analysis (EDA) is the process of examining and summarizing data to understand its key characteristics before applying any advanced analytics or modeling. It involves using statistics, visualizations, and data profiling to learn what the data contains and how its variables relate to each other.

EDA (Exploratory Data Analysis) is the process of exploring and understanding a dataset through statistical summaries and visualizations.

EDA helps you **"get to know"** your data so you can make informed decisions about cleaning, transforming, and modeling it.

It uses statistical summaries, visualizations, and data profiling to uncover:

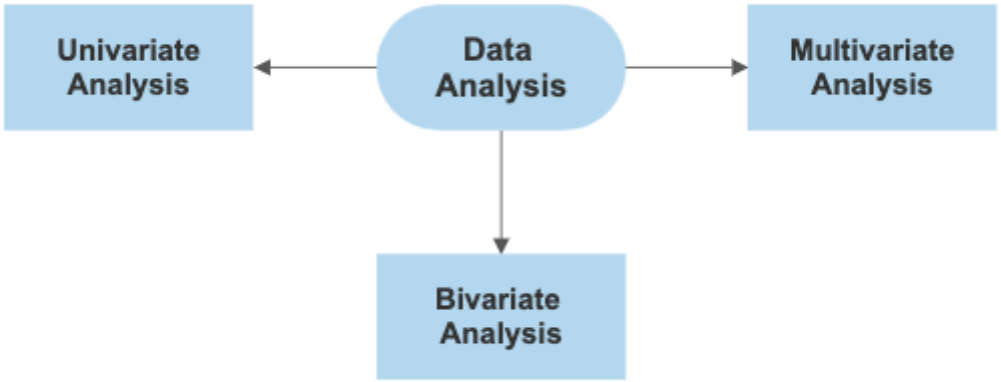
- Patterns and trends
- Missing or inconsistent data
- Relationships between variables
- Outliers or anomalies
- Initial insights that guide further analysis or model building



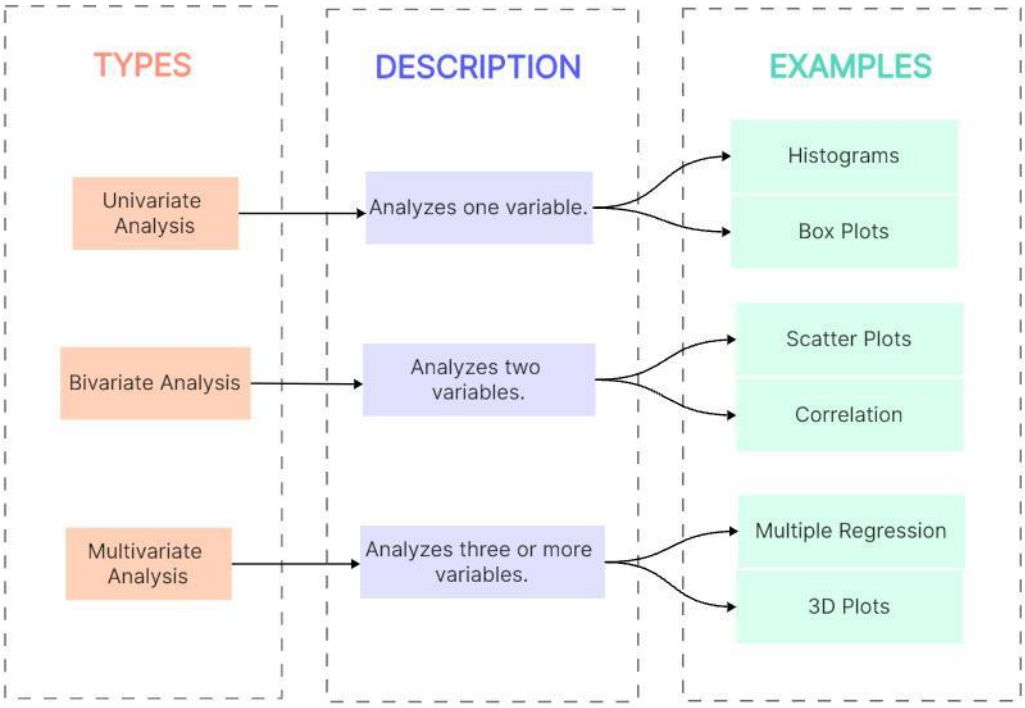
Common EDA techniques

- Summary statistics (mean, median, variance, etc.)
- Data visualization (histograms, box plots, scatter plots, heatmaps)
- Correlation analysis
- Distribution analysis
- Outlier detection

Types of EDA:



Understanding Univariate, Bivariate, and Multivariate Analysis in Data Science



Univariate Analysis is the statistical analysis of a **single variable**

(i.e., "uni" = one).

Goals:

- Understand the **central tendency**, **spread**, and **distribution**
- Identify **outliers**, **missing values**, and **patterns**
- Choose the right preprocessing techniques (e.g., binning, normalization)

Types of Variables:

Univariate analysis depends on the **type of variable**:

Variable Type	Examples	Analysis Type
Numerical	Age, Salary, Marks	Statistical + Visual
Categorical	Gender, City, Grade	Frequency + Visual

Univariate Analysis for Numerical Variables

Example: Age of Employees

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Sample data

```
data = pd.DataFrame({ "Age": [22, 25, 24, 29, 30, 23, 22, 45, 32, 41, 38, 27]})
```

Summary Statistics

```
print(data["Age"].describe())
```

Output:

```
count    12.000000
mean     30.250000
std       7.909809
min      22.000000
25%      23.250000
50%      27.500000
75%      32.750000
max      45.000000
```

Visualizations

Histogram

```
sns.histplot(data["Age"], bins=6, kde=True)
plt.title("Age Distribution")
plt.xlabel("Age")
plt.show()
```

Box Plot

```
sns.boxplot(x=data["Age"])
plt.title("Boxplot of Age")
```

```
plt.show()
```

Boxplots help detect outliers.
Histograms help understand the shape of distribution (normal, skewed, etc.)

Univariate Analysis for Categorical Variables

Example: Department

```
data = pd.DataFrame({ "Department": ["HR", "IT", "IT", "Sales", "HR", "IT", "Sales", "Sales", "IT"]})
```

Frequency Table

```
print(data["Department"].value_counts())
```

Bar Plot

```
sns.countplot(x="Department", data=data)

plt.title("Department Distribution")

plt.show()
```

Output:

```
IT      4
Sales   3
HR      2
```

Bar charts are great for visualizing categorical variable frequencies.

Bivariate Analysis

Bivariate analysis is a statistical method that explores the relationship between two variables.

Unlike univariate analysis, which focuses on a single variable, bivariate analysis examines the association between two variables.

The primary goal of bivariate analysis is to understand whether changes in one variable are associated with changes in another variable and to quantify the strength and direction of that association.

Bivariate Analysis - Key Aspects

Key aspects of bivariate analysis include -

1. **Types of Variables**: The two variables being studied can be of different types, such as:
    - **Categorical-Categorical:** Analyzing the relationship between two categorical variables.
    - **Categorical-Numerical:** Studying how a numerical variable differs across different categories.
    - **Numerical-Numerical:** Examining the correlation or association between two numerical variables.
2. **Visualization:** The analysis can be done using graphs plotting their types.
    - **Scatter plots:** Visualization of relationship between two numerical variables.
    - **Bar charts:** Comparing the distribution of a numerical variable across different categories.
    - **Cross-tabulation tables:** Summarize the joint frequencies of two categorical variables.
3. **Measures of Association:**
    - **Categorical-categorical relationship**: Use of measures like chi-squared tests or measures of association like Cramer's V.
    - **Numerical-numerical relationship:** Correlation coefficients like Pearson's correlation coefficient or Spearman's rank correlation coefficient can quantify the strength and direction of the association.
4. **Hypothesis Testing:** Bivariate analysis often involves testing hypotheses to determine whether the observed relationship is statistically significant or if it could have occurred by chance.
5. **Causality vs. Correlation:** It's essential to recognize that a significant association between two variables does not imply causation.
6. **Confounding Variables:** Bivariate analysis may uncover associations, but other variables (confounding variables) may influence the relationship.

7. **Interpretation:** Interpret the results to draw insights and make informed decisions. Consider the practical implications of the relationship between the two variables.

Performing Bivariate Analysis

To perform bivariate analysis, get the correlation coefficient using the **DataFrame.corr()** method which calculates the pairwise correlation of columns, excluding NA/null values.

Example

Python program to perform bivariate analysis using correlation coefficient.

```
Import pandas as pd
values = pd.DataFrame(  {
    "hours": [1, 1, 2, 2, 3, 3, 3, 3, 5, 5, 6, 6, 7, 8],
    "score": [75, 66, 78, 72, 85, 90, 82, 80, 90, 92, 94, 94, 91, 96],  })
print(f"The dataset is \n{values}")
corrCoef = values.corr()
print(f"The correlation Coefficient is \n{corrCoef}")
```

Output

The output of the above example is:

```
The dataset is
   hours  score
0      1     75
1      1     66
2      2     78
3      2     72
4      3     85
5      3     90
6      3     82
7      3     80
8      5     90
```

9 5 92

10 6 94

11 6 94

12 7 91

13 8 96

The correlation Coefficient is

hours score

hours 1.000000 0.885031

score 0.885031 1.000000

Multivariate Data Analysis (MVA)

Definition:
Multivariate data analysis is the examination of **three or more variables simultaneously** to understand relationships, patterns, or dependencies between them. It extends beyond univariate and bivariate analysis.

Purpose in EDA:

- Identify complex interactions between variables
- Detect patterns that involve multiple features
- Inform feature selection for modeling
- Detect multicollinearity (high correlation among features)

Techniques in Multivariate EDA

Technique	Description	Example
Correlation Matrix	Shows pairwise correlation of numeric variables	Heatmap
Pair Plot	Scatter plots of all numeric variables	sns.pairplot()
Grouped Analysis	Compare a numeric variable across two categorical variables	Boxplot, violin plot

Technique	Description	Example
PCA / Dimensionality Reduction	Reduce high-dimensional data for visualization	sklearn.decomposition.PCA

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Load dataset

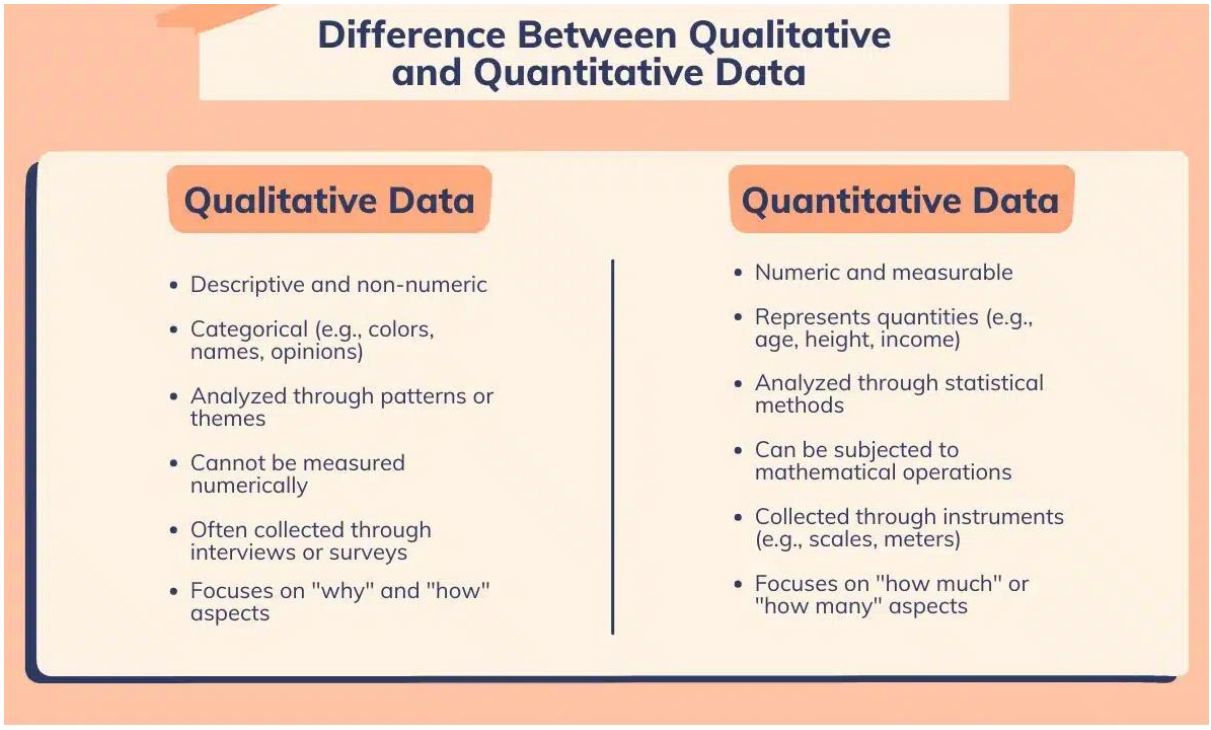
```
data = pd.read_csv("your_dataset.csv")
```

Correlation heatmap

```
sns.heatmap(data.corr(), annot=True, cmap="coolwarm")
plt.show()
```

Understanding The Data:

- Qualitative data
- Quantitative data



Qualitative data (also called **categorical data**)

It represents categories or labels rather than numeric quantities.
Examples:

- Gender (Male, Female)
- City (Delhi, Mumbai, Kolkata…)
- Product Type (Electronics, Clothing, Grocery)
- Satisfaction Level (Low, Medium, High)

During EDA, the goal is to **understand distribution, frequency, patterns, and relationships** involving categorical variables.

Types of Qualitative Data

a) Nominal (No order)

- Categories have **no natural ranking**.
  Example: *Color = Red, Blue, Green*

b) Ordinal (Ordered categories)

- Categories have a **rank order**, but not numeric distance.
  Example: *Satisfaction = Low, Medium, High*

Import libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Example dataset

```
data = {
   "Customer": ["C1", "C2", "C3", "C4", "C5"],
   "City": ["Delhi", "Mumbai", "Delhi", "Kolkata", "Delhi"],
   "Satisfaction": ["High", "Medium", "Low", "High", "Medium"]}
df = pd.DataFrame(data)
print("Dataset:")
print(df)
```

1. Frequency Distribution (Counts)

python

Frequency count of City

print("\nCity Frequency:")

print(df['City'].value_counts())

Frequency count of Satisfaction

print("\nSatisfaction Frequency:")

print(df['Satisfaction'].value_counts())

2. Bar Charts

plt.figure(figsize=(6,4))

sns.countplot(data=df, x='City')

plt.title("City Distribution")

plt.show()

plt.figure(figsize=(6,4))

sns.countplot(data=df, x='Satisfaction', order=['Low', 'Medium', 'High'])

plt.title("Satisfaction Distribution")

plt.show()

ORDINAL EDA (Example: Satisfaction)

Ordinal variables have **meaningful order**, so give them a custom category order.

ORDINAL EDA (Example: Satisfaction)

Ordinal variables have meaningful order, so give them a custom category order.

Convert to Ordered Category

order = ["Low", "Medium", "High"]

df["Satisfaction"] = pd.Categorical(df["Satisfaction"], categories=order, ordered=True)

print(df["Satisfaction"])

Quantitative Data

Quantitative data is **numerical data** — information that can be **measured**, **counted**, and expressed in **numbers**.
It answers questions like **"how much?"**, **"how many?"**, or **"how often?"**.

Examples of Quantitative Data

- Age: 18, 20, 25
- Salary: 30,000; 45,000
- Height: 165.5 cm
- Marks: 85, 92, 71
- Number of students: 50, 55

Discrete Data

✔ Meaning:

Data that can be **counted**.
Takes **whole numbers only** (no decimals).

✔ Characteristics:

- Finite or countable values
- Gaps between numbers
- Comes from **counting** things

✔ Examples:

- Number of students (30, 31, 32…)
- Number of cars (5, 10, 12…)
- Number of defects in a product
- Number of calls received

```
# --------- Discrete Data EDA Program ---------

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
import seaborn as sns

# Example of Discrete Data: Number of products sold per day
products_sold = [10, 12, 12, 15, 15, 15, 18, 20, 20, 22, 22, 22, 25]

# Convert to DataFrame
df = pd.DataFrame({"Products_Sold": products_sold})

# ---------- Basic Statistics ----------
print("Basic Statistics for Discrete Data")
print(df["Products_Sold"].describe())

# ---------- Frequency Count ----------
freq = df["Products_Sold"].value_counts()
print("\nFrequency Count:")
print(freq)

# ---------- Bar Chart (best for discrete data) ----------
plt.figure(figsize=(6,4))
sns.countplot(x=df["Products_Sold"])
plt.title("Bar Chart - Discrete Data (Products Sold)")
plt.xlabel("Products Sold")
plt.ylabel("Frequency")
plt.show()

# ---------- Histogram ----------
plt.figure(figsize=(6,4))
plt.hist(df["Products_Sold"], bins=6, edgecolor='black')
plt.title("Histogram - Discrete Data")
plt.xlabel("Products Sold")
plt.ylabel("Count")
plt.show()

# ---------- Boxplot ----------
plt.figure(figsize=(6,4))
sns.boxplot(x=df["Products_Sold"])
plt.title("Boxplot - Discrete Data")
plt.xlabel("Products Sold")
plt.show()
```

Continuous Data

✔ Meaning:

Data that can be **measured** and can take **any value** (including decimals).

✔ Characteristics:

- Infinite possibilities within a range
- Smooth, no gaps
- Comes from **measurement**

✔ Examples:

- Height (165.3 cm)
- Weight (62.75 kg)
- Temperature (36.8°C)
- Time (1.25 hours)
- Salary (45000.50)

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

# Continuous data
heights = [160.5, 162.3, 158.8, 170.2, 165.9, 172.4, 168.0, 159.7]

# Summary
print("Mean:", np.mean(heights))
print("Median:", np.median(heights))
print("Std Dev:", np.std(heights))

# Histogram (best for continuous data)
plt.hist(heights, bins=5)
plt.xlabel("Height (cm)")
plt.ylabel("Frequency")
plt.title("Continuous Data - Height Distribution")
plt.show()

# Density plot (smooth curve)
sns.kdeplot(heights, fill=True)
plt.title("Density Plot - Continuous Data")
plt.show()
```

Spread of Data:

The spread of a data set refers to how much the values in the set vary or are dispersed.

It provides insights into the variability and consistency of the data. Common measures to calculate the spread include range, variance, and standard deviation.

1. **Normal Distribution**
2. **Skewed Distribution**
3. **Skewness and Kurtiosis**

1. Normal Distribution

A normal distribution is a bell-shaped curve where most values cluster around the mean, and the spread is controlled by the standard deviation (SD).

✓ **Mean = center of the data**
✓ **Standard deviation = spread of the data**

A **normal distribution** (also called **Gaussian distribution** or **bell curve**) is a probability distribution where most values are **clustered around the mean**, and the frequencies gradually decrease on both sides.

It is one of the most important concepts in statistics and EDA.

How to Check Normal Distribution in EDA

1. Histogram

Shows the shape of distribution.

2. Density Plot

Smooth curve showing distribution.

3. Q–Q Plot (Quantile–Quantile Plot)

Checks how close data is to a normal distribution.

4. Skewness and Kurtosis

- Skewness ≈ 0 → symmetrical

- Kurtosis ≈ 3 → normal (mesokurtic)

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
# Generate sample normal distribution data
data = np.random.normal(loc=50, scale=10, size=500)
# Histogram
plt.hist(data, bins=20)
plt.title("Histogram - Normal Distribution")
plt.show()
# Density Plot
sns.kdeplot(data, fill=True)
plt.title("Density Plot")
plt.show()

# Q-Q Plot
stats.probplot(data, dist="norm", plot=plt)
plt.title("Q-Q Plot")
plt.show()
```

2) Skewed Distribution

In EDA, a **skewed distribution** is a distribution where the data is **not symmetric**.

Instead of forming a perfect bell shape (normal distribution), the data leans more to one side.

Skewness tells us **which side the tail of the distribution is longer**.

Types of Skewness

1. Positive Skew (Right Skew)

- Long tail on the **right** side
- Most values are **clustered on the left**
- Mean > Median

Examples:

- Income
- House prices
- Sales amounts

Visual shape:

Tail → ➡ RIGHT

2. Negative Skew (Left Skew)

- Long tail on the **left** side
-
- Most values are **clustered on the right**
- Mean < Median

Examples:

- Age at retirement
- High exam scores (where most people score well)

Visual shape:

Tail → LEFT

```
import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd
```

```
# Example of right-skewed data
right_skewed = np.random.exponential(scale=2, size=500)


# Example of left-skewed data
left_skewed = -np.random.exponential(scale=2, size=500) + 10
df = pd.DataFrame({"Right_Skewed": right_skewed, "Left_Skewed": left_skewed})
# Check skewness values
print(df.skew())
# Plot right skew
sns.histplot(right_skewed, bins=20, kde=True)
plt.title("Right-Skewed Distribution")
plt.show()
# Plot left skew
sns.histplot(left_skewed, bins=20, kde=True)
plt.title("Left-Skewed Distribution")
plt.show()
```

3) Skewness and Kurtosis

In **Exploratory Data Analysis (EDA)**, skewness and kurtosis help us understand the **shape of a numerical distribution**.

1. Skewness

Skewness tells us **how asymmetric** the data distribution is.

✔ Types of Skewness

a) Positive Skew (Right Skew)

- Tail on **right** side is longer
- Most values are small
- **Mean > Median**

b) Negative Skew (Left Skew)

- Tail on **left** side is longer
- Most values are large
- **Mean < Median**

✔ Skewness values interpret

Skewness values interpretation

Skewness	**Interpretation**
0	Perfectly symmetrical
> 0	Right-skewed
< 0	Left-skewed
Between -0.5 to +0.5	Approximately symmetric
> +1 or < -1	Highly skewed

Kurtosis

Kurtosis measures the **tailedness** or **peakedness** of a distribution.

It tells how **heavy or light the tails** are compared to a normal distribution.

✔ Types of Kurtosis

a) Mesokurtic (Kurtosis ≈ 3)

- Normal distribution
- Medium tails

b) Leptokurtic (Kurtosis > 3)

- Heavy tails
- More outliers
- Sharper peak

c) Platykurtic (Kurtosis <

- Light tails
- Fewer outliers
- Flatter peak

Property	Skewness	Kurtosis
Measures	Asymmetry	Tailedness / Peakedness
Value	+ve, –ve, 0	>3, =3, <3
Focus	Left/Right tail	Tail heaviness & outliers
Good For	Detecting direction of skew	Detecting extreme outliers

```
import pandas as pd
import numpy as np
# sample data
data = np.random.normal(50, 10, 500)  # normal distribution

df = pd.DataFrame({"Values": data})

print("Skewness:", df["Values"].skew())
print("Kurtosis:", df["Values"].kurt())
```

Unit-2: Automate EDA (Exploratory Data Analysis)

2.1 Python Libraries to Automate Exploratory Data Analysis

2.1.1 Pandas and NumPy

Pandas

- Pandas is a Python library used for data manipulation and analysis.
- Provides two major data structures:
  - **Series** (1-D labeled array)
  - **DataFrame** (2-D table with rows and columns)
- Features:
  - Data cleaning (handling missing values)
  - Easy filtering, grouping, merging, joining
  - Built-in methods for descriptive statistics
- Automates EDA tasks like:
  - .info(), .describe(), .value_counts(), .corr()

NumPy

- NumPy is used for numerical computations.
- Provides:
  - N-dimensional arrays (ndarray)
  - Fast mathematical operations (vectorized operations)
  - Linear algebra, random number generation
- Acts as the foundation for many libraries like Pandas, SciPy, and ML frameworks.

Step-by-Step Example Using Pandas and NumPy

Step 1: Import Libraries

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

Step 2: Create Data Using NumPy

```
# creating an array
data = np.array([10, 20, 30, 40, 50])
print("NumPy Array:", data)
```

Step 3: Convert to Pandas DataFrame

```
df = pd.DataFrame({
   'Marks': data,
   'Age': [15, 16, 15, 17, 16]
})
df
```

Step 4: Basic EDA Using Pandas

```
# show first rows
print(df.head())
# summary statistics
print(df.describe())
# check missing values
print(df.isnull().sum())
# correlation matrix
print(df.corr())
```

2.2 REGRESSION

Regression is a statistical method used to **model the relationship** between a **dependent variable (Y)** and one or more **independent variables (X)**.
Its main purpose is **prediction**, **forecasting**, and **understanding relationships** between variables.

2.2.1 Characteristics of Regression

1. Shows Relationship Between Variables

Regression tells how changes in the independent variable (X) affect the dependent variable (Y).
Example:

- Hours studied (X) → Marks obtained (Y)

2. Predictive Ability

Regression helps predict future values.
Example: Predicting house prices or salary.

3. Uses Continuous Output

Regression outputs a **continuous value**.
(Not categories like "yes/no")

4. Measures Strength & Direction

Regression gives:

- **Slope (β1)** → How strongly X affects Y
- **Intercept (β0)** → Value of Y when X = 0

5. Based on Statistical Foundations

Uses:

- Least squares method
- Line fitting
- Error minimization

2.2.2 Dependent and Independent Variables

Dependent Variable (Y)

- Also called **output**, **response**, or **target**
- The variable you want to **predict**
- Depends on the independent variable

Example:

- Predicting salary → Salary = Dependent variable

Independent Variable (X)

- Also called **predictor**, **input**, or **feature**
- These variables **affect** the dependent variable

Example:

- Experience affects salary → Experience = Independent variable

Example of Dependent and Independent Variables

```
import pandas as pd

data = {
   'Experience': [1, 2, 3, 4],
   'Salary': [25000, 30000, 35000, 40000]
}

df = pd.DataFrame(data)

X = df[['Experience']]    # Independent variable
Y = df['Salary']          # Dependent variable

print("Independent Variable (X):")
print(X)

print("\nDependent Variable (Y):")
print(Y)
```

2.2.3 Covariance and Correlation

Covariance

- Measures how two variables change **together**

- Positive covariance → Variables increase together
- Negative covariance → One increases while the other decreases
- Value is **not standardized**

◆ Formula:

$$\text{Cov}(X,Y) = \frac{\sum(X-\bar{X})(Y-\bar{Y})}{n-1}$$

Correlation

- Measures **strength** and **direction** of linear relationship
- Standardized value between **–1 to +1**

◆ Values:

- +1 → Perfect positive
- –1 → Perfect negative
- 0 → No linear relationship

◆ Formula:

$$\text{Correlation} = \frac{\text{Cov}(X,Y)}{\sigma_X \cdot \sigma_Y}$$

◆ Example of Covariance & Correlation

```
import pandas as pd

data = {
   'Hours': [1, 2, 3, 4, 5],
   'Marks': [40, 50, 60, 65, 80]
}

df = pd.DataFrame(data)

print("Covariance:")
print(df.cov())

print("\nCorrelation:")
print(df.corr())
```

2.3 Machine Learning Basics

2.3.1 Concepts of Machine Learning

Machine Learning (ML) is a branch of Artificial Intelligence (AI) that focuses on building systems that learn from data and improve automatically without being explicitly programmed for every task.

2.3.1.1 Understanding Machine Learning

Machine Learning is the process by which computers:

- Learn patterns from data
- Make predictions or decisions
- Improve performance with experience

Instead of writing fixed rules, we train models using data.

Example:

- Email spam filter learns from past emails
- Recommendation systems (YouTube, Netflix) suggest content based on user behavior
- Face recognition systems identify people from images

2.3.1.2 Benefits of Machine Learning

Machine Learning offers several advantages:

1. Automation of Tasks

Machine Learning automates repetitive and manual tasks that normally require human effort. Once trained, ML systems can work continuously without human intervention. This saves time, reduces labor cost, and increases productivity.

Example:
Automated email filtering and customer support chatbots.

---

2. Improved Accuracy and Performance

Machine Learning models learn from historical data and continuously improve their performance. As more data is provided, the system becomes more accurate and reliable compared to traditional programs.

Example:
Medical diagnosis systems that improve accuracy with more patient data.

---

3. Handling Large and Complex Data

Machine Learning can efficiently process huge volumes of structured and unstructured data such as images, videos, text, and audio. Traditional methods struggle to analyze such data.

Example:
Analyzing social media data and image recognition systems.

---

4. Fast and Efficient Decision Making

ML systems analyze data quickly and provide real-time results. This helps organizations make fast and data-driven decisions.

Example:
Fraud detection systems in banking that identify suspicious transactions instantly.

---

5. Adaptability and Continuous Learning

Machine Learning models can adapt to changes in data patterns. They update themselves when new data is available, ensuring long-term effectiveness.

Example:
Recommendation systems that adjust based on user preferences.

---

6. Cost Efficiency

By automating processes and reducing errors, Machine Learning helps lower operational costs in the long run.

Example:
Predictive maintenance in industries to prevent equipment failure.

---

7. Wide Range of Applications

Machine Learning is used in various domains such as healthcare, finance, education, transportation, and entertainment, making it highly versatile.

2.3.1.3 Machine Learning Life Cycle

The Machine Learning Life Cycle describes the steps involved in building an ML system.

1. Data Collection

- The first step in the ML life cycle
- Data is gathered from sources like databases, sensors, websites, or user inputs
- The quality of data directly affects model performance

Example:
Customer data collected for predicting buying behavior

---

2. Data Preparation

- Raw data is cleaned and organized
- Missing values, errors, and duplicate data are removed
- Data is converted into a suitable format for training

Purpose:
To make data accurate and usable for machine learning models

---

3. Model Selection

- A suitable machine learning algorithm is chosen
- Selection depends on the type of problem (classification, prediction, etc.)

Examples:

- Linear Regression
- Decision Trees
- Neural Networks

---

4. Model Training

- The prepared data is given to the model
- The model learns patterns and relationships from data
- This step builds the actual machine learning model

Result:
A trained model capable of making predictions

5. Testing and Evaluation

- The model is tested using unseen data
- Accuracy and performance are measured
- Helps identify errors and improve the model

Common metrics:
Accuracy, precision, recall

---

6. Deployment

- The trained and tested model is implemented in real applications
- Users can now interact with the model

Examples:

- Recommendation systems
- Fraud detection systems

---

7. Monitoring and Improvement

- Model performance is continuously monitored
- New data is used to retrain and improve the model
- Ensures long-term accuracy and reliability

2.4 Types of Machine Learning

Machine Learning can be classified into different types based on how the model learns from data. The two main types are **Supervised Learning** and **Unsupervised Learning**.

2.4.1 Supervised and Unsupervised Learning

1. Supervised Learning

Supervised Learning is a type of machine learning where the model is trained using labeled data. Each input is provided with a correct output, so the model learns the relationship between input and output. It is mainly used for classification and regression problems.

Examples:

- Spam email detection
- Predicting house prices
- Student result prediction

Python Program (Supervised Learning – Linear Regression)

```
from sklearn.linear_model import LinearRegression

# Training data

X = [[1], [2], [3], [4]]

y = [2, 4, 6, 8]

# Create and train model

model = LinearRegression()

model.fit(X, y)

# Prediction

print(model.predict([[5]]))
```

2. Unsupervised Learning

Unsupervised Learning is a type of machine learning where the model is trained using **unlabeled data**. The system finds hidden patterns or groups in the data without knowing the output in advance. It is mainly used for **clustering** and **pattern discovery**.

Examples:

- Customer segmentation
- Grouping similar images
- Market analysis

Python Program (Unsupervised Learning – K-Means Clustering)

```
from sklearn.cluster import KMeans
# Input data
X = [[1,2], [1,4], [1,0], [10,2], [10,4], [10,0]]
# Create model
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
# Output clusters
print(kmeans.labels_)
```

Difference Between Supervised and Unsupervised Learning

	Supervised Learning	Unsupervised Learning
Data Type	Labeled data (input + output)	Unlabeled data (input only)
Goal	Predict outcomes or classify data	Discover hidden patterns or groups
Output	Known / predefined	Unknown / discovered by model
Tasks	Classification, Regression	Clustering, Association
Accuracy	Usually higher (trained with correct output)	Can be lower (no labels)
Examples	Spam email detection, Predicting house prices	Customer segmentation, Grouping similar products

2.4.2 Applications of Machine Learning in Real-World Scenarios

Machine Learning is widely used in many real-world applications:

1. Healthcare

- Disease Diagnosis: ML models detect diseases like cancer, diabetes, and heart conditions from patient data.
- Medical Image Analysis: ML analyzes X-rays, MRIs, or CT scans to assist doctors.
- Patient Monitoring: Predicts health risks and alerts healthcare providers.

Example:
IBM Watson Health uses ML to help doctors diagnose diseases faster.

---

2. Finance

- Fraud Detection: Detects suspicious financial transactions in real-time.
- Credit Scoring: Assesses creditworthiness using past financial data.
- Stock Market Prediction: Predicts market trends using historical stock data.

Example:
Banks use ML algorithms to detect credit card fraud and prevent losses.

---

3. Education

- Student Performance Analysis: Identifies students who need extra help.
- Personalized Learning: Recommends study materials based on student behavior.

Example:
Learning platforms like Coursera or Khan Academy suggest courses based on student progress.

---

4. Transportation

- Self-Driving Cars: ML is used for navigation, obstacle detection, and decision-making.
- Traffic Prediction: Predicts congestion to suggest optimal routes.
- Route Optimization: Helps delivery services and ride-sharing apps plan efficient paths.

Example:
Tesla's Autopilot uses ML to drive safely and detect obstacles.

5. Entertainment

- Recommendation Systems: Suggest movies, videos, or music based on user preferences.
- Game Development: ML enhances AI opponents or user experience.

Example:
Netflix recommends shows using ML based on viewing history.

---

6. Retail and E-Commerce

- Product Recommendations: Suggests products to users based on past purchases.
- Customer Behavior Analysis: Helps companies design marketing strategies.

Example:
Amazon recommends products using collaborative filtering ML algorithms.

---

7. Industry and Manufacturing

- Predictive Maintenance: Predicts when machines may fail, reducing downtime.
- Quality Control: Detects defective products in manufacturing lines.

Example:
Siemens uses ML to predict failures in industrial machines and optimize production.

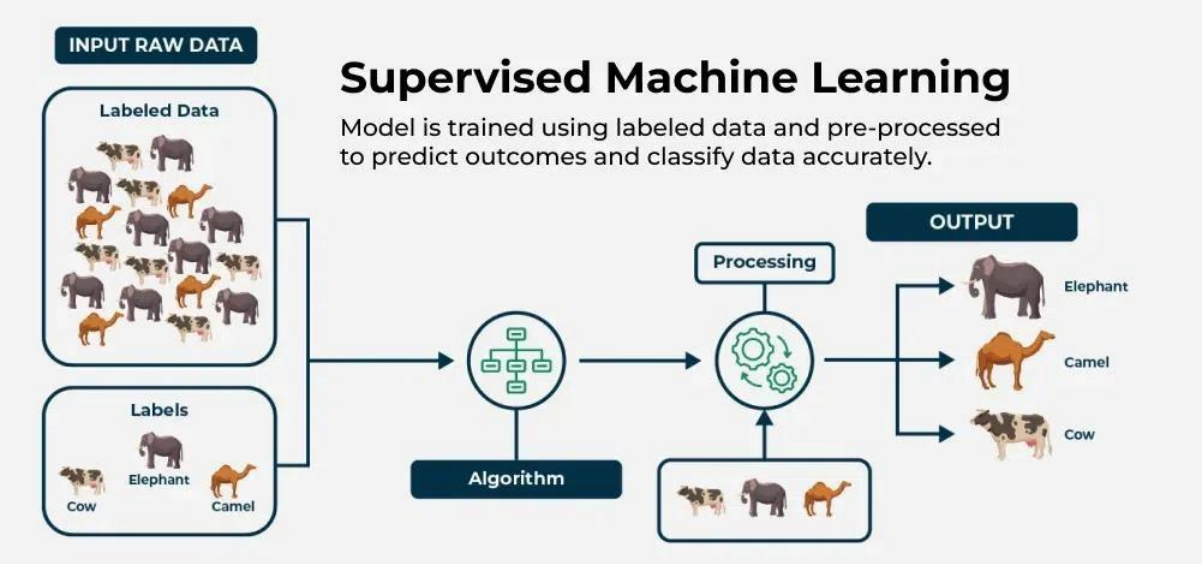
3. Understanding Supervised Learning

3.1 Overview of Supervised Learning

3.1.1 Concepts of Supervised Learning

Supervised learning is a type of machine learning where a model learns from labelled data—meaning every input has a corresponding correct output. The model makes predictions and compares them with the true outputs, adjusting itself to reduce errors and improve accuracy over time. The goal is to make accurate predictions on new, unseen data. For example, a model trained on images of handwritten digits can recognise new digits it has never seen before.

Supervised Learning is a type of Machine Learning where the model is trained using labeled data.
Each input in the dataset is associated with a correct output (label), and the goal of the algorithm is to learn a mapping from inputs to outputs.



Types

Classification: Where the output is a categorical variable (e.g., spam vs. non-spam emails, yes vs. no).

Regression: Where the output is a continuous variable (e.g., predicting house prices, stock prices).

Key Characteristics

- Uses labeled datasets
- Learning happens under the supervision of known outcomes
- Used for prediction and classification tasks
- Model performance can be measured easily

General Workflow

1. Collect labeled data
2. Split data into training and testing sets
3. Train the model
4. Evaluate model performance
5. Make predictions on new data

Example 1

Study Hours	Exam Score
2	40
4	60
6	75

Here, **Study Hours = Input**, **Exam Score = Output (Label)**

Example 2

User ID	Gender	Age	Salary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	1
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	1
15728773	Male	27	58000	1
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	1
15727311	Female	35	65000	0
15570769	Female	26	80000	1
15606274	Female	26	52000	0
15746139	Male	20	86000	1
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1

Figure A: CLASSIFICATION

Temperature	Pressure	Relative Humidity	Wind Direction	Wind Speed
10.69261758	986.882019	54.19337313	195.7150879	3.278597116
13.59184184	987.8729248	48.0648859	189.2951202	2.909167767
17.70494885	988.1119385	39.11965597	192.9273834	2.973036289
20.95430404	987.8500366	30.66273218	202.0752869	2.965289593
22.9278274	987.2833862	26.06723423	210.6589203	2.798230886
24.04233986	986.2907104	23.46918024	221.1188507	2.627005816
24.41475295	985.2338867	22.25082295	233.7911987	2.448749781
23.93361956	984.8914795	22.35178837	244.3504333	2.454271793
22.68800023	984.8461304	23.7538641	253.0864716	2.418341875
20.56425726	984.8380737	27.07867944	264.5071106	2.318677425
17.76400389	985.4262085	33.54900114	280.7827454	2.343950987
11.25680746	988.9386597	53.74139903	68.15406036	1.650191426
14.37810685	989.6819458	40.70884681	72.62069702	1.553469896
18.45114201	990.2960205	30.85038484	71.70604706	1.005017161
22.54895853	989.9562988	22.81738811	44.66042709	0.264133632
24.23155922	988.796875	19.74790765	318.3214111	0.329656571

Figure B: REGRESSION

Figure A: It is a dataset of a shopping store that is useful in predicting whether a customer will purchase a particular product under consideration or not based on his/her gender, age and salary.

Input: Gender, Age, Salary

Output: Purchased i.e. 0 or 1; 1 means yes the customer will purchase and 0 means that the customer won't purchase it.

Figure B: It is a Meteorological dataset that serves the purpose of predicting wind speed based on different parameters.

Input: Dew Point, Temperature, Pressure, Relative Humidity, Wind Direction

Output: Wind Speed

Working of Supervised Machine Learning

The working of supervised machine learning follows these key steps:

1. Collect Labeled Data

Gather a dataset where each input has a known correct output (label).

Example: Images of handwritten digits with their actual numbers as labels.

2. Split the Dataset

Divide the data into training data (about 80%) and testing data (about 20%).

The model will learn from the training data and be evaluated on the testing data.

3. Train the Model

Feed the training data (inputs and their labels) to a suitable supervised learning algorithm (like Decision Trees, SVM or Linear Regression).

The model tries to find patterns that map inputs to correct outputs.

4. Validate and Test the Model

Evaluate the model using testing data it has never seen before.

The model predicts outputs and these predictions are compared with the actual labels to calculate accuracy or error.

5. Deploy and Predict on New Data

Once the model performs well, it can be used to predict outputs for completely new, unseen data.

Supervised Machine Learning Algorithms

Supervised learning can be further divided into several different types, each with its own unique characteristics and applications. Here are some of the most common types of supervised learning algorithms:

Linear Regression: Linear regression is a type of supervised learning regression algorithm that is used to predict a continuous output value. It is one of the simplest and most widely used algorithms in supervised learning.

Logistic Regression: Logistic regression is a type of supervised learning classification algorithm that is used to predict a binary output variable.

Decision Trees : Decision tree is a tree-like structure that is used to model decisions and their possible consequences. Each internal node in the tree represents a decision, while each leaf node represents a possible outcome.

Random Forests: Random forests again are made up of multiple decision trees that work together to make predictions. Each tree in the forest is trained on a different subset of the input features and data. The final prediction is made by aggregating the predictions of all the trees in the forest.

Support Vector Machine(SVM): The SVM algorithm creates a hyperplane to segregate n-dimensional space into classes and identify the correct category of new data points. The extreme cases that help create the hyperplane are called support vectors, hence the name Support Vector Machine.

K-Nearest Neighbors: KNN works by finding k training examples closest to a given input and then predicts the class or value based on the majority class or average value of these neighbors. The performance of KNN can be influenced by the choice of k and the distance metric used to measure proximity.

Gradient Boosting: Gradient Boosting combines weak learners, like decision trees, to create a strong model. It iteratively builds new models that correct errors made by previous ones.

Naive Bayes Algorithm: The Naive Bayes algorithm is a supervised machine learning algorithm based on applying Bayes' Theorem with the "naive" assumption that features are independent of each other given the class label

Practical Examples of Supervised learning

is the most common form of machine learning. Think of it like a student learning from a teacher who provides both the **questions** and the **answers**.

Step 1: Data Collection (The Study Material)

You first need a dataset that contains "Labels" (the answers).

- **Features (X):** The input variables (e.g., square footage of a house).
- **Labels (y):** The output/answer (e.g., the price of the house).

Step 2: Data Preprocessing

Raw data is often messy. In this step, you:

- Clean out missing values.
- Convert text into numbers (e.g., "Male/Female" becomes "0/1").
- Scale the numbers so they are in a similar range.

Step 3: Splitting the Data (Train vs. Test)

You must split your data to ensure your model can handle new, unseen information.

- **Training Set (80%):** Used to teach the model.
- **Testing Set (20%):** Used to act as a "final exam" to see how accurate the model is.

Step 4: Choose an Algorithm

Depending on your goal, you pick a "brain" for your data:

- **Regression:** If you want to predict a **number** (e.g., stock price).
- **Classification:** If you want to predict a **category** (e.g., Spam or Not Spam).

Step 5: Training the Model (.fit)

This is the heart of machine learning. You feed the training data into the algorithm. The model looks at the features and the answers and identifies the mathematical patterns connecting them.

Step 6: Evaluation (.predict & Scoring)

You give the model the **Testing Set** (but hide the answers). Once the model makes its guesses, you compare those guesses to the real answers to get an **Accuracy Score**.

Step 7: Tuning and Deployment

If the accuracy is low, you go back and change settings (Hyperparameters) or provide better data. Once satisfied, the model is ready to predict real-world data.

Summary Workflow

Step	Action	Python Tool (Scikit-Learn)
Prepare	Load and clean data	pandas
Split	Divide into Train/Test	train_test_split
Train	Teach the model	model.fit(X_train, y_train)
Test	Make predictions	model.predict(X_test)
Score	Check accuracy	accuracy_score or R2_score

supervised learning is divided into two main categories: **Classification** and **Regression**. The difference lies in the type of output you are trying to predict.

1. Regression (Predicting a Quantity)

Regression is used when the output variable is a **continuous or numerical value**. You are trying to find the relationship between variables to predict a specific number.

- **Real-World Example: House Price Prediction.** Based on features like square footage, number of rooms, and location, a model predicts the exact price of a house (e.g., $450,000).
- **Other Examples:** Predicting stock prices, temperature, or age.

Python Code for Regression

We use LinearRegression from Scikit-Learn.

```
from sklearn.linear_model import LinearRegression
import numpy as np

# Data: House Size (sq ft)
X = [[1000], [1500], [2000], [2500], [3000]]
# Target: Price in Dollars
y = [200000, 300000, 400000, 500000, 600000]

# Initialize and Train
regressor = LinearRegression()
regressor.fit(X, y)

# Predict price for a 1800 sq ft house
prediction = regressor.predict([[1800]])
print(f"Predicted Price for 1800 sq ft: ${prediction[0]:.2f}")
```

2. Classification (Predicting a Category)

Classification is used when the output variable is a **category or label**. You are trying to put data into specific "buckets."

- **Real-World Example: Spam Detection.** An email service looks at the words in an email and classifies it as either "Spam" or "Not Spam."
- **Other Examples:** Face recognition, medical diagnosis (Sick vs. Healthy), or credit score approval (High Risk vs. Low Risk).

Python Code for Classification

We use LogisticRegression (despite the name, it is used for classification).

```
from sklearn.linear_model import LogisticRegression

# Data: [Hours Studied, Sleep Hours]
X = [[2, 5], [8, 8], [1, 4], [9, 7], [3, 6], [7, 9]]
# Target: 0 = Fail, 1 = Pass
y = [0, 1, 0, 1, 0, 1]

# Initialize and Train
classifier = LogisticRegression()
```

```
classifier.fit(X, y)

# Predict for a student who studied 5 hours and slept 7 hours
student_data = [[5, 7]]
prediction = classifier.predict(student_data)

result = "Pass" if prediction[0] == 1 else "Fail"
print(f"Prediction for the student: {result}")
```

Feature	Regression	Classification
Output Type	Continuous (Numbers)	Discrete (Categories/Labels)
Goal	To predict a specific value.	To predict a class/group.
Example	How much will it rain?	Will it rain today? (Yes/No)
Algorithms	Linear Regression, Random Forest Regressor.	Logistic Regression, SVM, Decision Trees.

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

from sklearn.linear_model import LinearRegression

# 1. Data: [Years of Experience] (X) independent data and [Salary] (y)dependent data
X = [[1], [2], [3], [4], [5]]
y = [10000, 20000, 30000, 40000, 50000]

# 2. Initialize the Model
model = LinearRegression()

# 3. Train the Model (Teaching the machine the pattern)
model.fit(X, y)

# 4. Make a Prediction
# We are asking: "If experience is 10 years, what is the salary?"
years_input = [[10]]
prediction = model.predict(years_input)
```

```
print(f"For years of experience, the estimated salary is: {prediction[0]}")
plt.scatter(X, y, color='blue', label='Actual Data') # Plotting the points
plt.plot(X, y, color='red', linewidth=2, label='Regression Line') # Plotting the line
plt.title('Experience vs Salary')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend()
plt.show()
```

Advantages of Supervised learning

Here are some advantages of supervised learning listed below:

Simplicity & clarity: Easy to understand and implement since it learns from labelled examples.

High accuracy: When sufficient labelled data is available, models achieve strong predictive performance.

Versatility: Works for both classifications like spam detection, disease prediction and regression like price forecasting.

Generalization: With enough diverse data and proper training, models can generalize well to unseen inputs.

Wide application: Used in speech recognition, medical diagnosis, sentiment analysis, fraud detection and more.

Disadvantages of Supervised learning

Requires labeled data: Large amounts of labeled datasets are expensive and time-consuming to prepare.

Bias from data: If training data is biased or unbalanced, the model may learn and amplify those biases.

Overfitting risk: Model may memorize training data instead of learning general patterns, especially with small datasets.

Limited adaptability: Performance drops significantly when applied to data distributions very different from training data.

Not scalable for some problems: In tasks with millions of possible labels like natural language, supervised labeling becomes impractical.

3.1.2 Difference between Classification and Regression

In data mining, there are two major predication problems, namely, **classification** and **regression**. The most basic difference between classification and regression is that classification algorithms are used to analyze discrete values, whereas regression algorithms analyze continuous real values.
The output variable must be either continuous nature or real value. The output variable in classification has to be a discrete value. In contrast, the output variable in regression must be either continuous in nature or real values.
In this article, we will discuss all the important differences between classification and regression. Let's start with some basics of Classification and Regression so that it becomes easier to understand how they are different from each other.

What is Classification?

Classification is the process of finding a model that represents and differentiate data classes or concepts, for the objective of being able to use the model to predict the class of objects whose class label is anonymous. The derived model is based on the analysis of a set of training records, i.e., data objects whose class label is familiar.
Classification is one of the most important concepts in data mining because it defines a process of assigning predefined class labels to instances depending on their attributes. Classification is a method that predetermined to make the analysis of high datasets effective.

What is Regression?

Regression is a type of supervised machine learning approach which can be used to forecast any continuous?valued attribute. Regression gives some business organization to explore the target variable and predictor variable associations. Thus, regression is one of the essential tools to explore the data that can be used for monetary forecasting and time series modeling.
We can use regression to perform classification. For this, it uses two methods, namely, **division** and **prediction**. In the case of division, the data is divided into regions situated on class, whereas in prediction, some formulae are used to predict the output value of the class.
Regression can predict some dependent datasets. Regression also supports methods to predict variables, but there are certain restrictions and assumptions like independence of variables, inherent normal distributions of the variables, etc.

Classification	Regression
Classification gives out discrete values.	Regression gives continuous values.
Given a group of data, this method helps group the data into different groups.	It uses the mapping function to map values to continuous output.
In classification, the nature of the predicted data is unordered.	Regression has ordered predicted data.
The mapping function is used to map values to pre?defined classes.	It attempts to find a best fit line. It tries to extrapolate the graph to find/predict the values.
Example include Decision tree, logistic regression.	Examples include Regression tree (Random forest), Linear regression
Classification is done by measuring the accuracy.	Regression is done using the root mean square error method.

Example

- **Classification:** Email → Spam or Not Spam
- **Regression:** Predict salary based on experience

Below is a **clear, detailed, and BCA-level explanation** of **Basic Terminologies (Unit 3.2)** with **proper syntax, simple language, tables, and practical examples** suitable for **exam writing**.

3.2 Basic Terminologies (Supervised Learning)

In supervised learning, understanding basic terminologies is essential because they form the foundation of how machine learning models are trained and evaluated.

3.2.1 Dataset, Features, Labels

Dataset: A dataset is a structured collection of data used to train, test, and evaluate a machine learning model.

It is usually organized in **rows and columns**, similar to a table or spreadsheet.

- **Rows** → Individual records or observations
- **Columns** → Attributes or variables

Types of Dataset

1. Training Dataset
2. Validation Dataset
3. Testing Dataset

Example Dataset (Student Performance)

Study Hours	Attendance (%)	Exam Marks
2	60	40
4	75	60
6	85	78
8	95	90

This dataset is used to predict **Exam Marks** based on **Study Hours** and **Attendance**.

Features

Definition

Features are the **input variables** or **independent variables** used by the model to make predictions.

- Represent the characteristics of the data
- Also called **predictor variables**
- Stored in columns except the target column

Notation

Features are commonly represented as **X**

[
X = (x_1, x_2, x_3, ..., x_n)
]

Example

From the above dataset:

- **Features:** Study Hours, Attendance (%)
- **Feature Vector:** (6, 85)

Labels

Definition

Labels are the **output variables** or **target values** that the model learns to predict.

- Also called **dependent variables**
- Known values during training
- Represent the correct answer

Notation

Labels are represented as **Y**

[
Y = (y_1, y_2, y_3, ..., y_n)
]

Example

- **Label:** Exam Marks
- For input (Study Hours = 6, Attendance = 85) → **Label = 78**

Relationship Between Dataset, Features, and Labels

Term	Meaning
Dataset	Complete data collection
Features (X)	Inputs to the model
Labels (Y)	Outputs to be predicted

3.2.2 Training Data, Test Data, Validation Data

To build a reliable machine learning model, the dataset is divided into three parts.

Training Data

Definition

Training data is the portion of the dataset used to **teach the model** how to make predictions.

- Model learns patterns from this data
- Weights and parameters are adjusted here
- Largest portion of the dataset

Typical Size

- **60% – 70%** of total data

Example

If total dataset has 1000 records:

- Training Data = 700 records

Validation Data

Definition

Validation data is used to **tune model parameters** and improve performance during training.

- Helps prevent overfitting
- Used for model selection
- Not used for final evaluation

Typical Size

- **10% – 20%**

Example

Used to decide:

- Best learning rate
- Best number of epochs
- Best model version

Test Data

Definition

Test data is used to **evaluate the final performance** of the trained model.

- Never seen by the model during training
- Provides unbiased performance measurement
- Used only once

Typical Size

- **10% – 20%**

Example of Data Splitting

Suppose a dataset has **100 student records**:

Data Type	Percentage	Records
Training Data	70%	70
Validation Data	15%	15
Test Data	15%	15

Simple Real-Life Example

House Price Prediction

Term	Example
Dataset	House records
Features	Size, Location, Bedrooms
Label	House Price
Training Data	Houses used to train model
Validation Data	Used to tune model
Test Data	New houses to test accuracy

Difference Between Training, Validation, and Test Data

Aspect	Training	Validation	Test
Used for Learning	Yes	No	No
Used for Tuning	No	Yes	No
Used for Evaluation	No	No	Yes
Seen by Model	Yes	Yes	No

3.3.3 Overfitting and Underfitting

In supervised machine learning, the goal of a model is to **learn the underlying pattern** in the data and **generalize well** to new, unseen data.
Two common problems that affect model performance are **Overfitting** and **Underfitting**.

Overfitting

Overfitting occurs when a machine learning model learns the **training data too perfectly**, including noise and irrelevant patterns, and performs poorly on new (test) data.

In this case, the model becomes **too complex**.

Key Characteristics of Overfitting

- Very **high training accuracy**
- **Low test accuracy**
- Model memorizes data instead of learning
- Poor generalization

Example (Student Marks Prediction)

Suppose a model is trained to predict exam marks based on:

- Study hours
- Attendance
- Sleep hours

If the model learns exact student patterns such as:

"If a student studied 6.3 hours and slept 7.2 hours, score is exactly 78"

This model may fail for new students → **Overfitting**.

Real-Life Analogy

A student memorizes answers for an exam instead of understanding concepts.
If questions change slightly, the student fails.

Causes of Overfitting

- Too many features
- Small dataset
- Very complex model (deep tree, high-degree polynomial)
- Training for too many epochs

How to Reduce Overfitting

- Use more training data
- Feature selection
- Regularization (L1, L2)

- Cross-validation
- Early stopping

Underfitting

Definition

Underfitting occurs when a model is **too simple** to learn the underlying pattern in the data.

The model fails to perform well on both **training and test data**.

Key Characteristics of Underfitting

- Low training accuracy
- Low test accuracy
- Model ignores important patterns
- Poor learning

Example (House Price Prediction)

Predicting house price using **only one feature** (number of rooms) and ignoring:

- Location
- Size
- Facilities

This simplistic model cannot capture the real pattern → **Underfitting**.

Real-Life Analogy

A student studies only headings of a subject without understanding details.

Causes of Underfitting

- Too simple model
- Insufficient features
- High bias
- Inadequate training time

How to Reduce Underfitting

- Increase model complexity
- Add more features
- Train longer
- Reduce regularization

Comparison: Overfitting vs Underfitting

Aspect	Overfitting	Underfitting
Model Complexity	Too complex	Too simple
Training Accuracy	Very high	Low
Test Accuracy	Low	Low
Bias	Low	High
Variance	High	Low
Generalization	Poor	Poor

Bias–Variance Tradeoff (Basic Concept)

- **High Bias** → Underfitting
- **High Variance** → Overfitting
- Best model balances **bias and variance**

Graphical Understanding (Conceptual)

- **Underfitting:** Straight line does not fit data
- **Good Fit:** Curve follows data trend

- **Overfitting:** Curve passes through every point

Simple Mathematical Example

Training Error vs Test Error

Model Type	Training Error	Test Error
Underfitting	High	High
Good Fit	Low	Low
Overfitting	Very Low	High

3.3.1 Loss Function – Mean Squared Error (MSE)

Loss Function

Definition

A **loss function** is a mathematical function that measures the **difference between the actual output and the predicted output** of a machine learning model.

- It tells **how wrong the model is**
- Used to **train and optimize** the model
- Smaller loss means better model performance

Mean Squared Error (MSE)

Definition

Mean Squared Error (MSE) is a commonly used loss function in **regression problems**.

It calculates the **average of the squared differences** between actual and predicted values.

Mathematical Formula

MSE=1n∑i=1n(yi−y^i)2\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2MSE=n1i=1∑n(yi−y^i)2

Where:

- nnn = Number of data points
- yiy_iyi = Actual value
- y^i\hat{y}_iy^i = Predicted value

Step-by-Step Numerical Example

Example: Predicting Student Marks

Actual Marks (y)	Predicted Marks (ŷ)
50	48
60	62
70	68

Step 1: Calculate Errors

(50−48)=2,(60−62)=−2,(70−68)=2(50 - 48) = 2,\quad (60 - 62) = -2,\quad (70 - 68) = 2(50−48)=2,(60−62)=−2,(70−68)=2

Step 2: Square the Errors

22=4,(−2)2=4,22=42^2 = 4,\quad (-2)^2 = 4,\quad 2^2 = 422=4,(−2)2=4,22=4

Step 3: Find the Mean

MSE=4+4+43=4\text{MSE} = \frac{4 + 4 + 4}{3} = 4MSE=34+4+4=4

Interpretation of MSE

- MSE = 0 → **Perfect prediction**
- Smaller MSE → **Better model**
- Larger MSE → **Poor model**

Why Squaring the Error?

1. Removes negative values
2. Penalizes large errors more
3. Makes the function differentiable for optimization

Properties of MSE

Property	Description
Non-negative	MSE ≥ 0
Differentiable	Useful for gradient descent
Sensitive to outliers	Large errors dominate
Unit	Square of original unit

Advantages of MSE

- Easy to understand and compute
- Strongly penalizes large errors
- Widely used in regression models
- Works well with gradient-based optimization

Limitations of MSE

- Highly sensitive to outliers
- Squared units are harder to interpret
- Not suitable for classification problems

Comparison with Other Loss Functions

Loss Function	Best Use
MSE	Regression
MAE	Regression (less sensitive to outliers)
Cross-Entropy	Classification

Real-Life Example

House Price Prediction

- Actual price = ₹50 lakh
- Predicted price = ₹60 lakh
- Error = ₹10 lakh
- Squared Error = 100

3.3.2 Definition of Mean Squared Error (MSE)

Definition

Mean Squared Error (MSE) is a loss function used in **regression problems** to measure the **average of the squared differences** between the actual values and the predicted values produced by a machine learning model.

It indicates **how much the predicted values deviate from the actual values**.

Mathematical Formula

MSE=1n∑i=1n(yi−y^i)2\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2MSE=n1i=1∑n(yi−y^i)2

Where:

- nnn = Total number of observations
- yiy_iyi = Actual value
- y^i\hat{y}_iy^i = Predicted value

Interpretation

- **MSE = 0** → Perfect prediction
- **Lower MSE** → Better model performance
- **Higher MSE** → Poor model performance

3.3.2.1 Computing MSE and Its Properties

Computing MSE (Step-by-Step)

Example: Exam Marks Prediction

Actual Marks (y)	Predicted Marks (ŷ)
40	42
60	58
80	75

Step 1: Calculate Errors

(40−42)=−2,(60−58)=2,(80−75)=5(40 - 42) = -2,\quad (60 - 58) = 2,\quad (80 - 75) = 5(40−42)=−2,(60−58)=2,(80−75)=5

Step 2: Square the Errors

(−2)2=4,(2)2=4,(5)2=25(-2)^2 = 4,\quad (2)^2 = 4,\quad (5)^2 = 25(−2)2=4,(2)2=4,(5)2=25

Step 3: Calculate Mean of Squared Errors

MSE=4+4+253=333=11\text{MSE} = \frac{4 + 4 + 25}{3} = \frac{33}{3} = 11MSE=34+4+25=333=11

Final MSE = 11
Properties of Mean Squared Error (MSE)

1. Non-Negative Value

- MSE is **always ≥ 0**
- Squaring removes negative errors

---

2. Penalizes Large Errors More

- Large errors have **greater impact** due to squaring
- Helps avoid large prediction mistakes

---

3. Differentiable Function

- MSE is smooth and differentiable
- Suitable for **gradient descent optimization**

---

4. Sensitive to Outliers

- Large errors dominate the loss
- Outliers can increase MSE significantly

---

5. Squared Units

- Unit of MSE is the **square of the target variable**
- Example: If output is in marks → MSE in marks²

Advantages of MSE

- Simple to calculate and understand
- Widely used in regression models
- Strong penalty for large errors
- Works well with linear regression

Limitations of MSE

- Highly sensitive to outliers
- Squared unit makes interpretation difficult
- Not suitable for classification problems

Real-Life Example

House Price Prediction

- Actual price = ₹40 lakh
- Predicted price = ₹50 lakh
- Error = ₹10 lakh
- Squared Error = 100

3.3.3 Mean Absolute Error (MAE)

3.3.3.1 Definition of MAE

Definition

Mean Absolute Error (MAE) is a loss function used in **regression problems** to measure the **average of the absolute differences** between the actual values and the predicted values.

MAE shows how far the predictions are from the actual values **without considering the direction of the error**.

Mathematical Formula

MAE=1n∑i=1n|yi−y^i|\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|MAE=n1i=1∑n|yi−y^i|

Where:

- nnn = Number of observations
- yiy_iyi = Actual value
- y^i\hat{y}_iy^i = Predicted value
- |⋅|| \cdot ||⋅| = Absolute value

Interpretation

- **MAE = 0** → Perfect prediction
- Smaller MAE → Better model performance
- MAE is in the **same unit** as the target variable

3.3.3.2 Computing MAE and Its Properties

Computing MAE (Step-by-Step Example)

Example: Predicting Student Marks

Actual Marks (y)	Predicted Marks (ŷ)
50	48
60	63
70	68

Step 1: Calculate Absolute Errors

$|50-48|=2, |60-63|=3, |70-68|=2$|50 - 48| = 2,\quad |60 - 63| = 3,\quad |70 - 68| = 2|50−48|=2,|60−63|=3,|70−68|=2

Step 2: Calculate Mean of Absolute Errors

$\text{MAE} = \frac{2 + 3 + 2}{3} = \frac{7}{3} = 2.33$MAE=2+3+23=73=2.33\text{MAE} = \frac{2 + 3 + 2}{3} = \frac{7}{3} = 2.33MAE=32+3+2=37=2.33

Final MAE = 2.33

Properties of MAE

---

1. Always Non-Negative

- $\text{MAE} \geq 0$
- Absolute value removes negative signs

---

2. Equal Weight to All Errors

- All errors contribute equally
- No extra penalty for large errors

---

3. Less Sensitive to Outliers

- Compared to MSE, MAE is more robust
- Outliers do not dominate the error

---

4. Same Unit as Target Variable

- Easy to interpret
- Example: Marks, price, temperature

---

5. Not Differentiable at Zero

- Absolute function is not smooth at zero
- Slightly difficult for optimization compared to MSE

Advantages of MAE

- Simple and easy to understand
- Interpretable in real-world units
- Less affected by outliers
- Useful when all errors are equally important

Limitations of MAE

- Does not strongly penalize large errors
- Less useful when large errors must be avoided
- Optimization is slower than MSE

3.3.3.3 Understanding Regression and R^2 (Coefficient of Determination)

Regression

Regression is a supervised learning technique used to predict **continuous numerical values**.

Examples of Regression

- House price prediction
- Salary prediction
- Sales forecasting
- Temperature prediction

Simple Linear Regression Equation

y=mx+cy = mx + cy=mx+c

Where:

- yyy = Predicted value
- xxx = Input feature
- mmm = Slope
- ccc = Intercept

R^2 Score (Coefficient of Determination)

Definition

R^2 (R-Squared) measures how well a regression model explains the **variance of the target variable**.

It indicates the **goodness of fit** of the model.

Formula

R2=1−SSresSStotR^2 = 1 - \frac{SS_{res}}{SS_{tot}}R2=1−SStotSSres

Where:

- SSresSS_{res}SSres = Sum of squared residuals
- SStotSS_{tot}SStot = Total sum of squares

Interpretation of R^2

R^2 Value	Meaning
1.0	Perfect prediction
0.9	Very good model
0.5	Average model
0.0	No predictive power
Negative	Worse than mean prediction

Example

If **$R^2 = 0.85$**, it means:

The model explains **85% of the variance** in the output.

Relationship Between MAE and R^2

Aspect	MAE	R^2
Measures	Prediction error	Model fit
Best Value	0	1
Unit	Same as target	Unitless
Used For	Error magnitude	Variance explanation

Real-Life Example (House Price)

- MAE = ₹2 lakh → Average error in prediction
- R^2 = 0.90 → Model explains 90% variation

Unit-4 : Vedic Mathematics Sutras :

*4.1 Nikhilam Navatashcaramam Dashatah *

*Meaning: *
*"All from 9 and the last from 10." *

This sutra is widely used for *fast multiplication * especially when numbers are *close to a base * like: 10, 100, 1000, 10,000…

*How it Works (Step-by-Step) *

This sutra splits the multiplication into two parts:
1. *Left Part: * Subtract each number from the nearest base (like 100)
2. *Right Part: * Multiply the differences
3. *Cross subtract to get the left side *

Python Example: nikhilam_multiplication(num1, num2)

This function multiplies two numbers (e.g., 96 and 98) using the "all from 9, last from 10" principle.

```
def nikhilam_multiplication(num1, num2): """
    Multiplies two numbers close to a base (power of 10) using the Nikhilam Navatashcaramam
    Dashatah Vedic Math sutra.
    Assumes both numbers are close to the same base (e.g., 100, 1000) and less than the base.
    """
    # Step 1: Choose the base (nearest power of 10) # This example
    assumes numbers near 100.
    base = 100

    # Step 2: Find the deviations from the base deviation1 = num1 - base # For
    96, deviation is -4 deviation2 = num2 - base # For 98, deviation is -2

    # Step 3: Calculate the Left Hand Side (LHS) of the answer
    # Cross-subtract one number's deviation from the other number # 96 + (-2) = 94 OR 98 + (-
    4) = 94
    left_hand_side = num1 + deviation2

    # Step 4: Calculate the Right Hand Side (RHS) of the answer # Multiply the deviations
    # -4 * -2 = 8
    right_hand_side = deviation1 * deviation2

    # Step 5: Combine LHS and RHS
    # The number of digits in the RHS must match the number of zeros in the base (base 100 has 2 zeros)
    # Convert RHS to a string and format it with leading zeros if necessary
```

```
    rhs_formatted = str(right_hand_side).zfill(len(str(base)) - 1)

    # Combine the parts (LHS * base + RHS is another way to think about it)
    result = int(str(left_hand_side) + rhs_formatted)

    print(f"Multiplying {num1} and {num2} using Nikhilam Sutra:") print(f"Base: {base}")
    print(f"Deviations: {deviation1}, {deviation2}") print(f"LHS (Cross-add/subtract):
    {left_hand_side}")
    print(f"RHS (Multiply deviations): {right_hand_side} (Formatted:
{rhs_formatted})") print(f"Result: {result}")

    return result

# --- Example Usage ---
nikhilam_multiplication(96,98) # Expected
Output: 9408

print("-" * 20)

# Another example using numbers close to 1000
def nikhilam_multiplication_thousands(num1,num2): base = 1000
    deviation1 = num1 - base # 998 -> -2 deviation2 = num2 -
    base # 995 -> -5 left_hand_side = num1 + deviation2 # 993
    right_hand_side = deviation1 * deviation2 # (-2) * (-5) = 10

    # Base 1000 has 3 zeros, so RHS needs 3 digits ('010') rhs_formatted =
    str(right_hand_side).zfill(len(str(base)) - 1) result = int(str(left_hand_side) + rhs_formatted)

    print(f"Multiplying {num1} and {num2} using Nikhilam Sutra (Base
{base}):")
    print(f"Result: {result}") return result

nikhilam_multiplication_thousands(998,  995)
# Expected Output: 993010
```

✅ ***Example 1:** Multiply numbers close to 100 *

*Example: *
*97 × 94 *

Here the base = 100

*Step 1: Write the deficits from 100 *

97 is *3 less * → -3
94 is *6 less * → -6

```
```

97 -3
× 94 -6
```

*Step 2: Multiply the deficits (right side) *

(-3) × (-6) = *18 *

*Step 3: Cross subtract (left side) *

97 - 6 = *91 *
(or 94 - 3 = also 91)

Final Answer:

*97 × 94 = 9118 *

✅ ***Example 2:** Multiply numbers close to 10 *

*Example: *

*8 × 7 *

Base = 10
Differences:
8 → -2
7 → -3

Right side:
(-2)(-3) = *6 *

Left side (cross subtract):
8 − 3 = *5 *

Final Answer:
*8 × 7 = 56 *

***Example 3:** Multiply above the base *

*Example: *

*103 × 105 *
Base = 100
Differences:

103 → +3
105 → +5
```

Right side:

3 × 5 = *15 *

Left side:

103 + 5 = *108 *

Final Answer:

*103 × 105 = 10815 *

* When to Use Nikhilam Sutra?
✓When numbers are close to powers of 10
✓When multiplying 2-digit or 3-digit numbers quickly
✓Very useful in competitive exams

```
def nikhilam_complement(number):
    """
    Calculates the Nikhilam complement of a number.
    This function assumes the number is an integer and finds its complement
    relative to the next higher power of 10.
    """
    if not isinstance(number, int) or number < 0:
        raise ValueError("Input must be a non-negative integer.")

    if number == 0:
        return 0

    # Determine the appropriate power of 10 (the base)
    power_of_10 = 10
    while power_of_10 <= number:
        power_of_10 *= 10

    # Convert the number to a string to access digits
    num_str = str(number)
    complement_digits = []

    # Apply "All from 9" to all digits except the last
    for digit in num_str[:-1]:
        complement_digits.append(str(9 - int(digit)))

    # Apply "the last from 10" to the last digit
    complement_digits.append(str(10 - int(num_str[-1])))

    # Join the digits and convert back to an integer
```

```
    return int("".join(complement_digits))

# Example usage:
print(f"Nikhilam complement of 8675: {nikhilam_complement(8675)}")
print(f"Nikhilam complement of 79: {nikhilam_complement(79)}")
print(f"Nikhilam complement of 123: {nikhilam_complement(123)}")
```

4.2 Ekadhikena Purvena : "By one more than the previous one."

*Ekadhikena Purvena *

*Meaning: *
*“By one more than the previous one.” *

This sutra is mainly used for *dividing numbers * and *finding decimal expansions *, especially for numbers of the form:
*1 ÷ (number ending with 9) *
Like 1/19, 1/29, 1/39, 1/49 …

It is also used in *squaring numbers ending in 5 *.
PART–1: Squaring Numbers Ending in 5
This is the most common real-life use.

*Formula: *
If a number ends in *5 *,
Example: *35 *

Then:
1. Take the previous digit: *3 *
2. Multiply it by *one more than itself * → *3 × 4 = 12 *
3. Add *25 * at the end → *1225 *

*Final Answer: $35^2 = 1225$ *
*REAL LIFE EXAMPLE (Shopping / Discount Calculation) *

Imagine you need to square numbers like:
* 25
* 35
* 45
* 75
* 95

Maybe for calculating area, volume, or cost.

Example:
A square park has a *side length of 45 meters *.
To find its area:

*45^2 = ? *

Using Ekadhikena Purvena:
Previous digit = 4
One more = 5
Multiply → *4 × 5 = 20 *
Add *25 * → *2025 *

✓Area of park = *2025 m^2 *

This helps in quick mental calculations in daily life.

PART–2: Divisions — Finding Decimals of 1/19, 1/29, 1/39 etc.

Example:
*Find the recurring decimal of 1/19 *

Here, "19" → previous digit is *1 *
Add one more → *2 *

This number (2) helps build the repeating decimal 0.052631578…

While the full method is longer, the sutra helps generate repeating sequences easily.

✓Squaring numbers ending in *5 *
✓Useful in real-life — area, cost calculation
✓Helps in dividing numbers like 1/19, 1/29
✓Very fast mental math technique

Python example demonstrating this technique to calculate 25^2

```
def square_using_ekadhikena(n): """
    Calculates the square of a number ending in 5 using the Ekadhikena Purvena
    method.
    """
    if n % 10 != 5:
        return "This method is best suited for numbers ending in 5."

    # Split the number: last_digit = 5, previous_part = 2 (for 25)
    previous_part = n // 10

    # "Ekadhikena": One more than the previous part
    next_part = previous_part + 1

    # Multiply the previous part by the next part (2 * 3 = 6)
    left_side = previous_part * next_part

    # The right side is always 25 (5 * 5)
    right_side = 25
```

```
    # Combine the results. The left side is in the hundreds place.
    result = (left_side * 100) + right_side return result

# Example: Calculate 25 * 25
number = 25
result = square_using_ekadhikena(number)
print(f"The square of {number} using Ekadhikena Purvena is: {result}")

# Example: Calculate 65 * 65
number_two = 65
result_two = square_using_ekadhikena(number_two)
print(f"The square of {number_two} using Ekadhikena Purvena is:
{result_two}")
------------------------

def square_ending_in_5(number):
  """
  Calculates the square of a number ending in 5 using the Ekadhikena Purvena sutra.
  Assumes the input 'number' is an integer ending in 5.
  """
  if number % 10 != 5:
    raise ValueError("Input number must end in 5 for this method.")

  # Extract the 'previous' part of the number (the digits before the 5)
  previous_part = number // 10

  # Calculate "one more than the previous one"
  one_more_than_previous = previous_part + 1
  # Multiply the previous part by "one more than the previous one"
  left_side = previous_part * one_more_than_previous

  # The right side of the result is always 25 for numbers ending in 5
  right_side = 25

  # Combine the parts to form the final square
  # This involves concatenating the string representations and converting back to int
  result_str = str(left_side) + str(right_side)
  return int(result_str)

# Example usage:
num1 = 25
print(f"The square of {num1} is: {square_ending_in_5(num1)}") # Output: 625

num2 = 35
print(f"The square of {num2} is: {square_ending_in_5(num2)}") # Output: 1225

num3 = 105
print(f"The square of {num3} is: {square_ending_in_5(num3)}") # Output: 11025
```

4.3 Udharan : "The extraction."

*Udharan *
*Meaning: *
*“The Extraction.” *
In Vedic Mathematics, *Udharan * refers to the method of *extracting values *, especially used in:
✓Square root extraction
✓Cube root extraction
✓Quick simplification
✓Mental math shortcuts

It means *taking out * useful numbers from a larger number to perform calculations easily.

*Real-Life Meaning *
In daily life, "extraction" means pulling out something important from a bigger group.
In mathematics, it means *extracting (or pulling out) * roots or values *without long division *.

*REAL LIFE EXAMPLE – Extracting Square Root Quickly *

*Example: Finding √2500 *

Normally you would do long division, but using extraction:
Step 1: Break 2500 into two parts
25 | 00

Step 2: Extract the square root of the first part

√25 = *5 *

Step 3: Add two zeros → answer = *50 *

✓Final Answer:

*√2500 = 50 *

This is extraction — we extracted the root of 25 to get the answer.

*REAL LIFE EXAMPLE – Finding the side of a square
land * A farmer has a *square field of area 3600 m² *.
To find the *side length *, he needs *√3600 *.

Using extraction:

Break 3600 → 36 | 00
√36 = *6 * → answer = *60 *

✓Side of the field = *60 meters *

This helps avoid long calculations.

*Example 3 – Extracting cube root mentally *

Find *³√13824 *
Split digits into groups of 3 from right:
13 | 824

Cube roots you know:
$2^3 = 8$
$3^3 = 27 \rightarrow$ too big

So first digit = *2 *
Last digit:
Look at last digit of 824 → 4
Cube ending with 4 → $4^3 = 64$ → ends in *4 *

✓Final answer = *24 *

This is how Udharan is used for rapid cube-root extraction.

- **Extracting Items from a List**

Similar to strings, you can use slicing for lists, or iterate through them to extract specific items based on a condition.

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8]

# Extract a slice of the list (e.g., elements from index 2 to 5)
extracted_slice = numbers[2:6] print(f"Extracted slice:
{extracted_slice}")

# Extract only the even numbers using a loop
even_numbers = [] for num
in numbers:
        if num % 2 == 0: even_numbers.append(num)
print(f"Even numbers: {even_numbers}")
```

Extracting Values from a Dictionary

You can extract values from a dictionary using its keys.

```
data = {"name": "John Doe", "age": 30, "city": "New York"}

# Extract a specific value by its key extracted_name =
data["name"] print(f"Extracted name: {extracted_name}")

# Extract all keys or values as a list all_keys =
list(data.keys()) all_values = list(data.values())
print(f"All keys: {all_keys}")
```

```
print(f"All values: {all_values}")
```

*Udharan = Extraction *, mainly for:
* Square roots
* Cube roots
* Simple mental math
* Shortcut calculations

```
import pandas as pd
from io import StringIO

# Sample in-line CSV data for example purpose
csv_data = """Name,Age,Occupation
John,25,Engineer
Jane,30,Teacher
Bob,22,Student
Alice,35,Doctor"""

# Read the CSV data into a DataFrame
df_csv = pd.read_csv(StringIO(csv_data))

# Extract data based on the 'Occupation' column
engineers_data_csv = df_csv[df_csv['Occupation'] == 'Engineer'][['Name', 'Age']]

# Display the extracted data
print("Data from CSV:")
print(engineers_data_csv)
```

4.4 Paraavartya : "Transposition and cancellation.":

*Paraavartya *

*Meaning: *
*“Transposition and cancellation.” *
This sutra is used to *solve equations quickly *, especially linear equations, by *moving terms from one side to another and simplifying *.

In modern math, Paraavartya =
👉 Move a term to the other side (*transpose *)
👉 Cancel or simplify common values (*cancel *)

*Where is Paraavartya used? *
✓Solving linear equations
✓Simplifying ratios
✓Checking proportions
✓Removing unwanted terms
✓Quick mental math

*REAL LIFE EXAMPLE 1 — Solving a Shopping Discount

Problem * A shopkeeper says:

> "After adding ₹50 delivery charge, your bill is ₹350."
You want to find the *actual product price *.

Equation:
x + 50 = 350

Using Paraavartya (Transpose 50 to other side):
x = 350 – 50
x = *300 *

✓ *Real product cost = ₹300 *

This is transposition: moving +50 to the other side as –50.

*REAL LIFE EXAMPLE 2 — Distance–Speed Problem *

A car travels *120 km * at *x km/h speed * in *2 hours *:

Equation:
distance = speed × time
120 = x × 2

Using Paraavartya:
x = 120 ÷ 2
x = *60 km/h *
This used *cancellation * (divide both sides by 2).

*REAL LIFE EXAMPLE 3 — Splitting a Bill (Quick Ratio

Simplification) * Three friends share a meal in the ratio:

4 : 6 : 8

Total bill = ₹900

Step 1: Add ratio:
4 + 6 + 8 = 18

Step 2: Each part = 900 ÷ 18 = 50

Step 3: Use cancellation:

* Person A: 4 × 50 = 200
* Person B: 6 × 50 = 300
* Person C: 8 × 50 = 400

The division became fast because of *transposition + cancellation *.

*REAL LIFE EXAMPLE 4 — Solving a Mobile Data Plan
Equation * Your data plan says:
"Total data = daily limit × number of days"

Total data = 150 GB
Days = 30
150 = 30 × x

Using Paraavartya:
x = 150 ÷ 30
x = *5 GB per day *
✓Summary of Paraavartya

*Paraavartya = Move (transpose) + Remove (cancel) *
Used for:
* Solving x + a = b
* Solving ax = b
* Ratio simplification
* Real-life calculations (money, speed, time, data, etc.)

- **Using zip() (pure Python):** The built-in zip() function can effectively transpose an iterable of iterables. This is a concise and efficient built-in method.

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
transposed_matrix = [list(row) for row in zip(*matrix)]
# Result: [[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

- **Using NumPy (for numerical computing):** The numpy library provides powerful tools for matrix operations, including a simple .T attribute for transposition. You can find installation instructions and documentation on the official NumPy website.

```
import numpy as np

matrix_np = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
transposed_matrix_np = matrix_np.T
# Result:
# [[1 4 7]
#  [2 5 8]
#  [3 6 9]]
```

Example: Solving a linear equation like 5x + 3 = 23

This is solved in a program by *transposing* values (moving +3to the other side as -3) and then *adjusting* (performing the subtraction and division).

```
def solve_linear_equation(a, b, c):
  # Equation form: ax + b = c
  # Transpose 'b' to the other side (c - b)
  adjusted_c = c - b
  # Solve for x (c - b) / a
  if a == 0:
    return "Cannot solve: 'a' cannot be zero"
  x = adjusted_c / a
  return x

# Example usage:
# 5x + 3 = 23
result = solve_linear_equation(a=5, b=3, c=23)
# Result: 4.0 (Check: 5*4 + 3 = 23)
```

4.5 Shunyam Saamyasamuccaye : "When the sum is the same that sum is zero."

Here is a simple and clear explanation of the Vedic Mathematics Sutra:

Shunyam Saamyasamuccaye

Meaning:
“When the sum is the same, that sum becomes zero.”

This sutra is used to *solve certain equations instantly* when *two expressions have equal sums*. In

simple words:
👉 If *left-side sum = right-side sum*,
👉 Then that part becomes *zero*,
👉 And the equation becomes *very easy* to solve.

Where is it used?

✓Equations where two terms add up to the same value
✓Quick mental solving of algebraic expressions
✓Simplifying fractions
✓Shortcut method in competitive exams

REAL LIFE EXAMPLE 1 — Sharing Money Between Two People

A father gives money to his two sons:
Son A receives:
(Amount + 20)
Son B receives:
(Amount – 20)
Total = ₹500

Equation:
(x + 20) + (x − 20) = 500
Here, *+20 and –20 cancel each other* because the *sum is the same*.

So:
x + x = 500
2x = 500
x = *250*

✓Father gave ₹250 to each (then one gets +20, one gets −20). Use

of Shunyam:
(+20) + (−20) = 0

*REAL LIFE EXAMPLE 2 — Exam Marks
Adjustment* A teacher adjusts marks as follows:
Actual marks: (x + 5)
Penalty: (x − 5)

Final total marks of two students = 110

Equation:
(x + 5) + (x − 5) = 110

+5 and –5 cancel → sum becomes zero

So:
2x = 110
x = *55*

✓Each student originally scored 55 marks.

*REAL LIFE EXAMPLE 3 — Balancing Weight on a

Scale* A weighing scale has:

Left: (Weight + 3 kg)
Right: (Weight – 3 kg)

If the scale is balanced:
(x + 3) = (x − 3)
Here the *+3 and −3* pull the same but opposite way → cancel.
Shunyam says the sum is same → sum becomes zero.
Thus difference is zero → Balanced.
This is why a 3 kg shift cancels out the opposite 3 kg shift.

```
# The equation to conceptually solve is 5*(x+1) = 3*(x+1)

def check_common_factor(term_left, term_right, common_factor_val):
    """
    Checks if a common factor exists and demonstrates that if the expressions
    are equal, the common factor must be zero (unless the equation is trivial).
    """
    # Simulate the left and right sides of the equation
    left_side = 5 * common_factor_val
    right_side = 3 * common_factor_val

    print(f"Left side: {left_side}, Right side: {right_side}")

    # The sutra suggests we can set the common factor to zero.
    # Let's test what happens when we do.
    if left_side == right_side:
        print(f"The 'sum' (expressions) are the same, so the common factor (x+1) must be zero.")
        # We deduce the solution for x
        x_solution = -1
        return x_solution
    else:
        return "Not applicable by simple observation."

# The 'samuccaya' or common term is (x+1). We test a value that makes it zero.
# common_factor_val is the value of (x+1), so if we want x=-1, this value is 0.
result = check_common_factor(5, 3, 0)
print(f"The value of x that makes the common factor zero is: {result}")
```

REAL LIFE EXAMPLE 4 — Temperature Adjustment Morning temperature: (T + 2°C)
Evening temperature: (T − 2°C)
Average temperature of the day stays the same because:
(+2) + (−2) = 0

Shunyam Saamyasamuccaye explains this cancellation principle.

✓Summary
Shunyam Saamyasamuccaye = When two expressions add to the same value → treat that part as zero.

Useful for quick solving:

* (x + a) + (x − a)
* (a + b) = (a + b)
* Balanced expressions
* Real-life adjustments (money, marks, temperature)

4.6 Anurupyena : "Proportionately."

This sutra helps in solving problems *by maintaining proportion*, especially when:

* Numbers are large but can be simplified proportionately
* Multiplication becomes easier by using ratios
* Dividing or scaling values
* Solving equations that involve proportional changes

In simple words:
If two numbers maintain the *same ratio*, you can simplify them proportionately to make calculations easy.

Where is it used?
✓Multiplication using proportional reduction
✓Ratio and proportion problems
✓Solving equations with proportionality
✓Simplifying big numbers mentally
✓Percentage, increase/decrease
✓Applications in real life: finance, shopping, cooking, construction

*REAL LIFE EXAMPLE 1 — Cooking Recipe

Scaling* A recipe for 4 people needs:
* 2 cups of rice
* 4 cups of water

If you want to cook for *8 people*, you scale *proportionately*:
People doubled → ingredients double

Rice = 2 × 2 = *4 cups*
Water = 4 × 2 = *8 cups*

This is *Anurupyena (proportionately)*.

```
# Problem: Find the missing term (x) in the proportion 3:9 :: 4:x
# The mathematical relationship is 3/9 = 4/x

# Given values
a = 3
b = 9
c = 4
```

```
# The formula derived from the principle of proportionality is: x = (b * c) / a
# Using the Anurupyena principle to solve this in Python:
missing_term_x = (b * c) / a

# Print the result
print(f"For the proportion {a}:{b} :: {c}:{missing_term_x}, the missing term (x) is: {missing_term_x}")
```

Output:

For the proportion 3:9 :: 4:12.0, the missing term (x) is: 12.0

REAL LIFE EXAMPLE 2 — Salary Increase

A company increases salary *proportionately* by 10%.
Old salary = ₹30,000
Increase = 10% = 0.10

New salary = 30,000 + (30,000 × 0.10)
New salary = 30,000 + 3,000 = *₹33,000*

Salary is raised *proportionately* for all employees.

REAL LIFE EXAMPLE 3 — Distance & Fuel Consumption A bike uses *2 litres* of petrol for *60 km*.

How much petrol for 150 km?
Distance ratio = 150 / 60 = 2.5
Petrol = 2 × 2.5 = *5 litres*

This is proportional scaling (Anurupyena).

REAL LIFE EXAMPLE 4 — Fast Multiplication Using Proportion Multiply: *48 × 25*
Direct multiplication is longer.

Use proportion:
25 = 100 / 4
So: 48 × 25
= 48 × (100 / 4)
= (48 ÷ 4) × 100
= 12 × 100
= *1200*

Because the numbers were adjusted proportionately.

REAL LIFE EXAMPLE 5 — Construction Material Planning For 1 square meter wall:
* 50 bricks
* 1 bag of cement

For *10 square meters*:

Bricks = 50 × 10 = *500 bricks*
Cement = 1 × 10 = *10 bags*

Materials scale *proportionately* with the wall area.

✓Summary

Anurupyena = "Proportionately"
Used when:

* Scaling up or down
* Multiplying using ratios
* Reducing numbers to simpler proportions
* Real-world planning and budgeting
* Solving equations with proportional changes

4.7 Sankalana-Vyavakalanabhyam : "By addition and by subtraction."

“By addition and by subtraction.”

This sutra is one of the most useful techniques in Vedic Mathematics.
It helps in solving problems quickly using *simple addition and subtraction* instead of long calculations.

It is mainly used in:
✓Solving linear equations
✓Solving simultaneous equations
✓Quick mental calculations
✓Checking differences between numbers
✓Addition–subtraction shortcuts

1. REAL LIFE EXAMPLE — Shopping Discount Calculation
A jacket costs *₹1500*, and the shopkeeper gives a *₹200 discount*.

Using:
Sankalana = Addition
Vyavakalan = Subtraction

Price after discount:

1500 − 200 = *₹1300*

Simple subtraction helps avoid long calculation.
2. REAL LIFE EXAMPLE — Temperature Change

Morning temperature: 18°C
Evening temperature: 25°C

Increase in temperature = 25 − 18 = *7°C*

We used *Vyavakalan (Subtraction)* to find the change.

3. REAL LIFE EXAMPLE — Solving Linear Equations Example:
x + 5 = 20
To solve:
Subtract 5 from both sides:
x = 20 − 5
x = *15*

Using *addition and subtraction*, we solve instantly.

4. REAL LIFE EXAMPLE — Simultaneous Equations Solve:2x + y = 20
2x − y = 10

Add both equations (Sankalana):
(2x + y) + (2x − y) = 20 +
10 4x = 30
x = 30 ÷ 4 = *7.5*

Now subtract (Vyavakalan):
(2x + y) − (2x − y) = 20 −
10 2y = 10
y = *5*

This sutra makes solving equations much faster.

```
# Given system of linear equations:
# eq1: x + y = 20
# eq2: x - y = 10

# 1. Sankalana (Addition)
# Add (x + y) and (x - y) to find a simple equation for
x. # (x + y) + (x - y) = 20 + 10
# 2x = 30
sum_of_equations = 20 + 10
```

```
x = sum_of_equations / 2

# 2. Vyavakalanabhyam (Subtraction)
# Subtract (x - y) from (x + y) to find a simple equation for y.
# (x + y) - (x - y) = 20 - 10
# 2y = 10
difference_of_equations = 20 - 10
y = difference_of_equations / 2

print(f"The principle uses addition and subtraction to quickly find the solution.")
print(f"By addition, we find x = {x}")
print(f"By subtraction, we find y = {y}")

# Verification
print(f"\nVerification:")
print(f"x + y = {x + y} (matches 20)")
print(f"x - y = {x - y} (matches 10)")
```

*5. REAL LIFE EXAMPLE — Bank Balance
Calculation* You deposit ₹3000
You withdraw ₹800

Final balance = 3000 − 800
= *₹2200*

Daily banking uses *addition and subtraction* continuously.

- Summary
Sankalana–Vyavakalanabhyam = Solve "by addition and by subtraction."

Used for:
* Addition/subtraction shortcuts
* Solving equations quickly
* Understanding daily changes (temperature, money, distance)
* Banking, shopping, budgeting
* Eliminating variables in equations

4.8 Puranapuranabhyam : "By the completion or non-completion."

Puranapuranabhyam
Meaning:
"By completion or non-completion."
This sutra is used when a number is *near a base* (10, 100, 1000…).
You can *complete the base* or *use the part that is incomplete* to solve problems quickly.

In simple words:
If a number is *close to a round number*,
Use the difference from the base to make calculations easier.

It is very useful in:
✓Multiplication
✓Addition/Subtraction
✓Completing numbers mentally
✓Checking totals
✓Quick estimates
✓Finding missing values

The Vedic Mathematics sutra
Puranapuranabhyam, meaning "by the completion or non-completion", refers to solving problems by considering a whole entity or its complement to simplify calculations, particularly in algebra. In Python, this concept can be illustrated by using **complements** in arithmetic operations or by factoring polynomials to find their **complete** set of roots.
Example 1: Using Complements for Addition
The principle of Puranapuranabhyam can be used in mental math by converting numbers to their nearest base (e.g., a multiple of 10, 100) and using complements. For example, adding

99+499 plus 4
99+4
can be thought of as

(100−1)+4open paren 100 minus 1 close paren plus 4
(100−1)+4
, which simplifies to

100+3=103100 plus 3 equals 103
100+3=103
. The "completion" here is mentally rounding 99 to 100.

```
def add_using_completion(number, amount, base=100):
  """
  Adds two numbers using the 'completion' (complements) method,
  rounding the primary number to the nearest base for simplification.
  """
  if number + amount < base:
    return number + amount # Regular addition if below base

  # Completion logic: round 'number' to the nearest 'base'
  # and adjust 'amount' accordingly.
  deviation = base - number
  # The operation becomes: base + (amount - deviation)
  result = base + (amount - deviation)

  return result
```

```
# Example usage:
num1 = 99
num2 = 4
result = add_using_completion(num1, num2)
print(f"Adding {num1} and {num2} using completion gives: {result}")

num3 = 43
num4 = 16
# This uses the regular addition part, as it doesn't cross a significant 'base' boundary in a simple way
result_simple = add_using_completion(num3, num4, base=100)
print(f"Adding {num3} and {num4} (without 'completion' logic crossing 100) gives: {result_simple}")
```

*REAL LIFE EXAMPLE 1 — Shopping & Round-Off

Calculation* You buy items costing:
* ₹98
* ₹97
* ₹105

Instead of adding directly:
98 is *2 less* than 100
97 is *3 less* than 100
105 is *5 more* than 100

So use completion:
(100 − 2)
(100 − 3)
(100 + 5)

Add proportionately:
100 + 100 + 100 = 300
Adjust differences: −2 −3 +5 = 0

✓Total = *300*

This avoids long addition and uses *completion and non-completion*.

REAL LIFE EXAMPLE 2 — Multiplying Numbers Near a Base

Multiply 98 × 97

Base = 100
98 is *2 less*
97 is *3 less*

Use Puranapuranabhyam:

Left side:
100 − (2 + 3) = 95

Right side:
(2 × 3) = 6 → write as 06

✓Answer = *9506*

Used completion (100) and non-completion (2, 3).

*REAL LIFE EXAMPLE 3 — Filling a Water

Tank* A tank holds *100 litres*.
You already filled *93 litres*.
How much more is needed?

Use completion:
100 − 93 = *7 litres*

This is completing to the nearest base (100 L).

REAL LIFE EXAMPLE 4 — Completing Attendance in School A class has *50 students*.
Present = 47
Absent = ?

Use completion:

50 − 47 = *3 absentees*
We complete to the full capacity (50).

*REAL LIFE EXAMPLE 5 — Electricity Bill Round-

Off* Your electricity consumption is:
396 units

The bill rate changes at *400 units*.

How many units before next slab?

Complete to 400:

400 − 396 = *4 units*

This is used daily in household calculations.

```
def solve_cubic_equation_by_completion(coefficients):
# We can define a function to check if the roots work
        defcheck_roots(roots): for x in roots:
                    if x**3 - 2*x**2 - 5*x + 6 != 0: return False
                return True

        potential_roots = [1, -2, 3] # Based on the 'completion' method result

        if check_roots(potential_roots):
                return f"By 'completion' (factoring), the roots are:
{potential_roots}" else:
                return "Roots are incorrect."

# Example usage:
equation_coeffs = [1, -2, -5, 6]
result=solve_cubic_equation_by_completion(equation_coeffs) print(result)
```

Puranapuranabhyam = “Use completion (to nearest base) or non-completion (difference).”

Best for:
* Fast multiplication
* Approximations
* Completing round numbers
* Shopping & billing
* Filling/remaining calculations
* Time, money, storage estimates

4.9 Chalana-Kalana : "By motion or by applying a shift."

Here is a clear and simple explanation of the Vedic Mathematics Sutra:
***Chalana–Kalana** :
“By motion or by applying a shift.”

This sutra means:
Shift the numbers
Adjust (move) values
Simplify the calculation using a small change
Then correct (reverse shift) if necessary

It is used when:
* Numbers can be *slightly adjusted* to make calculation easier
* You can *add or subtract* a small amount to simplify
* The problem becomes easy after a *shift (motion)*
* Then you adjust back to get the correct answer

REAL LIFE EXAMPLE 1 — Adjusting Price During Shopping

Suppose the cost of an item is *₹199*.
You want to calculate the price for *5 items*.
199 × 5 is tough
But *200 × 5* is very easy.

Apply *shift* (Chalana):
Shift 199 → 200 (add 1)

Calculate:
200 × 5 = 1000

Now adjust for the shift:
You added 1 five times → 5 extra
So subtract 5:
1000 − 5 = *₹995*

✓Fast calculation using a shift.

REAL LIFE EXAMPLE 2 — Time Calculation

You travel *58 minutes* every day to college.
You want to know weekly total (7 days).
58 × 7 is hard.
Shift 58 → 60 (add 2)

Calculate:
60 × 7 = 420 minutes
Adjustment: added 2 seven times → 14 extra

420 − 14 = *406 minutes*

In Python, the concept of "by motion or by applying a shift" (**Chalana-Kalana**) can be simply demonstrated through list slicing or arithmetic operations on a variable.

Example 1: List Slicing (Applying a "Shift" or "Motion" of elements)

This example shows shifting elements within a list to the left or right, effectively changing their position (motion).

```
# Original list of elements elements = [10, 20,
30, 40, 50] print(f"Original: {elements}")

# Shifting the elements to the left by 2 positions
```

```
# The slice starts from index 2 and goes to the end, effectively moving those elements forward.
shifted_left = elements[2:]
print(f"Shifted Left (by 2): {shifted_left}")

# We can also wrap the 'shifted-out' elements around to the end # This simulates continuous
motion or a cyclical shift wrapped_shift = elements[2:] + elements[:2]
print(f"Cyclically Shifted (by 2): {wrapped_shift}")
```

Output:

Original: [10, 20, 30, 40, 50]

Shifted Left (by 2): [30, 40, 50]

Cyclically Shifted (by 2): [30, 40, 50, 10, 20]

Example 2: Arithmetic Operation (Applying a "Shift" to a value's state)

```
# Initial position
position = 5

# Applying a positive shift (motion in one direction)
shift_amount = 3 position +=
shift_amount
print(f"After positive shift: {position}")

# Applying a negative shift (motion in the opposite direction)
shift_amount_negative = -2 position +=
shift_amount_negative
print(f"After negative shift: {position}")
```

Output:

After positive shift: 8 After negative
shift: 6

✓Calculated quickly using one shift.

Summary

Chalana–Kalana = “Move, adjust, shift numbers to simplify — then correct.”

Useful for:
* Fast multiplication
* Approximations
* Shopping and billing
* Time calculations
* Travel and fuel math
* Daily mental math

4.10 Yavadunam : "Whatever is the deficiency."

This sutra is used when a number is *close to a base* such as
10, 100, 1000, 50, etc.

We use:
How much it is *less* than the base (the deficiency)
Then use that deficiency to simplify multiplication or calculation.

*REAL-LIFE EXAMPLE 1 — Supermarket Discount

Calculation* A product costs *₹98*.
The shopkeeper gives you a *₹2 discount*.

Here,
Base = 100
Price = 98
Deficiency = 100 − 98 = *2*
So the final price = 98 − 2 = *₹96*

✓We used the deficiency (2) to calculate the discount quickly.

REAL-LIFE EXAMPLE 2 — Multiplying Numbers Near 100
Example: *98 × 97*
Both are near 100
Find their deficiencies:
98 → deficiency 2
97 → deficiency 3

Step 1:
Subtract crosswise:
98 − 3 = *95*
(or 97 − 2 — same)

Step 2:
Multiply deficiencies:
2 × 3 = *6* → write as *06*

Final answer = *9506*

✓Fast multiplication using deficiency.

*REAL-LIFE EXAMPLE 3 — Estimating

Savings* You plan to save *₹950* per month.
Your target is *₹1000*.

Deficiency = 1000 − 950 = *50*

So you know:
You are short by *₹50 per month*
Over 12 months → 50 × 12 = *₹600 deficiency for the year*

✓Yavadūnam helps estimate how much you fall short from a target.

*REAL-LIFE EXAMPLE 4 — Fuel Tank

Filling* Your bike tank capacity = *15 litres*
You currently have *13.5 litres*

Deficiency = 15 − 13.5 = *1.5 litres*

✓You instantly know how much more fuel you need to fill the tank.

*REAL-LIFE EXAMPLE 5 — Marks Needed to

Pass* Pass marks = 40
Student score = 33

Deficiency = 40 − 33 = *7*

✓Student needs *7 more marks* to pass.

The Sanskrit term **"Yavadunam"** ("Whatever is the deficiency" or "Whatever the extent of its deficiency") is a principle from Vedic Mathematics used as a shortcut for calculations, specifically for **squaring numbers near a base** (a power of 10 like 10, 100, 1000).

In Python, the concept is implemented by functions that calculate this "deficiency" (difference from the base) and apply the steps of the Vedic math method.

Python Example: Squaring a Number using the "Yavadunam" Principle

This Python function implements the *Yavadunam* method for squaring numbers close to a power of 10. The result is calculated in two parts: the left side (LHS) and the right side (RHS).

```
def yavadunam_square(number, base): """
    Calculates the square of a number using the Yavadunam (deficiency) sutra.

    The method is most efficient for numbers close to a power of 10 (the base).

    Args:
```

```
        number (int): The number to be squared.
        base (int): The nearest power of 10 (e.g., 10, 100, 1000).

    Returns:
        int: The square of the number.
    """
    # Step 1: Find the deficiency (or excess)
    deficiency = number - base # This is the "Yavadunam" or "deficiency"

    # Step 2: Calculate the left-hand side (LHS) of the answer
    # "Lessen it to the same extent" (or increase if it's an excess)
    left_hand_side = number + deficiency

    # Step 3: Calculate the right-hand side (RHS) of the answer # "Set up the square of the
    deficiency"
    right_hand_side = deficiency ** 2

    # Step 4: Combine the results (handling carry-over if RHS has too many digits)
    # The number of digits in RHS must match the number of zeros in the base
    num_zeros = len(str(base)) - 1

    # Check for carry-over
    if len(str(right_hand_side)) > num_zeros:
        # Calculate the carry and the final RHS part
        carry = right_hand_side // (10**num_zeros) right_hand_side =
        right_hand_side % (10**num_zeros) left_hand_side += carry

    # Format the result with leading zeros for the RHS if needed and combine # The use of an f-string with fill
    and width specifiers helps with
formatting
    format_string = f"{{:0{num_zeros}d}}".format(right_hand_side) final_result_str = str(left_hand_side) +
    format_string

    return int(final_result_str)

# --- Simple Examples ---

# Example 1: Squaring 98 (base 100)
num1 = 98
base1 = 100
# Deficiency: 100 - 98 = 2
```

```
print(f"The square of {num1} is: {yavadunam_square(num1, base1)}") # Output: 9604

# Example 2: Squaring 104 (base 100)
num2 = 104
base2 = 100
# Excess: 104 - 100 = 4
print(f"The square of {num2} is: {yavadunam_square(num2, base2)}") # Output: 10816

# Example 3: Squaring 9 (base 10)
num3 = 9
base3 = 10
print(f"The square of {num3} is: {yavadunam_square(num3, base3)}") # Output: 81

# Example 4: Squaring 89 (base 100, demonstrates carry-over handling)
num4 = 89
base4 = 100
print(f"The square of {num4} is: {yavadunam_square(num4, base4)}") # Output: 7921
```

Used for:
* Fast multiplication
* Approximations
* Finding shortage from a target
* Shopping, savings, fuel, marks comparison
* Mental math calculations

4.11 Vyastisamanstih : "The parts and the whole."

This sutra tells us:
👉 Break a number/problem into *parts (Vyasti)*
👉 Combine them back to get the *whole (Samasti)*

Example 1: List and Aggregation

This example uses a list of individual items (the "parts") and calculates their sum (the "whole").

python

```
# The individual parts (vyasti)
individual_scores = [8, 12, 5, 10, 7]
print(f"Individual scores (Vyasti): {individual_scores}")

# The collective whole (samansti)
```

```
total_score = sum(individual_scores)

print(f"Total score of all individuals combined (Samansti): {total_score}")

# The relationship: the whole is the sum of its parts
```

Example 2: Class and Instances

This example uses object-oriented programming, where individual "Person" objects (the "parts") belong to a larger "Community" class that manages them (the "whole").

python

```
class Person:
    def __init__(self, name, wealth): self.name = name
        self.wealth = wealth

    def __repr__(self):
        return f"{self.name} (${self.wealth}k)"

class Community:
    def __init__(self, members): self.members = members

    def get_total_community_wealth(self):
        # The whole (community wealth) is derived from its parts (members'
wealth)
        total = sum(member.wealth for member in self.members) return total

# Create the individual parts (vyasti)
p1 = Person("Alice", 50) p2 =
Person("Bob", 75)
p3 = Person("Charlie", 120)

# Create the collective whole (samansti) from the parts
local_community = Community([p1, p2, p3])

print(f"Individual members (Vyasti): {local_community.members}") print(f"Total wealth of the
community (Samansti):
${local_community.get_total_community_wealth()}k")
```

It is very useful in:

✓Mental addition
✓Mental multiplication
✓Bill/price calculation
✓Time splitting
✓Real-world quick math

*REAL-LIFE EXAMPLE 1 — Grocery

Billing* You buy 3 items:
* Sugar: ₹45
* Bread: ₹30
* Milk: ₹25
Instead of adding 45 + 30 + 25 directly, break into parts:
(40 + 5) + (30) + (20 + 5)

Group simple parts:
40 + 30 + 20 = *90*
5 + 5 = *10*
Total = 90 + 10 = *₹100*

✓Used parts (split) and whole (combine).

REAL-LIFE EXAMPLE 2 — Mental Multiplication
Multiply *12 × 7*
Break 12 into parts:
12 = 10 + 2

Multiply separately:
10 × 7 = 70
2 × 7 = 14

Combine (whole):
70 + 14 = *84*

✓Problem becomes easy by breaking into parts.

*REAL-LIFE EXAMPLE 3 — Time

Management* You study:
* 25 minutes of Math
* 20 minutes of English
* 15 minutes of Science

Break into parts:
(20 + 5) + 20 + (10 + 5)

Combine:

20 + 20 + 10 = *50*
5 + 5 = *10*

Total study time = 50 + 10 = *60 minutes*

✓Parts → Whole approach.

*REAL-LIFE EXAMPLE 4 — Restaurant Bill

Splitting* Bill = ₹1,270
You and 2 friends will share.

Break 1270 → parts:
1200 + 70

Then divide:
1200 ÷ 3 = 400
70 ÷ 3 ≈ 23.33
Combine:
400 + 23.33 ≈ *₹423 each*

✓Breaking the whole into parts simplifies division.

*REAL-LIFE EXAMPLE 5 — Salary
Breakdown* Monthly salary = ₹30,000
Break into meaningful parts:

* Basic: 20,000
* Allowances: 8,000
* Incentives: 2,000

Whole = 30,000
Parts help understand money flow.

✓Vyasti → understanding
✓Samasti → complete monthly salary

*REAL-LIFE EXAMPLE 6 — Multiplying Large

Numbers* Multiply *23 × 14*
Break:
23 = 20 + 3
14 = 10 + 4

Multiply parts:
20 × 10 = 200
20 × 4 = 80

3 × 10 = 30
3 × 4 = 12

Combine all:
200 + 80 + 30 + 12 = *322*

✓Classic “parts → whole” method.

⭐Summary

Vyastisamanstih = “Break into parts → solve → combine to get whole.”

Useful for:
* Addition
* Multiplication
* Budgeting
* Time management
* Splitting bills
* Fast mental math

4.12 Sesanyan : "The remainder."

This sutra is used when:
✓Instead of calculating the full answer,
✓We only need the *remainder* after division.

In Python, the concept of "the remainder" is achieved using the **modulo operator**, represented by the percent sign (%). This operator returns the remainder after one number is divided by another.

Here is a simple example:

python

```
# The modulo operator (%) returns the remainder of a division
dividend = 10
divisor = 3
remainder = dividend % divisor

print(f"When {dividend} is divided by {divisor}, the remainder is:
{remainder}")
# Output: When 10 is divided by 3, the remainder is: 1
```

More Examples

- **To check if a number is even or odd:** If a number modulo 2 is 0, it's even; otherwise, it's odd.

  python

```
number = 4
if number % 2 == 0:
    print(f"{number} is an even number.") else:
    print(f"{number} is an odd number.")
# Output: 4 is an even number.
```

- **With floating-point numbers:** The modulo operator also works with floattypes.

  python

```
result_float = 10.5 % 3.2
print(f"The remainder of 10.5 % 3.2 is: {result_float}")
# Output: The remainder of 10.5 % 3.2 is: 0.9000000000000004
```

It helps in:
* Quick divisibility
* Checking balance left
* Modular arithmetic
* Daily-life remainder problems
* Fast mental math

REAL-LIFE EXAMPLE 1 — Sharing Chocolates

You have *29 chocolates* and want to divide them among *5 children*.

Instead of full division,

29 ÷ 5 = 5 remainder *4*

✓Each child gets 5 chocolates
✓Remainder = *4 chocolates left*

This is Sesanyan — focusing on the remainder.

*REAL-LIFE EXAMPLE 2 — Weekly

Planning* You have *50 pages* to read.
You plan to read *7 pages per day*.

50 ÷ 7 = 7 remainder *1*

This remainder tells you:

After 7 days of reading, *1 page will be left*.

✓Sesanyan helps know "what remains unfinished".

REAL-LIFE EXAMPLE 3 — Bus Seat Allocation

A bus has *47 passengers* and each row has *4 seats*.
47 ÷ 4 = 11 rows filled, remainder *3*

✓3 passengers don't fit into full rows.

This remainder helps plan extra seating.

*REAL-LIFE EXAMPLE 4 — Packing
Items* A box can hold *12 bottles*.
You have *50 bottles*.
50 ÷ 12 = 4 full boxes, remainder *2*
✓2 bottles remain unpacked.

*REAL-LIFE EXAMPLE 5 — Divisibility

Check* Is *125* divisible by *8*?
125 ÷ 8 = remainder *5*

Since remainder ≠ 0
✓Not divisible

Sesanyan helps quickly detect divisibility.

*REAL-LIFE EXAMPLE 6 — Multiplying Large Numbers (Shortcut using

remainder)* Find the *last digit* of:
27 × 19

Only the last digits matter:
7 × 9 = 63 → remainder 3 (mod 10)

So the final answer ends with *3*.

✓Sesanyan focuses on remainder after base-10 division. Summary

Śeṣānyāna = "Work with the remainder instead of full calculation."

Useful for:
* Sharing items
* Time and schedule planning
* Packing and arrangement
* Checking divisibility
* Finding last digits
* Quick mental arithmetic

4.13 Gunitasamuchyah : "The product of the sum."

This sutra states that:

👉 *If two expressions have the same sum, their products are equal*
OR
👉 *The sum of the factors can help find the product quickly*

It is commonly used to simplify multiplication.
REAL-LIFE EXAMPLE 1 — EASY MENTAL MULTIPLICATION
Multiply:
17 × 13

Notice the numbers:

17 + 13 = *30*

Now rewrite:
17 = 15 + 2
13 = 15 − 2

So, "product of the sum" formula:
$(a + b)(a - b) = a^2 - b^2$
with a = 15, b = 2

Product = $15^2 - 2^2$
= 225 − 4
= *221*
✓Fast calculation using the sum around the middle value.

REAL-LIFE EXAMPLE 2 — SHOPPING DISCOUNT TRICK

A shopkeeper says:
Buy 2 shirts priced ₹500 and ₹700, get ₹100 off total.
Sum = 500 + 700 = 1200
Product is not needed directly, but the *samuccaya* (sum) helps decide:
Total = 1200 − 100
= *₹1100*

Here, the "sum" determines the final price, not individual prices.

✓The sutra helps see the whole instead of focusing on parts.

REAL-LIFE EXAMPLE 3 — TIME & WORK

Two workers finish work in:
* Worker A: 6 hours
* Worker B: 3 hours

To find combined work rate:
Sum of times = 6 + 3 = 9
Product = 6 × 3 = 18

Combined time = Product ÷ Sum
= 18 ÷ 9 = *2 hours*

✓Work rate problems use “product of the sum” directly.

REAL-LIFE EXAMPLE 4 — ELECTRICITY CALCULATION

If two resistances (R_1) and (R_2) are in parallel:

Equivalent resistance:
(R = \frac{R_1 R_2}{R_1 +

R_2}) This formula uses:

* *Product* ($R_1 \times R_2$)
* *Sum* ($R_1 + R_2$)

Example:
$R_1 = 6\Omega$, $R_2 = 3\Omega$

R = (6 × 3) ÷ (6 + 3)
= 18 ÷ 9
= *2Ω*

✓The sutra appears naturally in physics formulas.

```
def sum_of_coefficients(expression_str):
    # This simple example treats a number as the "product" for the Vedic Math check
    # In a real Python program, you would get this number from your actual calculation.
    return sum(int(digit) for digit in str(expression_str) if digit.isdigit())

# Example using the algebraic expression (x + 7)(x + 9) resulting in 80 (sum of coeffs)
factor1_sum = (1 + 7) # Sum of coefficients in (x+7)
```

```
factor2_sum = (1 + 9) # Sum of coefficients in (x+9)

# Calculate the "product of the sum"
product_of_sum = factor1_sum * factor2_sum

# Calculate the "sum of the product" (sum of coefficients of the resulting polynomial)
# The resulting polynomial's coefficients are 1, 16, 63
sum_of_product_coeffs = 1 + 16 + 63

print(f"Factor 1 Sum: {factor1_sum}")
print(f"Factor 2 Sum: {factor2_sum}")
print(f"Product of the Sum (LHS): {product_of_sum}")
print(f"Sum of Product Coefficients (RHS): {sum_of_product_coeffs}")

if product_of_sum == sum_of_product_coeffs:
   print("Gunitasamuchyah principle holds: Calculation verified!")
else:
   print("Gunitasamuchyah principle fails: Calculation might be incorrect.")
```

REAL-LIFE EXAMPLE 5 — SPEED OF VEHICLES
A car travels different distances at two speeds.

Formula for average speed uses:

Product of the speeds ÷ Sum of the speeds

Example:
Speed going: 40 km/hr
Speed returning: 60 km/hr

Average speed = (40 × 60) ÷ (40 + 60)
= 2400 ÷ 100
= *48 km/hr*

✓Again “product of the sum” is central.

Summary

Gunitasamuccayah = “Use the relationship between sum and product to simplify calculations.”

Useful for:
* Quick multiplication
* Time–work problems
* Physics formulas (resistance, average speed)
* Mental math involving symmetry

4.14 Vistaran : “Expansion."

Here is a clear and simple explanation of the Vedic Mathematics Sutra:

Python Example: Simulating "Expansion"

This example uses the Python extend()method to illustrate how one list "expands" to include elements from another list, conceptually similar to the meaning of *Vistaran* (expansion).

```
vistaran_base = ["idea_A", "idea_B"]

# Another list with new elements to be added (the "expansion" content).
new_content = ["idea_C", "idea_D", "idea_E"]

print(f"Original base list: {vistaran_base}") print(f"Content to add:
{new_content}")

# Use the extend() method to expand the base list with elements from new_content.
vistaran_base.extend(new_content)  #

print(f"Expanded list (after Vistaran): {vistaran_base}")
```

Alternative Example: Using the +Operator

The addition operator (+) can also be used to combine two lists, creating a new, expanded list.

```
# Using the + operator for expansion (concatenation)

list_one = [10, 20]
list_two = [30, 40, 50]

# A new, expanded list is created by adding the two lists together.
expanded_list = list_one + list_two

print(f"List one: {list_one}") print(f"List two:
{list_two}")
print(f"New expanded list: {expanded_list}")
```

This sutra is used when we want to *expand*:

* Numbers
* Expressions
* Multiplication
* Algebraic

forms It means:

👉 Break a big expression into smaller expanded parts
👉 Work on each part
👉 Combine the results

This is like “FOIL expansion” in algebra, or splitting numbers for easy multiplication.

*REAL-LIFE EXAMPLE 1 — Shopping Bill
Expansion* A shopkeeper calculates:
₹23 × 14

Instead of direct multiplication, expand:
23 = 20 + 3
14 = 10 + 4

Now expand (Vistāraṇa):
(20 × 10) + (20 × 4) + (3 × 10) + (3 × 4)
= 200 + 80 + 30 + 12
= *322*

✓Large bill calculation becomes easy by expanding.
REAL-LIFE EXAMPLE 2 — Monthly Electricity Charge

Electricity bill formula:
Bill = Units × Rate
If you used *147 units* at ₹*6 per unit*, expand:

147 = 100 + 40 + 7

Now calculate:
100 × 6 = 600
40 × 6 = 240
7 × 6 = 42

Total = 600 + 240 + 42 = *₹882*

✓Expansion simplifies calculations without a calculator.

*REAL-LIFE EXAMPLE 3 — Area Calculation (Algebra
Expansion)* You want to calculate the area of a rectangle:

Length = (10 + 2)
Breadth = (8 + 1)

Area = (10 + 2)(8 + 1)

Expand:
= 10×8 + 10×1 + 2×8 + 2×1

= 80 + 10 + 16 + 2
= *108 square units*

✓Expansion makes the area easier to compute.

*REAL-LIFE EXAMPLE 4 — Salary

Breakdown* Your monthly income:

₹28,500

Instead of working with a big number, expand:

28,500 = 20,000 + 8,000 + 500

This helps in:

* Budget planning
* EMI calculation
* Savings distribution

Example:

Saving 10% of each part:

2000 + 800 + 50 = *₹2,850 saved*
✓Expansion helps simplify financial planning.

REAL-LIFE EXAMPLE 5 — Mental Math for Multiplying Multiply:
46 × 9

Expand 9 as (10 − 1):
46 × (10 − 1)
= (46 × 10) − (46 × 1)
= 460 − 46
= *414*

✓Expansion prevents large multiplication mistakes.

REAL-LIFE EXAMPLE 6 — Construction Material Estimate A mason needs to calculate total bricks:
(20 + 5) rows
Each row has (100 + 20) bricks

Expand:
(25)(120)

= 20×120 + 5×120
= 2400 + 600
= *3000 bricks*

✓A simple expansion makes large calculations easy.
⭐ Summary

Vistāraṇa = “Expand the number or expression into parts → solve → combine.”

Useful for:

* Shopping and billing
* Area and measurement
* Mental multiplication
* Electricity/water bills
* Finance and salary planning
* Algebraic expansion

4.15 Rupan : "Form."

“Form” or “Standard Form.”

This sutra is used to *bring numbers into a simpler or standard form* to make calculations easier. It

means:
👉 Convert a number into a more convenient form
👉 Apply the calculation
👉 Convert back if needed

Rūpam helps especially in:
✓Multiplication
✓Division
✓Fractions
✓Approximations
✓Mental math

In Python, the term "form" typically refers to two main concepts: data structures and user interfaces. A simple example using a **class** to structure data is provided below, while a Graphical User Interface (GUI) form requires libraries like Tkinter.

Simple Python Data Form using a Class

A "form" can be conceptualized as a way to structure data logically, much like a form you fill out on a website. In simple Python code, this can be achieved using a **class** to define the fields and methods for handling the data.

```
class UserForm:
        """Represents a simple user information form."""  def _init_(self, name,
        email):
                self.name = name self.email =
                email

        def display_info(self):
                """Prints the information collected in the form."""  print(f"User Name: {self.name}")
                print(f"User Email: {self.email}")

# --- Example Usage ---

# 1. Create an instance of the form with specific data
user_data = UserForm("Rupan", "rupan.example@email.com")

# 2. Access the data and display it
user_data.display_info()
```

* ⭐ Example 1 — Money Calculation (Daily

Use)* You want to multiply:

₹49 × 4
49 is not convenient.
Convert the form using Rūpam:

49 → 50 − 1 (standard form)
Now calculate:
50 × 4 = 200
1 × 4 = 4

Final answer:
200 − 4 = *196*

✓Rūpam = Convert 49 into a simpler form (50 – 1).

* ⭐ Example 2 — Cooking

Measurement* A recipe needs *¾ cup of

sugar*.
You want to cook *4 servings*.

Convert form:
¾ → 0.75 (Rūpam)

Now multiply:
0.75 × 4 = *3 cups*

✓Converted fraction to an easy form.

* ⭐ Example 3 — Filling Fuel*

You want to calculate fuel cost:
Petrol rate = ₹ *97 per L*
Fuel filled = *6 L*

Convert form:
97 → 100 − 3

Now calculate:
100 × 6 = 600
3 × 6 = 18
600 − 18 = *₹582*

✓Rūpam makes mental calculation fast.
* ⭐ Example 4 — Student Marks Conversion*
Marks scored: *37 out of 50*
Convert to percentage:
37/50
= (37 × 2) / 100 (Rūpam: convert denominator 50 → 100)
= 74%

✓Rūpam helps convert denominators to easier base values.

* ⭐ Example 5 — Unit Conversion (Real life)*

Distance = *2500 meters*
Convert to kilometers.

Use Rūpam:
1000 m = 1 km
2500 m = 2.5 km

✓Convert form → easy to interpret.

* ⭐ Example 6 — Discount Calculation*

A dress originally priced at ₹ *999* is discounted by *10%*.
Convert form:
999 → 1000 − 1

10% of 1000 = 100
10% of 1 = 0.1
Discount = 100 − 0.1 = *99.9 (≈100)*

✓Rūpam simplifies percentage calculations.

Summary
Rūpam = “Convert into an easier form for faster calculation.”

Used for:
* Approximations (98 → 100, 49 → 50)
* Fractions → decimals
* Converting units
* Mental multiplication
* Percentage and discount problems
* Standardizing expressions

4.16 Chidana : "By splitting."

*This sutra teaches us to *split a number or expression into smaller, easier parts* so that calculations become simple and fast.

It is similar to:
* Breaking numbers
* Splitting digits
* Dividing big problems into smaller chunks

Here is a simple example in Python demonstrating how to split a string using the built- in split() method:

Python split() Example

The split()method divides a string into a list of substrings based on a specified separator. If no separator is specified, any whitespace (spaces, tabs, newlines) is used.

```
# 1. Splitting a sentence by spaces
sentence = "Chidana means by splitting in Python" # The string is "split"
wherever a space is found words_list = sentence.split()

print(f"Original string: {sentence}")
print(f"Resulting list (split by whitespace): {words_list}")
# Output: Resulting list (split by whitespace): ['Chidana', 'means', 'by', 'splitting', 'in', 'Python']

print("-" * 20)

# 2. Splitting a string by a specific character (e.g., a comma)
data = "apple,banana,cherry,date"
# The string is "split" wherever a comma is found
fruits_list = data.split(',')

print(f"Original string: {data}")
print(f"Resulting list (split by comma): {fruits_list}")
```

* ⭐ Example 1 — Grocery Bill Splitting*
You buy items costing:
₹ *137* + ₹ *248*

Instead of adding directly:
Split numbers:
137 → 100 + 30 + 7
248 → 200 + 40 + 8

Now add:
(100 + 200) = 300
(30 + 40) = 70
(7 + 8) = 15
Total = 300 + 70 + 15 = *₹385*

✓Chidana makes big numbers easier by splitting into parts.

* ⭐ Example 2 — Multiplying Mentally*

Multiply: *34 × 6*
Split 34:
34 → 30 + 4

Now multiply:
30 × 6 = 180
4 × 6 = 24

Add: 180 + 24 = *204*

✓Splitting helps track time easily.

* ⭐ Example 4 — Splitting to Find

Area* Rectangle sides: *(10 + 3)* and *(8

+ 2)* Split both sides:

Area = (10 + 3)(8 + 2)
= 10×8 + 10×2 + 3×8 + 3×2
= 80 + 20 + 24 + 6
= *130*

✓Splitting makes area calculation simple.

* ⭐ Example 5 — Calculating Salary

Breakdown* Monthly salary = ₹ *25,750*
Split into easy parts:

25,750 → 20,000 + 5,000 + 700 + 50

This helps you plan savings or expenses.

Example: Save 10%:

2000 + 500 + 70 + 5 = *₹2,575 saved*

✓Splitting helps in budgeting.

* ⭐ Example 6 — Mental Square Calculation*
Find (52^2)
Split 52:
52 → 50 + 2

Use expansion:
(50 + 2)^2
= $50^2 + 2 \times 50 \times 2 + 2^2$
= 2500 + 200 + 4
= *2704*
✓Splitting makes squaring easy.
Summary

Chidana = "Split the number into smaller parts → calculate → combine."

Useful in:

* Shopping bills
* Mental multiplication
* Time calculations
* Area/measurement
* Budgeting
* Algebraic expansion