Savitribai Phule Pune University

(Formerly University of Pune)

T.Y.B.Sc.(Computer Science)

With

Major: Computer Science

(Faculty of Science and Technology)

(For Colleges Affiliated to Savitribai Phule Pune University)

Choice Based Credit System (CBCS) Syllabus Under National Education Policy (NEP)

To be implemented from Academic Year 2026-2027

Title of the Course: B.Sc.(Computer Science)

Savitribai Phule Pune University

(Formerly University of Pune)

T.Y.B.Sc.(Computer Science) Semester V

WORKBOOK

Course Code: CS-308-MJ-P

(Lab Course based on CS-307-MJ-T

WORKBOOK

T.Y.B.Sc. (Computer Science)

Data Science and Analytics

Course Type: Major Elective

Course Code: CS-308-MJ-P

Course Title: Lab Course based on CS-307-MJ-T

SEMESTER V

Student Name:			
College:			
Roll No:		**Exam Seat No:**	
Year:		**Division:**	

BOARD OF STUDIES

1. Dr. Patil Ranjeet	2. Dr. Joshi vinayak
3. Dr. Wani Vilas	4. Dr. Mulay Prashant
5. Dr. Sardesai Anjali	6. Dr. Shelar Madhukar
7. Dr. Bharambe Manisha	8. Dr. Deshpande Madhuri
9. Dr. Kardile Vilas	10. Dr. Korpal Ritambhara
11. Dr. Gangurde Rajendra	12. Dr. Manza Ramesh
13. Dr. Bachav Archana	14. Dr. Bhat Shridhar

Co-ordinators

- **Dr. Prashant Mulay**

Annasaheb Magar College, Hadapsar , Pune.

Member, BOS Computer Science, Savitribai Phule Pune University

- **Dr. Sardesai Anjali**

Modern College of Arts, Science and Commerce , Shivajinagar, Pune.

Member, BOS Computer Science, Savitribai Phule Pune University

Prepared and Edited by

Ms. Sunita J. Saykar	Annasaheb Magar Mahavidyalaya, Hadapsar, Pune

Prepared by:

Mrs. Ratna S. Chaudhari	Commerence, Management and Computer Science College, Nashik

Table of Contents

Introduction

About the workbook

This workbook is intended to be used by T.Y.B.Sc (Computer Science) students for the Lab Course based on **CS-307-MJ-T Data Science and Analytics**

In today's data-driven world, the ability to process, analyze, and interpret data is essential for solving real-world problems and making informed decisions. Data Processing, Data Visualization, and Machine Learning are key components of Data Science that transform raw data into meaningful insights and intelligent solutions. Data Processing focuses on preparing data through cleaning, transformation, and organization, ensuring its quality and usability. Data Visualization enables the effective representation of data through graphical techniques, helping to identify patterns, trends, and relationships. Machine Learning extends this capability by allowing systems to learn from data, make predictions, and support automated decision-making.

This laboratory provides hands-on experience with modern data analytics techniques using Python and industry-standard tools. It emphasizes practical learning through data exploration, visualization, and the implementation of machine learning models, thereby developing analytical thinking and problem-solving skills required in today's technology-driven environment.

Objectives

The primary objectives of this laboratory are:

- To understand the fundamental concepts of data processing, data visualization, and machine learning.
- To apply data cleaning, integration, transformation, reduction, and discretization techniques on real-world datasets.
- To utilize basic and advanced visualization tools for effective data analysis and presentation.
- To develop supervised and unsupervised machine learning models for solving practical problems.
- To evaluate machine learning models using appropriate performance metrics and validation techniques.
- To promote analytical thinking, ethical data handling, and informed decision-making.

How to Use This Workbook

The practical syllabus is organized into six assignments, designed to provide a progressive learning experience in Data Processing, Data Visualization, and Machine Learning. Depending on the complexity and scope of the topic, assignments may contain **Set A**, **Set B**, and in some cases, **Set C** exercises.

- **Set A** includes fundamental experiments and core implementations that introduce the essential concepts and techniques of the subject. Completion of Set A is **mandatory** for all students.
- **Set B** consists of advanced applications, extensions, or variations of the concepts covered in Set A. These exercises help students strengthen their practical skills and gain a deeper understanding of the topic. Students are encouraged to complete Set B whenever possible.

- **Set C**, where provided, contains exploratory and challenge-oriented exercises intended for self-learning and additional practice. These exercises enable students to investigate concepts in greater depth and enhance their analytical and problem-solving abilities.
- **Assignment 1** focuses on data preprocessing techniques such as cleaning, integration, transformation, reduction, and discretization, which form the foundation for reliable data analysis.
- **Assignments 2 and 3** introduce basic and advanced data visualization techniques, enabling students to explore, analyze, and communicate insights effectively through graphical representations.
- **Assignments 4 and 5** cover supervised machine learning algorithms for regression and classification, helping students build predictive models and evaluate their performance.
- **Assignment 6** explores unsupervised learning techniques, including clustering and association rule mining, for discovering hidden patterns and relationships within data.

  Students are encouraged to perform all experiments systematically, analyze the results obtained, and interpret the outcomes. Along with laboratory sessions, independent practice and exploration of additional datasets are recommended to strengthen conceptual understanding and develop practical skills in data analytics and machine learning.

Instructions to Students

- Bring this workbook to every lab session.
- Read the assignment problems and related concepts before coming to lab.
- Execute Python programs for the given questions and record the outputs clearly.
- Analyze the results and maintain proper documentation for each solution.
- The instructor will specify the problems to be solved during the lab session. Students must complete them within the allotted time and get them verified. Additional practice should be done in the lab and at home to complete as many problems from the workbook as possible.

Assessment Scheme (0–5 Scale)

Each exercise will be evaluated as follows:

- Not Done – 0
- Incomplete – 1
- Late Completion – 2
- Needs Improvement – 3
- Complete – 4
- Well Done – 5

Instructions to the Practical In-Charge

- Introduce each assignment by explaining the required concepts and objectives.
- Allocate problems for practice: ensure **Set A is completed by all students**, assign Set B based on time availability, and encourage discussion and practice of Set C for deeper understanding.
- Ensure that students adhere to all laboratory guidelines and complete the assigned tasks within the session.
- Assess each student's performance for every assignment on a **0–5 scale** using the prescribed criteria.
- Record the awarded marks in the respective assignment completion page of the lab record.

Instructions to the Lab Administrator and Exam Guidelines

- Ensure that all required hardware and software resources are properly installed and available for each student.
- Internet access must be disabled in the computer laboratory during examinations.
- Use of external storage devices such as pen drives must not be permitted during examinations.

Laboratory Environment

- **Operating System:** Linux
- **Programming Language:** Python 3.8+
- **Code Editor/IDE:** Visual Studio Code (VS Code) / Jupyter Notebook
- **Install the required libraries**

Dataset Sources –

Download the required dataset from different online platform like kaggle, github etc. Do the appropriate changes as per the requirement of lab assignment.

Some Data source links -

https://www.kaggle.com/datasets/sivaram1987/association-rule-learningapriori

https://github.com/shivang98/Market-Basket-Optimization

https://www.kaggle.com/datasets/hemanthkumar05/market-basket-optimization

https://www.kaggle.com/datasets/irfanasrullah/groceries

Assignment Completion Sheet

Sr. No	Assignment Title	Marks	Signature
1	Data Pre-Processing Techniques		
2	Basic Data Visualization Tools		
3	Advanced Data Visualization tools		
4	Supervised Machine Learning Models for Regression		
5	Supervised Machine Learning Models for Classification		
6	Unsupervised Machine Learning Models for Clustering (K-means clustering) and Association Rule Mining		
Total out of 30			

This is to certify that **Mr/Ms** __

University Exam Seat Number __have successfully and satisfactorily completed and submitted the course work for **T.Y.B.Sc.(CS)** Lab course CS-308-MJ-P Semester V prescribed by Savitribai Phule Pune University during the academic year __.

Instructor **Head of Department**

Internal Examiner **External Examiner**

Assignment 1

Data Pre-Processing Techniques – Cleaning, Integration, Transformation, Reduction, and Discretization **No. of slots = 02**

Objectives

1. To understand the importance of data preprocessing and its role in improving data quality for data mining and machine learning applications.
2. To perform Data Cleaning and Data Integration techniques for handling missing values, inconsistencies, and combining data from multiple sources.
3. To apply Data Transformation techniques such as Rescaling, Normalization, Binarization, Standardization, Label Encoding, and One-Hot Encoding.
4. To implement Data Reduction methods including Dimensionality Reduction, Feature Selection, Feature Extraction, Data Cube Aggregation, and Numerosity Reduction.
5. To perform Data Discretization using Equal Width Binning and Equal Frequency Binning for converting continuous data into discrete intervals.

Data Preprocessing -

Data reprocessing is the process of converting raw, incomplete, inconsistent, and noisy data into a clean and understandable format before applying data mining, machine learning, or statistical analysis techniques.

Real-world data collected from various sources often contains missing values, duplicate records, errors, inconsistencies, and irrelevant information. Such data can negatively affect the performance and accuracy of analytical models. Therefore, preprocessing is considered one of the most important steps in the data analysis pipeline.

Need for Data Preprocessing

- Real-world data is often incomplete and may contain missing values.
- Data collected from different sources can be inconsistent.
- Large datasets may contain redundant or irrelevant features.
- Different attributes may have different scales and units.
- Machine learning algorithms require data in a structured and numerical format.

Objectives of Data Preprocessing

- Improve data quality and reliability.
- Increase the accuracy of machine learning models.
- Reduce processing time and computational cost.
- Remove noise, redundancy, and inconsistencies.
- Transform data into a suitable format for analysis.

Major Steps in Data Preprocessing

1. Data Cleaning

Data Cleaning is the process of identifying and correcting errors, inconsistencies, and inaccuracies in a dataset to improve its quality. This includes handling missing values, removing duplicates, detecting outliers, and standardizing formats. The goal is to ensure that the data is accurate, complete, and usable for analysis, leading to more reliable insights and better decision-making.

```
import pandas as pd
# User-defined DataFrame
df = pd.DataFrame({
'Name': ['Amit', 'Vihaan', 'Amit', 'Ishita', 'Raj',
'Priya', 'Kapil'],
'Salary': [50000, None, 50000, None, 700000,
55000, 58000],
'Age': [25, 30, 25, 22, 28, None, None],
'Date': ['01/05/2024', '15/06/2024', '01/05/2024',
'20/07/2024', '10/08/2024', '25/09/2024',
'12/10/2024'] })
print("Original Data:")
print(df)

# Handle missing values
df['Salary'].fillna(0, inplace=True)
mean_age = df['Age'].mean()
df['Age'].fillna(mean_age, inplace=True)

# Remove duplicate rows
df.drop_duplicates(inplace=True)

# Detect outliers using IQR
Q1 = df['Salary'].quantile(0.25)
Q3 = df['Salary'].quantile(0.75)
IQR = Q3 - Q1
outliers = df[
   (df['Salary'] < Q1 - 1.5 * IQR) |
   (df['Salary'] > Q3 + 1.5 * IQR)]
print("\n Outliers:")
print(outliers)

#Standardize formats
df['Name'] = df['Name'].str.lower()
df['Date'] = pd.to_datetime(df['Date'],
dayfirst=True).dt.strftime('%Y-%m-%d')
print("\n Cleaned Data:")
print(df)
```

Output –

Orginial Data:

	Name	Salary	Age	Date
0	Amit	50000.0	25.0	01/05/2024
1	Vihaan	NaN	30.0	15/06/2024
2	Amit	50000.0	25.0	01/05/2024
3	Ishita	NaN	22.0	20/07/2024
4	Raj	700000.0	28.0	10/08/2024
5	Priya	55000.0	NaN	25/09/2024
6	Kapil	58000.0	NaN	12/10/2024

Outliers –

	Name	Salary	Age	Date
4	Raj	700000.0	28.0	10/08/2024

Cleaned Data

	Name	Salary	Age	Date
0	amit	50000.0	25.000000	2024-05-01
1	vihaan	0.0	30.000000	2024-06-15
3	ishita	0.0	22.000000	2024-07-20
4	raj	700000.0	28.000000	2024-08-10
5	priya	55000.0	26.333333	2024-09-25
6	kapil	58000.0	26.333333	2024-10-12

Changes made:

1) Missing values in the Salary column were replaced with 0.
2) Missing values in the Age column were replaced with the mean age.
3) Duplicate rows were removed from the dataset.
4) Outliers in the Salary column were detected using the IQR method.
5) All names were converted to lowercase.
6) All dates were standardized to the YYYY-MM-DD format.

2. Data Integration

Data Integration is the process of collecting, combining, and consolidating data from different sources to provide a unified and consistent dataset for analysis and decision-making.

Techniques used:

- Schema Matching: Aligns attributes from different data sources.
- Entity Resolution: Identifies records that refer to the same real-world entity.
- Correlation Analysis: Finds and removes redundant attributes.
- Data Conflict Resolution: Resolves inconsistencies in units or data values.
- Duplicate Elimination: Removes overlapping records after integration.

```
import pandas as pd
# Dataset 1: Student details
df1 = pd.DataFrame({ 'ID': [1, 2, 3],
   'Name': ['Aarav', 'Diya', 'Rohan']
})
print("Data Frame1:\n")
print(df1)
# Dataset 2: Marks + City
df2 = pd. DataFrame({
   'ID': [1, 2, 3],
   'Name': ['Aarav', 'Diya', 'Rohan'],
   'Marks': [85, 90, 78],
   'City': ['Surat', 'Mumbai', 'Delhi']
})
print("Data Frame2:\n")
print(df2)
# 1. Schema Matching (standardize column names)
df1.columns = df1.columns.str.lower()
df2.columns = df2.columns.str.lower()
```

Output

Data Frame 1:

	id	name
0	1	Aarav
1	2	Diya
2	3	Rohan

Data Frame 2:

	id	name	marks	city
0	1	Aarav	85	Surat
1	2	Diya	90	Mumbai
2	3	Rohan	78	Delhi

```
# 2. Entity Resolution + Integration (merge datasets)
df = pd.merge(df1, df2, on='id', how='outer')
# 3. Duplicate Elimination
df.drop_duplicates(inplace=True)
# 4. Resolve column conflict (name columns)
df['name'] = df['name_x'].combine_first(df['name_y'])
df.drop(['name_x', 'name_y'], axis=1, inplace=True)
# 5. Display final integrated dataset
print("Integrated Dataset:")
print(df)
```

Integrated Dataset:

	id	marks	city	name
0	1	85	Surat	Aarav
1	2	90	Mumbai	Diya
2	3	78	Delhi	Rohan

3. Data Transformation

Data transformation is a preprocessing step used to convert raw data into a suitable format for machine learning models. It helps improve data quality, consistency, and compatibility with different algorithms. It is required because real-world data often has different scales, units, and formats.

Techniques:

A. Rescaling: Rescaling (Min–Max Scaling) is a data transformation technique used to scale numeric values into a fixed range, usually 0 to 1. It helps in improving the performance of machine learning algorithms by bringing all features to the same scale.

Formula:

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

Function: MinMaxScaler()

Syntax:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)
```

B. Normalization: Normalization is a data preprocessing technique used to scale numerical features to a common range (usually 0 to 1 or -1 to 1). It helps improve model accuracy by ensuring that all features contribute equally and prevents features with larger values from dominating the learning process.

Formula:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Function: Normalizer()

Syntax:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalized_data = normalizer.fit_transform(data)
```

C. Binarization: Binarization is the process of converting data into 0s and 1s based on a threshold. If a value is above the threshold, it becomes 1, otherwise 0. It is used in Machine Learning to simplify data and make it suitable for models that work with binary inputs.

Function: Binarizer()

Syntax:

```
from sklearn.preprocessing import Binarizer
binarizer = Binarizer(threshold=0.5)
binary_data = binarizer.fit_transform(data)
```

D. Standardization: Standardization is a preprocessing technique in Machine Learning where data is transformed to a common scale by removing the mean and scaling by the standard deviation, so that features with different units or ranges can be compared fairly.

Formula:

$$z = \frac{x - \mu}{\sigma}$$

Where:

x = original value, μ = mean of data

σ = standard deviation z = standardized value

Function: StandardScaler()

Syntax:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
standardized_data = scaler.fit_transform(data)
```

E. Label Encoding: Label Encoding is a data preprocessing technique used to convert categorical values into numerical labels. It assigns a unique integer to each category, allowing machine learning algorithms to process categorical data in a numerical format. The encoded labels are typically assigned based on the sorted order of unique categories.

Function: LabelEncoder()

Syntax:

```
from sklearn.preprocessing import LabelEncoder
```

encoder = LabelEncoder()

encoded_data = encoder.fit_transform(data)

F. One-Hot Encoding: One-Hot Encoding is a data preprocessing technique used to convert categorical data into a numerical format that machine learning models can understand. It creates separate binary columns for each category, where 1 represents the presence of a category and 0 represents its absence.

Function: get_dummies() (Pandas)

Syntax:

import pandas as pd

one_hot_data = pd.get_dummies(data)

Python code for data transformation techniques

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler,
Normalizer, Binarizer, StandardScaler, LabelEncoder
# Sample Data
numeric_data = np.array([[10], [20], [30], [40]])

multi_data = np.array([[1, 2], [3, 4], [5, 6]])

categorical_data = ["Red", "Blue", "Green", "Blue"]

print("Original Numeric Data:\n", numeric_data)

print("\nOriginal Multi Data:\n", multi_data)

print("\nOriginal Categorical Data:\n",
categorical_data)

# Rescaling function
rescale =
MinMaxScaler().fit_transform(numeric_data)
print("\nRescaled Data:\n", rescale)
```

Output –

```
Original Numeric Data:
[[10]
[20]
[30]
[40]]

Original Multi Data:
[[1 2]
[3 4]
[5 6]]

Original Categorical Data:

['Red', 'Blue', 'Green', 'Blue']

Rescaled Data:
[[0.        ]
 [0.33333333]
 [0.66666667]
 [1.        ]]

Normalized Data:
[[0.4472136  0.89442719]
 [0.6        0.8       ]
 [0.6401844  0.76822128]]
```

```
# Normalization function
normalize = Normalizer().fit_transform(multi_data)
print("\nNormalized Data:\n", normalize)
# Binarization function
binarize =
Binarizer(threshold=15).fit_transform(numeric_data)
print("\nBinarized Data:\n", binarize)

# Standardization function
standardize =
StandardScaler().fit_transform(numeric_data)
print("\nStandardized Data:\n", standardize)

# Label Encoding function
label_encode =
LabelEncoder().fit_transform(categorical_data)
print("\nLabel Encoded Data:\n", label_encode)

# One-Hot Encoding function
one_hot = pd.get_dummies(pd.DataFrame({"Color":
categorical_data}), columns=["Color"])
print("\nOne-Hot Encoded Data:\n", one_hot)
```

Binarized Data:

```
[[0]
 [1]
 [1]
 [1]]
```

Standardized Data:

```
[[-1.34164079]
 [-0.4472136 ]
 [ 0.4472136 ]
 [ 1.34164079]]
```

Label Encoded Data:

```
[2 0 1 0]
```

One-Hot Encoded Data:

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

4. Data Reduction

Data reduction is a technique used in data mining to reduce the size of a dataset while still preserving the most important information. This can be beneficial in situations where the dataset is too large to be processed efficiently, or where the dataset contains a large amount of irrelevant or redundant information.

Techniques:

- **Dimensionality Reduction**

  It reduces the number of input variables (features) by transforming the data into a lower-dimensional space. It keeps most of the important information while removing redundancy and noise. This helps improve model performance and reduces computation time.

Function –

from sklearn.decomposition import PCA pca = PCA(n_components=2) X_reduced = pca.fit_transform(X)	**Explanation -** 1. **from sklearn.decomposition import PCA** - imports PCA (Principal Component Analysis), a method used to reduce the number of features in a dataset. 2. **pca = PCA(n_components=2)** creates a PCA model that reduces the data into 2 principal components (new important features). 3. **X_reduced = pca.fit_transform(X)** applies PCA on dataset X, learns the main patterns, and transforms the data into a lower-dimensional form with 2 features.

- **Feature Selection**

  It selects only the most relevant features from the dataset and removes irrelevant or less important ones. Unlike dimensionality reduction, it does not create new features; it just filters existing ones.

Function -

from sklearn.feature_selection import SelectKBest, chi2 X_selected = SelectKBest(chi2, k=2).fit_transform(X, y)	**Explanation -** 1. **SelectKBest** selects the best features from the dataset based on a scoring method. 2. **chi2** is the Chi-square statistical test used to measure the relationship between each feature and the target variable. 3. **k=2** means that only the top 2 most important features will be selected from the dataset. **4. fit_transform(X, y)** first learns which features are most relevant using the input data X and target y, and then transforms the dataset by keeping only those selected features.

Feature Extraction

It transforms original features into a new set of features by combining or modifying them. The new features are usually more compact and informative (e.g., PCA, LDA). It helps improve model learning by representing data in a better form.

Function -

from sklearn.decomposition import PCA X_new_features = PCA(n_components=2).fit_transform(X)	**from sklearn.decomposition import PCA** - imports the PCA method used for dimensionality reduction. **PCA(n_components=2)** creates a PCA model that reduces the dataset into 2 main components (new features). **fit_transform(X)** fits the PCA model on dataset X and transforms it into a reduced form. **X_new_features = ...** stores the transformed dataset with 2 new features in the variable X_new_features

- **Data Cube Aggregation**

  This technique is used to aggregate data in a simpler form. Data Cube Aggregation is a multidimensional aggregation that uses aggregation at various levels of a data cube to represent the original data set, thus achieving data reduction.

Function -

```
import pandas as pd
df.groupby("column").agg("sum")
```

- **Numerosity Reduction**

  It reduces data volume by replacing large datasets with smaller representations without losing much information. Methods include sampling, clustering, and regression models.

Function -

```
df_reduced = df.sample(frac=0.5)
```

Pyhton code for techniques of data reduction –

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.feature_selection import
SelectKBest, chi2
from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import
MinMaxScaler
df = pd.DataFrame({
    "Study_Hours": [2, 4, 6, 8, 10],
    "Attendance": [60, 70, 80, 90, 95],
    "Quiz_Score": [50, 55, 75, 85, 90],
    "Assignment": [3, 5, 7, 9, 10],
    "Pass": [0, 0, 1, 1, 1]
})
print("\n Original Data \n")
print(df)

# Split Features
X = df.drop("Pass", axis=1)
y = df["Pass"]
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Dimensionality Reduction
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
print("\n Dimensionality Reduction (PCA) \n")
print(pd.DataFrame(X_pca, columns=["PC1",
"PC2"]))
# Feature Selection (SelectKBest_chi2)
fs = SelectKBest(score_func=chi2, k=2)
X_selected = fs.fit_transform(X_scaled, y)
print("\n Feature Selection \n")
print(X_selected)
# Feature Extraction
df["Performance_Score"] = (
    df["Study_Hours"] * 0.3 +
    df["Attendance"] * 0.3 +
    df["Quiz_Score"] * 0.3 +
    df["Assignment"] * 0.1 )
print("\n Feature Extraction \n")
print(df[[
    "Study_Hours",
    "Attendance",
    "Quiz_Score",
    "Assignment",
    "Performance_Score" ]])
# Data Cube Aggregation
print("\n Data Cube Aggregation \n")
print(df.groupby("Pass").agg({
    "Study_Hours": "mean",
```

Output:

Original Data

	Study_Hours	Attendance	Quiz_Score	Assignment	Pass
0	2	60	50	3	0
1	4	70	55	5	0
2	6	80	75	7	1
3	8	90	85	9	1
4	10	95	90	10	1

Dimensionality Reduction

[[-6.8 2.1]

[-3.2 0.8]

[0.5 -0.6]

[3.7 -1.2]

[5.8 -1.1]]

Feature Selection

[[2 60 50]

[4 70 55]

[6 80 75]

[8 90 85]

[10 95 90]]

Feature Extraction

	Study_Hours	Attendance	Quiz_Score	Assignment	Performance_Score
0	2	60	50	3	33.9
1	4	70	55	5	38.4
2	6	80	75	7	47.4
3	8	90	85	9	53.1
4	10	95	90	10	56.4

Data Cube Aggregation

Pass	Study_Hours	Attendance	Quiz_Score	Assignment
0 (Fail)	3.0	65.0	52.5	4.0
1 (Pass)	8.0	88.3	83.3	8.7

```
    "Attendance": "mean",
    "Quiz_Score": "mean",
    "Assignment": "mean"}))
# Numerosity Reduction
X_train, X_reduced, y_train, y_reduced =
train_test_split(
    X_scaled, y,
    test_size=0.4,
    stratify=y,
    random_state=42
)
print("\n numerosity reduction \n")
print(X_reduced)
```

Numerosity Reduction

	Study_Hours	Attendance	Quiz_Score	Assignment	Pass
1	4	70	55	5	0
3	8	90	85	9	1
4	10	95	90	10	1

5. Data Discretization

Data discretization is a preprocessing technique that Transforms continuous data into discrete intervals or bins, making it easier to analyze and interpret. It reduces data complexity, improves interpretability, and can enhance the performance of classification and clustering algorithms.

Techniques -

5.1 Equal Width Binning

In Equal Width Binning, the range of data values is divided into k intervals of equal size. Each interval has the same width, but the number of data points in each bin may vary.

Formula

$$Width = \frac{Maximum - Minimum}{Number\ of\ Bins}$$

Example

Data = {10, 20, 30, 40, 50, 60, 70, 80}

Number of bins = 4

Maximum Value = 80, Minimum Value = 10

Range = 80 − 10 = 70

Bin Width = (Maximum Value − Minimum Value) / Number of Bins

Bin Width = (80 − 10) / 4

Bin Width = 17.5

Bins:

- Bin 1: 10 – 27.5
- Bin 2: 27.5 – 45.0
- Bin 3: 45.0 – 62.5
- Bin 4: 62.5 – 80.0

5.2 Equal Frequency Binning

Equal Frequency Binning divide continuous data into bins such that each bin contains approximately the same number of observations. Unlike Equal Width Binning, the widths of the intervals may vary, but the frequency (number of data points) in each bin remains nearly equal.

This method is also known as **Quantile Binning** because the bins are created based on quantiles of the data distribution.

Equal Frequency Binning is particularly useful when the data is unevenly distributed, as it ensures a balanced number of records in each bin.

Example

Consider the dataset:

Data = {10, 20, 30, 40, 50, 60, 70, 80}

Number of bins = 4

Since there are 8 values and 4 bins, each bin will contain 2 values.

Bin	Values
Bin 1	10, 20
Bin 2	30, 40
Bin 3	50, 60
Bin 4	70, 80

Thus, each bin contains an equal frequency of observations.

Python code for Data Discretization-

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import KBinsDiscretizer

# Sample Dataset
data = np.array([10, 20, 30, 40, 50, 60, 70, 80]).reshape(-1, 1)
df = pd.DataFrame(data, columns=['Values'])

# Equal Width Binning
equal_width = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')

df['Equal_Width_Bin'] = equal_width.fit_transform(data)

# Equal Frequency Binning
equal_freq = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='quantile')
```

Output:

	Values	Equal_Width_Bin	Equal_Frequency_Bin
0	10	0.0	0.0
1	20	0.0	0.0
2	30	0.0	0.0
3	40	1.0	1.0
4	50	1.0	1.0
5	60	2.0	1.0
6	70	2.0	2.0
7	80	2.0	2.0

Strategy used –

Uniform: Divides the data range into bins of equal width, where each bin has the same interval size regardless of how many data points it contains.

df['Equal_Frequency_Bin'] = equal_freq.fit_transform(data) **# Display Result** print(df)	**Quantile:** Divides the data into bins such that each bin contains approximately the same number of data points, ensuring equal frequency distribution across bins.

Lab Assignment

SET A

1. Write a Python program to create an "employee" DataFrame with the following attributes: Employee_ID, Age, Department, Experience, and Performance_Score. Perform data cleaning by handling missing values, removing duplicates, and detecting outliers using the IQR method.
2. Write a Python program to create dataframes student_info, student_result and perform data integration on them.
3. Write a Python program to create a DataFrame using sales_data.csv with the attributes Product_ID, Price, Quantity_Sold, Discount, and Revenue. Apply data transformation techniques on the DataFrame and display the transformed data.
   a) Min-Max Scaling
   b) Standardization
   c) Normalization
4. Write a Python program to create a dataframe using customer_data.csv and perform encoding techniques:
   [Consider appropriate attributes for following operations]
   a) Label Encoding
   b) One-Hot Encoding
5. Write a Python program to create a dataframe using age_data.csv and perform Data Discretization using:
   a) Equal Width Binning
   b) Equal Frequency Binning

SET B

1. Write a Python program to create a dataframe using iris flower dataset and reduce 4D data to 2D data using PCA.
2. Write a Python program to perform data reduction techniques as
   a) Feature Selection to identify important features for disease prediction.
   b) Feature extraction
   (Use Patient_health.csv – Pno, Age, BP, Chol, Sugar(0 = No, 1 = Yes), BMI, Disease)

3. Load Superstore dataset(Order_ID, Region, Category, Sales,Profit, Discount) and perform Data Cube Aggregation to analyze region-wise and category-wise sales performance.
4. Load Employee(eid, age, workclass, education, hours-per-week, income) dataset and perform Numerosity Reduction.

SET C

Case Study

A small online store wants to analyze customer behavior to identify whether a customer is a **High Spender** or **Low Spender**. The company collects data from two different sources: customer profile details and purchase history. However, the data contains missing values, needs integration, transformation, reduction, and discretization before building a prediction model

Write a Python program to load required data sets and apply data preprocessing techniques and display the cleaned data.

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion: __/___/____**

Assignment 2

Basic Data Visualization Tools

Number of Slots = 01

Objectives

- To understand the concept and importance of Data Visualization.
- To learn the usage of the Matplotlib library for creating visualizations.
- To create and customize different types of charts and graphs.
- To analyze datasets using graphical representations.

Data Visualization is the process of representing data in a graphical or pictorial form using charts, graphs, and plots. It helps users understand complex datasets by presenting information in a clear and meaningful way. Data visualization enables the identification of patterns, trends, relationships, and outliers that may not be easily observed from raw data. It is an important component of Exploratory Data Analysis (EDA) and plays a significant role in data analysis, decision-making, and communication of results.

In this assignment, we focus on basic data visualization tools using the Matplotlib library. Students will learn how to create and customize commonly used charts such as Histograms, Bar Charts, Scatter Plots, Line Charts, Area Plots, Pie Charts, Donut Charts, and Bubble Plots. These visualizations help in understanding data distribution, comparison, trends, and relationships between variables.

Steps Involved in our Visualization

1. Importing packages
2. Importing and Cleaning Data
3. Creating Visualizations

Components of a Graph

A graph consists of several attributes that help in presenting data clearly and effectively.

Sr.No	Attributes	Meaning
1.	Title	A title provides a brief description of the graph and helps users understand its purpose. plt.title("Student Marks Analysis")
2.	X-Axis	The X-axis (horizontal axis) represents categories or independent variables. plt.xlabel("Student_name")
3.	Y-Axis	The Y-axis (vertical axis) represents values or dependent variables. plt.ylabel("Marks")

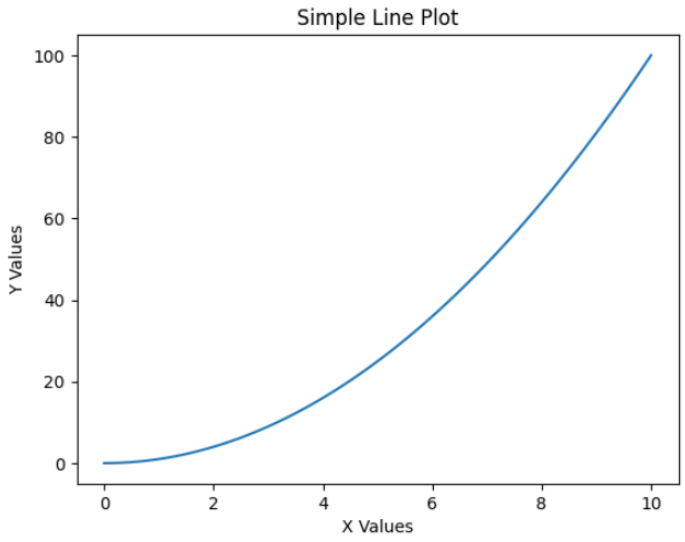
<table>
<tr><td>4.</td><td>Legend</td><td>A legend identifies different datasets displayed in a graph.
plt.legend()Example:
plt.plot(x, y, label="Sales")
plt.legend()</td></tr>
<tr><td>5.</td><td>Grid</td><td>Grid lines improve readability and help estimate values accurately.
plt.grid(True)</td></tr>
<tr><td>6.</td><td>Figure Size</td><td>Figure size controls the width and height of the graph.
plt.figure(figsize=(8,5))</td></tr>
<tr><td>7.</td><td>Color</td><td>Colors improve the visual appearance of graphs.
plt.plot(x, y, color='red')
Common colors:
red ,blue,green,yellow,orange,black,purple</td></tr>
<tr><td>8.</td><td>Marker</td><td>Markers highlight data points.
plt.plot(x, y, marker='o')<table><tr><th>Marker</th><th>Shape</th></tr><tr><td>o</td><td>Circle</td></tr><tr><td>s</td><td>Square</td></tr><tr><td>^</td><td>Triangle</td></tr><tr><td>*</td><td>Star</td></tr><tr><td>+</td><td>Plus</td></tr></table></td></tr>
<tr><td>9.</td><td>Line Style</td><td>Changes the appearance of lines.
plt.plot(x, y, linestyle='--')<table><tr><th>Style</th><th>Meaning</th></tr><tr><td>-</td><td>Solid Line</td></tr><tr><td>--</td><td>Dashed Line</td></tr><tr><td>….</td><td>Dotted Line</td></tr><tr><td>-.</td><td>Dash-Dot Line</td></tr></table></td></tr>
</table>

Create a Simple Plot

Before learning different types of data visualizations, let us create a simple plot using Matplotlib. The simplest way to create a graph in Matplotlib is by using the plot() function.

A plot requires two sets of values: one for the **X-axis** and another for the **Y-axis**. In this example, we will generate a sequence of values using NumPy's linspace() function and then create a simple line plot.

The linspace() function generates evenly spaced values over a specified range, making it useful for creating sample data for visualization.

import numpy as np import matplotlib.pyplot as plt x = np.linspace(0, 10, 100) y = x**2 plt.plot(x, y) plt.title("Simple Line Plot") plt.xlabel("X Values") plt.ylabel("Y Values") plt.show()	Output: 

Output:

A simple line graph showing the relationship between the X values and their corresponding squared values ($Y = X^2$).

Explanation:

- np.linspace(0, 10, 100) generates 100 evenly spaced values between 0 and 10.
- plt.plot() creates the line graph.
- plt.title() adds a title to the graph.
- plt.xlabel() labels the X-axis.
- plt.ylabel() labels the Y-axis.
- plt.show() displays the graph.

Choosing the Right Graph Based on Data Type

Before creating a visualization, it is important to understand the type of data being analyzed. Different types of data require different visualization techniques. Choosing the appropriate graph helps in presenting information accurately and effectively.

Types of Data

Data can be broadly classified into two categories:

1. Categorical Data (Qualitative Data)

Categorical data represents groups, categories, labels, or characteristics rather than numerical measurements. The values in categorical data are used to classify observations into different groups and cannot be used for mathematical calculations.

Categorical data helps answer questions such as "Which category?" or "What type?" rather than "How much?" or "How many?".

<table>
<tr>
<th>Examples</th>
<th>Sample Data</th>
</tr>
<tr>
<td>

Department Names
Product Categories
Cities
Gender
Blood Groups

</td>
<td>
<table>
<tr><th>Department</th><th>Students</th></tr>
<tr><td>Computer Science</td><td>120</td></tr>
<tr><td>IT</td><td>100</td></tr>
<tr><td>Electronics</td><td>80</td></tr>
</table>
</td>
</tr>
</table>

Suitable Graphs

- Bar Chart
- Pie Chart
- Donut Chart

These graphs help compare categories and display the proportion of each category within a dataset.

2. Numerical Data (Quantitative Data)

Numerical data consists of measurable quantities represented by numbers. Unlike categorical data, numerical data can be used for mathematical operations such as addition, subtraction, averaging, and statistical analysis.

Numerical data answers questions such as "How much?", "How many?", or "How often?".

Examples

- Marks
- Salary
- Height
- Weight
- Temperature

Numerical data is further classified into Discrete Data and Continuous Data.

a) Discrete Data

Discrete data consists of distinct, countable values. These values are usually whole numbers and cannot take fractional or decimal values in most situations. Discrete data is obtained through counting rather than measurement.

Examples	Sample Data
• Number of Students • Number of Employees • Number of Cars • Number of Books • Number of Customers	**Class** **No of Students** FY 60 SY 55 TY 50

Suitable Graphs

- Bar Chart
- Pie Chart

These visualizations are useful for comparing counts across different categories and showing the proportion of each category.

b) Continuous Data

Continuous data consists of values that can take any value within a specified range. These values are obtained through measurement and may include decimal or fractional values.

Continuous data is often used to study distributions, trends, and variations in data.

Examples	Sample Data
• Height • Weight • Age • Temperature • Salary • Time	**Student** **Marks** A 65.25 B 78.00 C 85.12

Suitable Graphs

- Histogram
- Line Chart
- Area Plot

These graphs help analyze distributions, identify trends, and understand how values change over a range or over time.

Basic Data Visualization Tools

The following are some of the most commonly used data visualization tools in Matplotlib. Each graph is designed for a specific type of data analysis and provides unique insights into the dataset.

1. Histogram

A Histogram is used to represent the frequency distribution of continuous numerical data. It divides data into intervals called bins and displays the frequency of observations within each interval. Histograms help understand data distribution, spread, and patterns.

Function

- The function used for Histogram is plt.hist().
- It requires a list of numerical values.
- Data is grouped into bins and displayed as adjacent bars.

Customizations

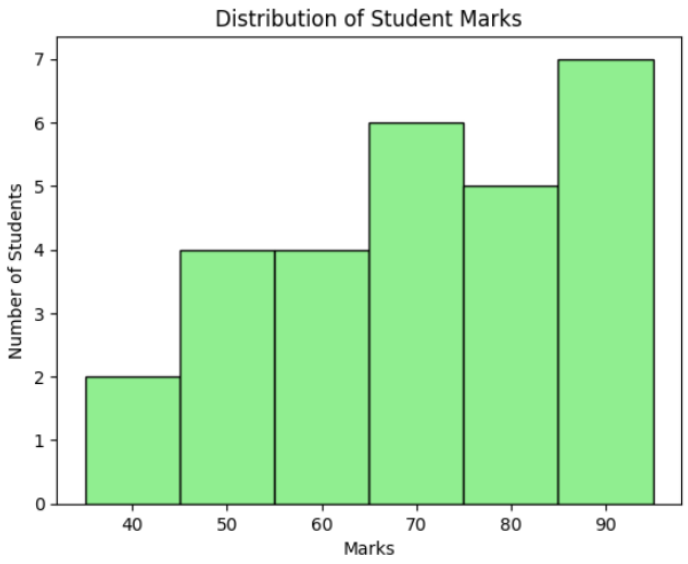
plt.hist() function supports the following arguments:

- bins – Specifies the number of intervals.
- color – Sets the bar color.
- edgecolor – Sets the border color.
- density – Displays probability density.
- cumulative – Creates cumulative histogram.
- label – Adds a legend label.

Example:

```
import matplotlib.pyplot as plt

marks = [35, 42, 45, 48, 50, 52, 55, 58,
60, 62, 65, 67, 69, 70, 72, 74, 76, 78, 80,
82, 84, 85, 87, 88, 90, 92, 94, 95]
plt.hist(marks, bins=6,
color='lightgreen', edgecolor='black')
plt.title("Distribution of Student Marks")
plt.xlabel("Marks")
plt.ylabel("Number of Students")
plt.show()
```

Output:



2. Bar Chart

A Bar Chart is used to compare values among different categories. Each category is represented by a rectangular bar whose height corresponds to its value.

Function

- The function used for Bar Chart is plt.bar().
- It requires category names and corresponding values.

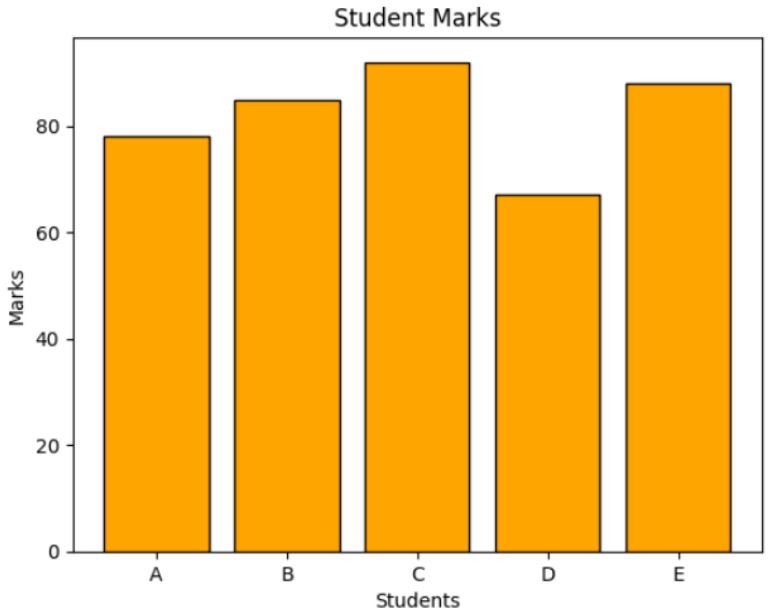
Customizations

plt.bar() supports:

- color – Sets bar color.
- width – Controls bar width.
- edgecolor – Sets border color.
- label – Adds legend.

```
import matplotlib.pyplot as plt
students = ['A','B','C','D','E']
marks = [78,85,92,67,88]
plt.bar(students, marks,
color='orange', edgecolor='black')
plt.title("Student Marks")
plt.xlabel("Students")
plt.ylabel("Marks")
plt.show()
```

Output:



3. Scatter Plot

A Scatter Plot is used to show the relationship between two numerical variables. Each point represents one observation.

Function

- The function used for Scatter Plot is plt.scatter().
- Requires X and Y values.

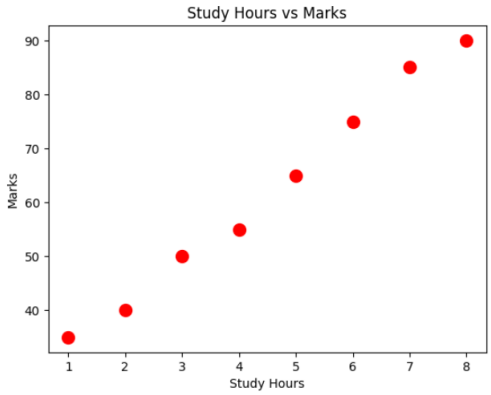
Customizations

plt.scatter() supports:

- color – Point color.
- marker – Shape of points.

- s – Size of points.
- alpha – Transparency level.

Example

import matplotlib.pyplot as plt study_hours = [1,2,3,4,5,6,7,8] marks = [35,40,50,55,65,75,85,90] plt.scatter(study_hours,marks,color='red',s=100) plt.title("Study Hours vs Marks") plt.xlabel("Study Hours") plt.ylabel("Marks") plt.show()	Output: 

4. Line Chart

A Line Chart is used to display trends or changes over time. Data points are connected using straight lines.

Function

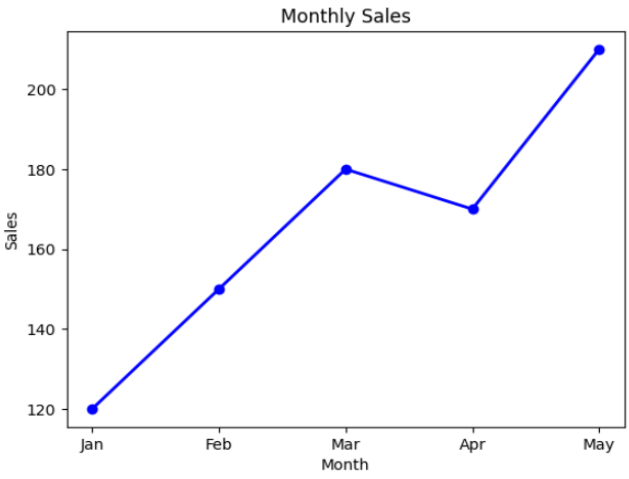
- The function used for Line Chart is plt.plot().

Customizations

plt.plot() supports:

- **color** – Sets the color of the line.
- **linewidth** – Specifies the thickness of the line.
- **linestyle** – Defines the style of the line (solid, dashed, dotted, etc.).
- **marker** – Displays markers at the data points on the line.

Example

import matplotlib.pyplot as plt months = ['Jan','Feb','Mar','Apr','May'] sales = [120,150,180,170,210] plt.plot(months,sales, color='blue',marker='o',linewidth=2) plt.title("Monthly Sales") plt.xlabel("Month") plt.ylabel("Sales") plt.show()	Output: 

5. Area Plot

An Area Plot is similar to a line chart but the area below the line is filled with color. It is useful for showing cumulative trends.

Function

- The function used for Area Plot is plt.fill_between().

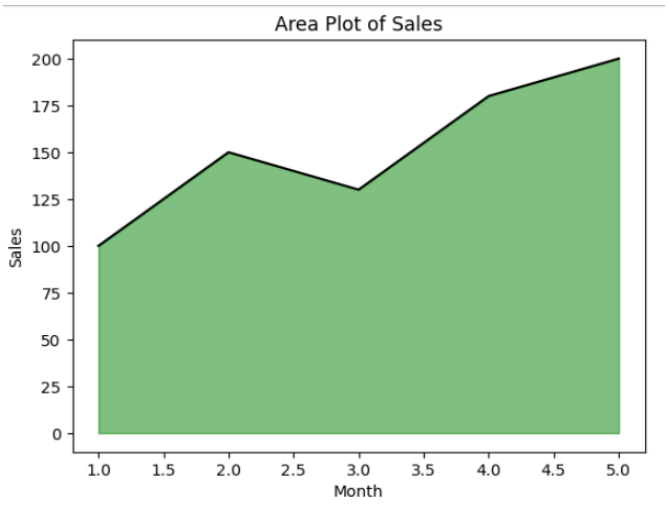
Customizations

- **color** – Sets the fill color of the area.
- **alpha** – Controls the transparency level of the area.
- **label** – Adds a label for the legend.

Example

```
import matplotlib.pyplot as plt
months = [1,2,3,4,5]
sales = [100,150,130,180,200]
plt.fill_between (months,sales,
color='green', alpha=0.5)
plt.plot(months,sales,color='black')
plt.title("Area Plot of Sales")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.show()
```

Output:



6. Pie Chart

A Pie Chart displays the proportion of each value against the total sum. Each slice represents a percentage contribution.

Function

- The function used for Pie Chart is plt.pie().

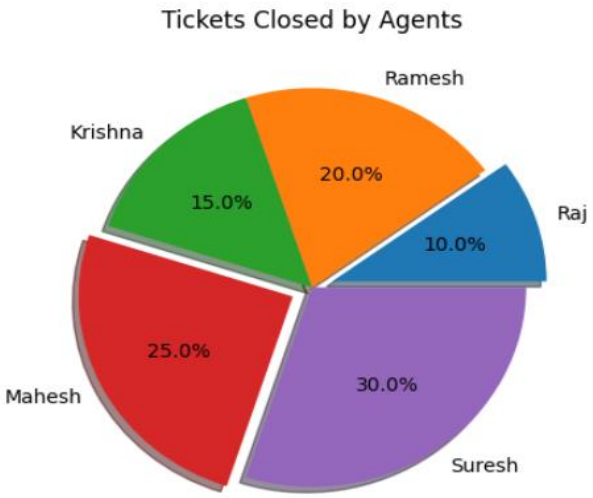
Customizations

- **labels** – Specifies labels for each pie slice.
- **explode** – Separates one or more slices from the pie chart.
- **autopct** – Displays percentage values on the slices.
- **shadow** – Adds a shadow effect to the pie chart.
- **colors** – Sets custom colors for the pie slices.
- **startangle** – Rotates the starting position of the pie chart by a specified angle.

Example

```
import matplotlib.pyplot as plt

tickets_closed = [10,20,15,25,30]
agents =
['Raj','Ramesh','Krishna','Mahesh','Suresh']

explode = [0.1,0,0,0.1,0]

plt.pie(tickets_closed,
labels=agents,explode=explode,
autopct='%1.1f%%',shadow=True)

plt.title("Tickets Closed by Agents")

plt.show()
```

Output:



7. Donut Chart

A Donut Chart is a modified Pie Chart with a hole at the center. It displays proportional data while providing additional space for labels.

Function

- Uses plt.pie() with a center circle.

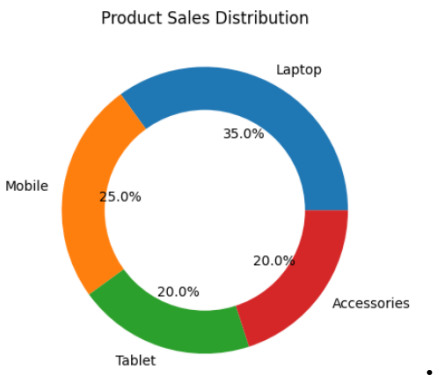
Customizations

- **labels** – Specifies labels for each pie slice.
- **explode** – Separates slices from the center of the pie chart.
- **autopct** – Displays percentage values on the slices.
- **colors** – Sets custom colors for the pie slices.
- **shadow** – Adds a shadow effect to the pie chart.

Example

```
import matplotlib.pyplot as plt
sales = [35,25,20,20]
products =
['Laptop','Mobile','Tablet','Accessories']

plt.pie(sales, labels=products,
autopct='%1.1f%%')

centre_circle = plt.Circle((0,0), 0.70,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.title("Product Sales Distribution")

plt.show()
```

Output



:

8. Bubble Plot

A Bubble Plot is an extension of a Scatter Plot where the size of each point represents an additional variable.

Function

- The function used for Bubble Plot is plt.scatter() with size parameter s.

Customizations

- **s** – Specifies the size of the bubbles.
- **color** – Sets the color of the bubbles.
- **alpha** – Controls the transparency of the bubbles.
- **marker** – Specifies the shape of the bubbles.

Example

```
import matplotlib.pyplot as plt

experience = [1,2,3,4,5]

salary = [25000,30000,40000,50000,65000]

employees = [50,80,120,150,200]

plt.scatter(experience, salary,s=employees,

alpha=0.5)

plt.title("Experience vs Salary")

plt.xlabel("Experience (Years)")

plt.ylabel("Salary")

plt.show()
```

Output:

Lab Assignment

SET A

1. Generate a dataset representing the daily temperatures (in °C) recorded over 50 days. Visualize the dataset using a Line Chart and Scatter Plot. Apply appropriate colors, titles, axis labels, and styling options. [Attributes for dataset – (Date, Temperature (°C))]
2. Consider the following dataset representing the number of books available in different sections of a library:

   sections = ["Science", "Literature", "History", "Technology", "Commerce"]

   books = [450, 380, 250, 520, 310]

   Create a **Bar Chart** to visualize the number of books available in each section. Use different bar colors, add an appropriate title, and label both axes.
3. Generate a suitable dataset containing the ages of 100 gym members. Create a Histogram to visualize the age distribution of the members. Customize the graph using appropriate bins, colors, title, and labels. [Attributes for dataset – Member ID, Age]
4. A company conducted a survey to determine the preferred mode of transportation among employees as shown below:

   transport = ["Bus", "Train", "Car", "Bike", "Bicycle"], employees = [120, 80, 60, 90, 30]

   Create a Pie Chart to represent the percentage of employees using each mode of transportation. Display the percentage contribution of each category.

SET B

1. The environmental department collected the following information for six cities:

   cities = ["A", "B", "C", "D", "E", "F"],

   population = [5, 8, 12, 15, 20, 25]

   pollution = [40, 48, 55, 63, 72, 85],

   green_cover = [350, 300, 250, 220, 180, 150]

   Create a Bar Chart showing pollution levels and a Bubble Plot showing the relationship among population, pollution, and green cover.
2. Load the **Titanic dataset (titanic.csv)**

   Create a histogram showing the number of passengers in each passenger class.

   Create a Pie Chart showing the survival distribution.
3. Write a Python program to draw scatter plots to compare two features of the iris dataset.

SET C

1. Banking Loan Analysis

Generate a dataset containing loan information.

Attributes: Customer Age, Loan Amount, Annual Income, Credit Score

Perform the following tasks:

a) Create Histograms for Loan Amount and Credit Score.

b) Generate Scatter Plots for Income vs Loan Amount.

c) Create Bubble Plots using Income, Credit Score, and Loan Amount.

2. Smart City Traffic Analysis

Generate a dataset showing monthly traffic statistics.

Attributes: Month, Cars, Two-Wheelers, Public Transport Users

Perform the following tasks:

a) Create Line Charts showing traffic trends.

b) Generate Area Plots for cumulative traffic volume.

c) Create Pie Charts showing transportation mode share.

d) Analyze urban mobility patterns and congestion trends.

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion: __/___/____**

Assignment 3

Advanced Data Visualization tools

Number of Slots = 2

Objectives:

- To understand and apply advanced data visualization techniques using Python.
- To create and interpret Box Plots, Heat Maps, Dendrograms, Venn Diagrams, and Treemaps.
- To visualize multidimensional, textual, and geographical data using 3D Scatter Plots, Word Clouds, and Geospatial Visualizations.
- To analyze complex datasets and extract meaningful insights through advanced visual representations.

Introduction:

Data visualization helps in understanding and interpreting data through graphical representations. In the previous assignment, basic visualization techniques such as Histograms, Bar Charts, Scatter Plots, Line Charts, Area Plots, Pie Charts, Donut Charts, and Bubble Plots were studied.

However, real-world datasets are often complex and require advanced visualization techniques to uncover deeper insights. Advanced visualizations help in analyzing data distributions, correlations, clusters, hierarchical relationships, text patterns, and geographical information.

In this assignment, we will learn advanced visualization tools including Box Plots, Heat Maps, Dendrograms, Venn Diagrams, Treemaps, 3D Scatter Plots, Word Clouds, and Geospatial Data Visualizations.

Visualization Tools

Visualization Tools	**Purpose**
Box Plot	Detect Outliers and Distribution
Heat Map	Analyze Correlations
Dendrogram	Visualize Hierarchical Clustering
Venn Diagram	Show Set Relationships
Treemap	Display Hierarchical Data
3D Scatter Plot	Analyze Three Variables
Word Cloud	Visualize Text Data
Geospatial Visualization	Visualize Location-Based Data

1. Box Plot

A Box Plot (also known as a Box-and-Whisker Plot) is a statistical visualization used to represent the distribution of numerical data. It provides a summary of the dataset using five important statistical measures: Minimum Value, First Quartile (Q1), Median (Q2), Third Quartile (Q3), and Maximum Value**.**

Box plots are useful for understanding the spread of data, identifying skewness, comparing distributions, and detecting outliers. They are widely used in data analysis and exploratory data analysis (EDA).

Outliers

Outliers are data points that significantly differ from the rest of the dataset. In a Box Plot, outliers are displayed as individual points outside the whiskers.

Five-Number Summary

A Box Plot is based on the following five values:

- **Minimum Value** – Smallest value in the dataset.
- **First Quartile (Q1)** – 25% of data lies below this value.
- **Median (Q2)** – Middle value of the dataset.
- **Third Quartile (Q3)** – 75% of data lies below this value.
- **Maximum Value** – Largest value in the dataset.

Function

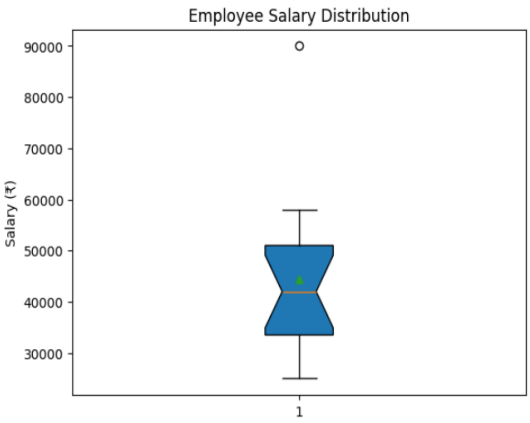
- The function used to create a Box Plot is **plt.boxplot()**.
- It accepts a list of numerical values or multiple datasets.
- The graph displays quartiles, median, whiskers, and outliers.

Customizations

The plt.boxplot() function supports the following arguments:

- **labels** – Labels for multiple datasets.
- **vert** – Displays vertical or horizontal box plot.
- **patch_artist** – Enables box coloring.
- **showmeans** – Displays the mean value.
- **showfliers** – Displays outliers.
- **notch** – Displays a notch around the median.
- **widths** – Controls box width.

Example

import matplotlib.pyplot as plt salary = [25000, 28000, 30000, 32000, 35000,37000, 40000, 42000, 45000, 48000, 50000, 52000, 55000, 58000, 90000] plt.boxplot(salary, patch_artist=True, showmeans=True, notch=True) plt.title("Employee Salary Distribution") plt.ylabel("Salary (₹)") plt.show()	Output: 

2. Heat Map

A **Heat Map** is a graphical representation of data in which values are represented using different colors. It helps visualize patterns, variations, and relationships within a dataset. Higher values are usually represented by darker or warmer colors, while lower values are represented by lighter or cooler colors.

Heat Maps are commonly used for correlation analysis, data comparison, and identifying trends in large datasets. They provide an easy way to understand complex numerical information through color intensity.

Function

Heat Maps are commonly created using the **Seaborn** library.

- The function used to create a Heat Map is **sns.heatmap()**.
- It accepts a matrix or DataFrame as input.
- The graph displays values using color gradients.

Customizations

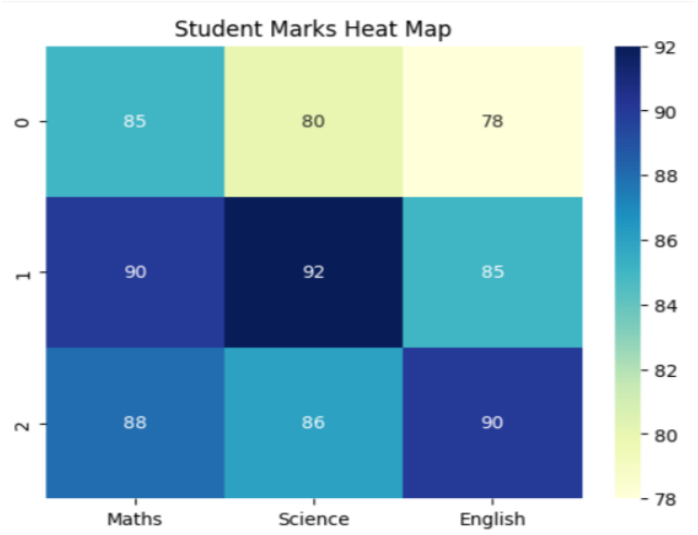
The sns.heatmap() function supports the following arguments:

- **annot** – Displays numerical values inside cells.
- **cmap** – Specifies the color map.
- **linewidths** – Sets the width of cell borders.
- **linecolor** – Sets border color.
- **fmt** – Formats displayed values.
- **cbar** – Displays or hides the color bar.
- **square** – Makes cells square-shaped.

Example

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

data = {
   'Maths': [85, 90, 88],
   'Science': [80, 92, 86],
   'English': [78, 85, 90]
}

df = pd.DataFrame(data)

sns.heatmap(df,
            annot=True,
            cmap='YlGnBu')

plt.title("Student Marks Heat Map")
plt.show()
```

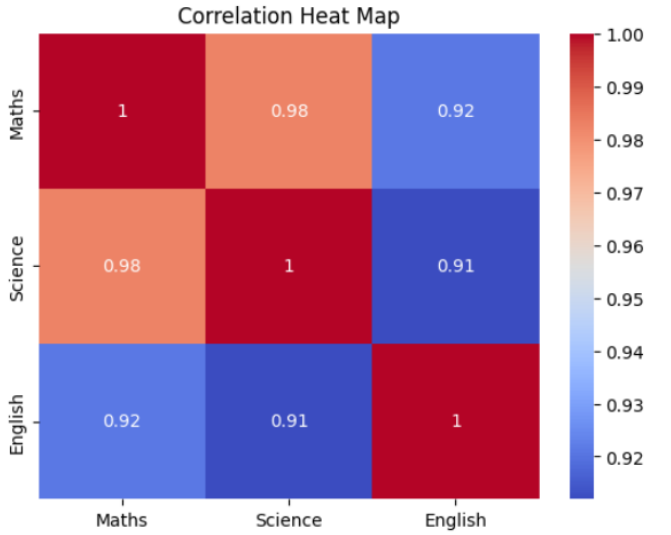
Output:



Example with Correlation Matrix -

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

data = {
   'Maths': [85, 90, 88, 75, 95],
   'Science': [80, 92, 86, 70, 98],
   'English': [78, 85, 90, 72, 93]
}

df = pd.DataFrame(data)

corr_matrix = df.corr()

sns.heatmap(corr_matrix,
            annot=True,
            cmap='coolwarm')

plt.title("Correlation Heat Map")
plt.show()
```

Output:



3. Dendrogram

A Dendrogram is a tree-like diagram used to represent hierarchical relationships among data points. It is commonly used in Hierarchical Clustering to show how similar objects are grouped together.

A Dendrogram helps visualize the arrangement of clusters produced by clustering algorithms. The closer two data points are merged in the diagram, the more similar they are. It is widely used in data mining, machine learning, bioinformatics, and pattern recognition.

Function

Dendrograms are created using the **SciPy** library.

- The function used to create a Dendrogram is **dendrogram()**.
- Hierarchical clustering is performed using the **linkage()** function.
- The graph displays the hierarchical relationship among data points.

Customizations

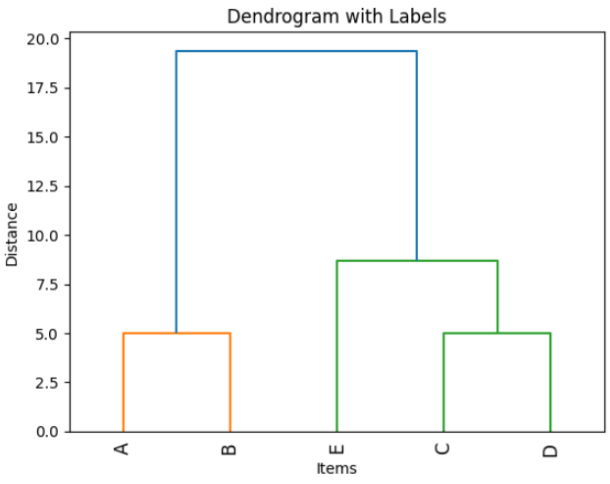
The dendrogram() function supports the following arguments:

- **labels** – Labels for data points.
- **orientation** – Controls the direction of the tree.
- **leaf_rotation** – Rotates labels.
- **leaf_font_size** – Sets label font size.
- **color_threshold** – Changes cluster colors.
- **show_leaf_counts** – Displays the number of items in each cluster.

Example

```
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import
dendrogram, linkage

data = [[5], [10], [15], [20], [25]]

labels = ['A', 'B', 'C', 'D', 'E']

linked = linkage(data, method='ward')

dendrogram(linked, labels=labels,
leaf_rotation=90)

plt.title("Dendrogram with Labels")
plt.xlabel("Items")
plt.ylabel("Distance")
plt.show()
```

Output:



4. Venn Diagram

A Venn Diagram is a graphical representation used to show the relationships between two or more sets. It consists of overlapping circles, where each circle represents a set and the overlapping regions represent common elements shared between the sets.

Venn Diagrams are widely used in mathematics, statistics, probability, logic, and data analysis to visualize similarities, differences, and intersections among groups of data.

Function

Venn Diagrams can be created using the **matplotlib-venn** library.

- The function used to create a two-set Venn Diagram is **venn2()**.
- The function used to create a three-set Venn Diagram is **venn3()**.
- The graph displays common and unique elements among sets.

Customizations

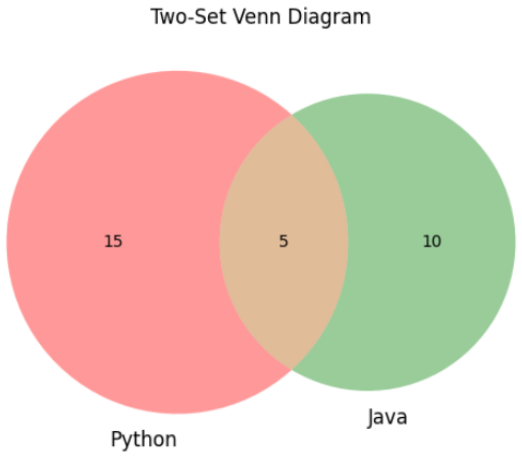
The venn2() and venn3() functions support the following arguments:

- **subsets** – Specifies the size of each region.
- **set_labels** – Labels for the sets.
- **set_colors** – Colors of the circles.
- **alpha** – Controls transparency.
- **normalize_to** – Adjusts diagram scaling.

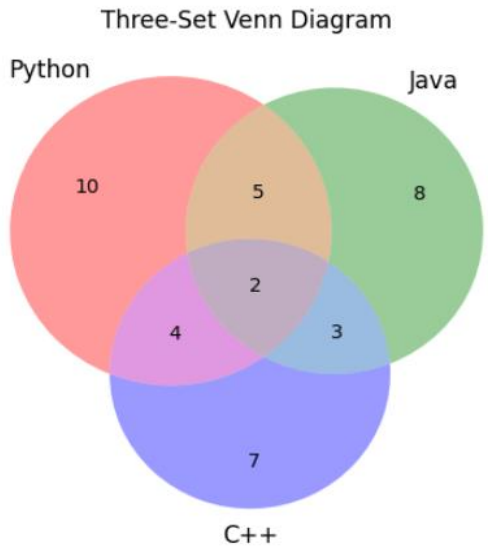
Example 1: Two-Set Venn Diagram

```
from matplotlib_venn import venn2
import matplotlib.pyplot as plt

venn2(subsets=(15, 10, 5),
      set_labels=('Python', 'Java'))

plt.title("Two-Set Venn Diagram")

plt.show()
```

Output:



Example 2: Three-Set Venn Diagram

from matplotlib_venn import venn3 import matplotlib.pyplot as plt venn3(subsets=(10, 8, 5, 7, 4, 3, 2), set_labels=('Python', 'Java', 'C++')) plt.title("Three-Set Venn Diagram") plt.show()	Output: 

5. Treemap

A Treemap is a visualization technique used to display hierarchical data using nested rectangles. Each rectangle represents a category, and its size is proportional to the value it represents. Larger values are displayed as larger rectangles, while smaller values are displayed as smaller rectangles. Treemaps are useful when dealing with hierarchical datasets and when comparing the relative proportions of different categories within a dataset. They provide an efficient way to visualize large amounts of information in a limited space.

Function

Treemaps can be created using the **Squarify** library.

- The function used to create a Treemap is **squarify.plot()**.
- It accepts numerical values representing the size of each category.
- The graph displays categories as rectangles of varying sizes.

Customizations

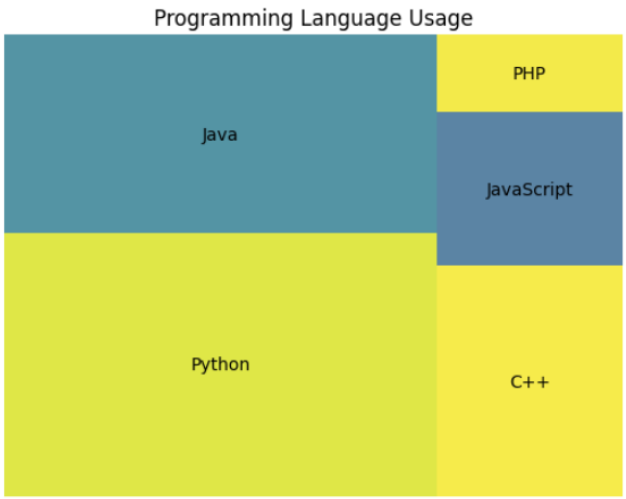
The squarify.plot() function supports the following arguments:

- **sizes** – Specifies the size of each rectangle.
- **label** – Labels for categories.
- **color** – Sets rectangle colors.
- **alpha** – Controls transparency.
- **pad** – Adds spacing between rectangles.
- **text_kwargs** – Customizes label appearance.

Example

```
import matplotlib.pyplot as plt
import squarify

sizes = [40, 30, 15, 10, 5]
labels = ['Python', 'Java', 'C++',
'JavaScript', 'PHP']

squarify.plot(sizes=sizes,
              label=labels,
              alpha=0.8)

plt.title("Programming Language Usage")

plt.axis("off")
plt.show()
```

Output:



6. 3D Scatter Plot

A **3D Scatter Plot** is an extension of a traditional Scatter Plot that displays data points in three dimensions using the **X-axis, Y-axis, and Z-axis**. It is used to visualize the relationship among three numerical variables simultaneously.

Unlike a 2D Scatter Plot, which represents only two variables, a 3D Scatter Plot allows the analysis of multidimensional data and helps identify patterns, trends, clusters, and relationships among three different attributes.

Function

3D Scatter Plots are created using the **Matplotlib mplot3d** toolkit.

- The function used to create a 3D Scatter Plot is **ax.scatter()**.
- It requires three sets of values representing X, Y, and Z coordinates.
- The graph displays data points in a three-dimensional space.

Customizations

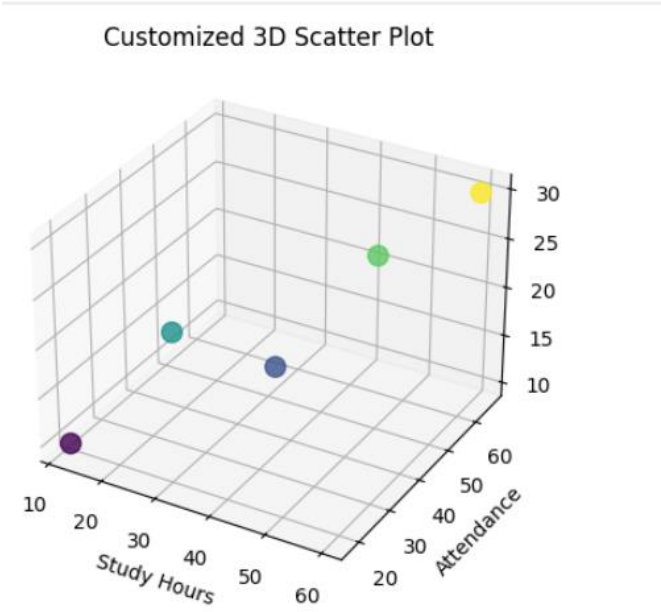
The ax.scatter() function supports the following arguments:

- **c** – Sets point colors.
- **marker** – Specifies marker style.
- **s** – Controls marker size.
- **alpha** – Controls transparency.
- **label** – Adds labels for data points.
- **cmap** – Applies color mapping based on values.

Example

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d
import Axes3D

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

x = [12, 24, 36, 48, 60]
y = [18, 30, 42, 54, 66]
z = [10, 20, 15, 25, 30]

ax.scatter(x, y, z,
        c=z,
        s=100,
        marker='o',
        alpha=0.8)

ax.set_title("Customized 3D Scatter Plot")
ax.set_xlabel("Study Hours")
ax.set_ylabel("Attendance")
ax.set_zlabel("Marks")

plt.show()
```

Output:



7. Word Cloud

A **Word Cloud** is a visual representation of textual data in which words are displayed with different sizes based on their frequency or importance. Words that appear more frequently in the text are displayed in a larger font size, while less frequent words appear smaller.

Word Clouds help summarize large amounts of text data and quickly identify the most important keywords. They are widely used in text analytics, social media analysis, customer feedback analysis, and natural language processing (NLP).

Function

Word Clouds are created using the **WordCloud** library.

- The function used to create a Word Cloud is **WordCloud()**.
- The **generate()** method is used to generate the Word Cloud from text data.
- Words are displayed according to their frequency in the text.

Customizations

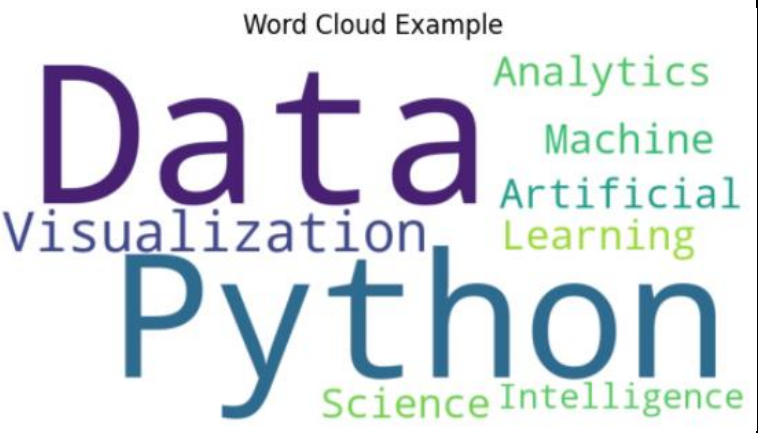
The WordCloud() function supports the following arguments:

- **width** – Sets the width of the image.
- **height** – Sets the height of the image.
- **background_color** – Sets the background color.
- **max_words** – Limits the number of displayed words.
- **colormap** – Sets the color scheme.
- **stopwords** – Removes common words.
- **min_font_size** – Sets the minimum font size.

Example

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

text = """
Python Python Python Data Data
Data
Visualization Machine Learning
Analytics
Data Science Python Visualization
Artificial Intelligence Data
"""

wc = WordCloud(width=800,
               height=400,
               background_color='white',
               max_words=50)

wordcloud = wc.generate(text)

plt.imshow(wordcloud)
plt.axis("off")
plt.title("Word Cloud Example")

plt.show()
```

Output:



8. Geospatial Data Visualization

Geospatial Data Visualization is the process of representing geographical or location-based data on maps. It helps users understand spatial patterns, distributions, and relationships between different locations.

Geospatial visualizations use coordinates such as Latitude and Longitude to plot locations on a map. These visualizations are widely used in transportation, weather forecasting, urban planning, business analytics, GPS systems, and geographical information systems (GIS).

Function

Geospatial visualizations can be created using libraries such as Plotly, GeoPandas, and Folium.

In this task, we will use Plotly Express to display locations on a map.

- The function used is px.scatter_geo().
- It plots geographical locations using latitude and longitude coordinates.
- Each point on the map represents a location.

Customizations

The px.scatter_geo() function supports the following arguments:

- **lat** – Latitude values.
- **lon** – Longitude values.
- **hover_name** – Displays location names.
- **title** – Adds a title to the map.
- **color** – Assigns colors based on categories.
- **size** – Controls marker size.
- **projection** – Sets map projection style.

Example 1: Plotting Cities on a Map

```
import pandas as pd
import plotly.express as px

data = {
   'City': ['Pune', 'Mumbai', 'Nagpur', 'Nashik'],
   'Latitude': [18.5204, 19.0760, 21.1458, 19.9975],
   'Longitude': [73.8567, 72.8777, 79.0882,73.7898]
}
df = pd.DataFrame(data)
fig = px.scatter_geo(df,
                     lat='Latitude',
                     lon='Longitude',
                     hover_name='City',
                     title='Cities in Maharashtra')
fig.show()
```

Output:

Example 2: Plotting Cities with Different Marker Sizes

```
import pandas as pd
import plotly.express as px
data = {
   'City': ['Pune', 'Mumbai', 'Nagpur',
'Nashik'],
   'Population': [7, 20, 3, 2],
   'Latitude': [18.5204, 19.0760, 21.1458,
19.9975],
   'Longitude': [73.8567, 72.8777, 79.0882,
73.7898]
}
df = pd.DataFrame(data)
fig = px.scatter_geo(df,
                     lat='Latitude',
                     lon='Longitude',
                     size='Population',
                     hover_name='City',
                     title='City Population
Visualization')
fig.show()
```

Output:

Lab Assignment

SET A

1. A retail store has recorded the daily sales (₹) for two months. Create a suitable dataset and generate a Box Plot to identify any outliers. Customize the graph with appropriate title and labels.
2. Write a Python program to generate a box plot to show the Interquartile range and outliers for the three species for each feature using IRIS data set.
3. Create a dataset containing monthly values of Temperature, Humidity, Rainfall, and Wind Speed. Generate a Heat Map to visualize the correlation between these weather parameters.
4. Create a dataset containing information about different products such as Price, Rating, and Number of Sales. Generate a Dendrogram to group similar products based on their characteristics.
5. A survey was conducted to identify customers who use Netflix and Amazon Prime. Create a Venn Diagram showing - Customers using only Netflix, Customers using only Amazon Prime,Customers using both services

Set B

1. Download the Auto MPG Dataset (mpg.csv)[MPG, Horsepower, Weight, Cylinders, Acceleration.] and perform the following tasks:
   a. Create Box Plots for MPG and Horsepower, Identify outliers
   b. Generate a Heat Map showing correlations among numerical attributes.

2. Create a dataset containing movie titles available on Netflix, Amazon Prime, and Disney+ and perform the following tasks:
   a. Create a Venn Diagram comparing Netflix and Amazon Prime movie collections.
   b. Generate a Word Cloud using movie genres or titles.
3. Create a dataset containing information about different cities.
   Attributes: City, Population, GDP, Literacy Rate
   Perform the following tasks:
   a. Generate a **Treemap** showing city-wise population distribution.
   b. Create a **3D Scatter Plot** using Population, GDP, and Literacy Rate.
4. Download a publicly available **Earthquake Dataset (CSV format)** containing latitude, longitude, and magnitude information.
   Perform the following tasks:
   a. Create a **Geospatial Visualization** showing earthquake locations on a map.
   b. Use marker size or color to represent earthquake magnitude.

SET C

1. Download climate data for multiple countries.
   Attributes: Country, Average Temperature, CO_2 Emissions, Rainfall, Forest Area
   Perform the following tasks:
   a. Generate a Heat Map showing feature correlations.
   b. Create a 3D Scatter Plot using Temperature, CO_2 Emissions, and Forest Area.
   c. Create a Geospatial Visualization displaying country-wise climate indicators.
2. Create a dataset containing information about different colleges.
   Attributes: College Name, Placement %, Research Publications, Student Strength, Average Package
   Perform the following tasks:
   a. Create a Treemap showing student distribution.
   b. Generate a 3D Scatter Plot using Placement %, Publications, and Average Package.
   c. Create a Heat Map for feature correlation.

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion: __/___/____**

Assignment 4

Supervised Machine Learning Models for Regression

(Linear Regression, Polynomial Regression, Logistic Regression)

No. of slots: 03

Objectives

- Analyze data using appropriate tools and techniques to identify useful insights and support decision-making.
- Apply linear, polynomial, and logistic regression models to analyze data and solve real-world business problems.

Machine Learning:

Machine learning enables a machine to automatically learn from data, improve performance from experiences, and predict things without being explicitly programmed. With the help of sample historical data, which is known as training data, machine learning algorithms build a mathematical model that helps in making predictions or decisions without being explicitly programmed. Machine learning brings computer science and statistics together for creating predictive models.

Machine learning can be classified into three types:

1) **Supervised Learning -** Supervised learning is a type of machine learning method in which we provide sample labeled data to the machine learning system in order to train it, and on that basis, it predicts the output.
2) **Unsupervised Learning -** Unsupervised learning is a learning method in which a machine learns without any supervision.
3) **Reinforcement Learning -** Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action. The agent learns automatically with these feedbacks and improves its performance. In reinforcement learning, the agent interacts with the environment and explores it. The goal of an agent is to get the most reward points, and hence, it improves its performance.

Regression Analysis:

Regression Analysis is a statistical and machine learning technique used to predict continuous numerical values by modeling the relationship between independent variables (inputs) and a dependent variable (output). Its goal is to find the best-fitting line or curve that describes the data and helps estimate how changes in input variables affect the output. In machine learning, regression is a supervised learning method widely used for forecasting and predictive analytics. The model learns from input-output data during training and predicts continuous values for new inputs.

Types of Regression(Frequently Used)

1. **Linear Regression**

   Predicts continuous numerical values by modeling a straight-line relationship between independent and dependent variables.

   Application: House price prediction, sales forecasting, and salary estimation.

   Types of Linear Regression

   a. **Simple Linear Regression**: Uses one independent variable to predict a dependent variable.

      Example: Predicting salary based on years of experience.

   b. **Multiple Linear Regression**: Uses two or more independent variables.

      Example: Predicting house prices based on area, location, and number of bedrooms.

2. **Logistic Regression**

   Used for classification problems by predicting the probability of an outcome belonging to a particular class.

   Application: Email spam detection, disease diagnosis, customer churn prediction, and fraud detection.

   Types of Logistic regression

   a. **Binary Logistic Regression**: Predicts one of two possible outcomes.

      Example: Spam or Not Spam.

   b. **Multinomial Logistic Regression**: Predicts one of more than two categories.

      Example: Classifying fruits as Apple, Banana, or Orange.

   c. **Ordinal Logistic Regression**: Predicts ordered categories.

      Example: Customer ratings (Poor, Average, Good, Excellent)

3. **Polynomial Regression**

   Extends linear regression by fitting a curved line to capture non-linear relationships between variables.

   Application: Stock trend analysis, population growth prediction, and weather forecasting.

   Types of Polynomial regression

   a. **Quadratic Regression (Degree 2)**: Includes x^2terms.

      Example: Modeling the relationship between speed and fuel efficiency.

   b. **Cubic Regression (Degree 3)**: Includes x^3terms.

      Example: Population growth trends.

c. **Higher-Degree Polynomial Regression:** Uses higher powers (x^4, x^5, etc.) for complex patterns.

Example: Scientific and engineering data modeling

Steps in Regression Analysis

1. **Define the Problem**

   Identify the dependent variable (target) and independent variables (features).

2. **Collect Data**

   Gather relevant and sufficient data for analysis.

3. **Preprocess the Data**

   Handle missing values, remove outliers, and prepare the dataset.

4. **Split the Dataset**

   Divide the data into training and testing sets.

5. **Choose a Regression Model**

   Select an appropriate model such as Linear, Polynomial, or Logistic Regression.

6. **Train the Model**

   Fit the regression model using the training data.

7. **Evaluate the Model**

   Measure performance using metrics such as R^2 Score, MAE, MSE, or RMSE.

8. **Make Predictions**

   Use the trained model to predict outcomes for new data.

9. **Visualization**

   Analyze the relationship between variables and draw conclusions.

Sample code for Linear regression -

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

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,
r2_score

# Features: Age, Experience
age = np.random.randint(22, 60, 100)
experience = np.random.randint(0, 35, 100)

# Salary formula (realistic pattern + noise)
salary = (30000+ (age * 1500)+ (experience * 3000)
```

Output

Simple Linear Regression(Age → Salary)

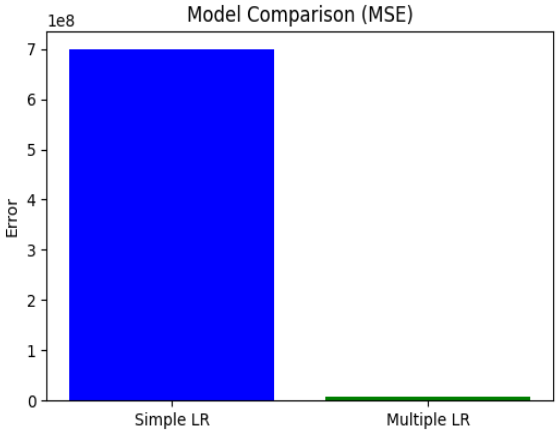
```
MSE: 699299864.1029439
R2 Score: 0.39601414560695547
Predicted Salary for Age 35:
134058.4645056139
```

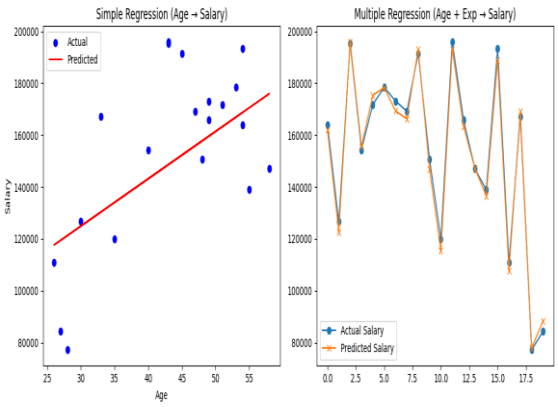
Multiple Linear Regression

```
MSE: 8393289.211130708
R2 Score: 0.9927507093657801
```

```
+ np.random.randint(-5000, 5000, 100))

# Reshape features
X_simple = age.reshape(-1, 1)   # only Age
X_multi = np.column_stack((age, experience))
y = salary

# Simple Linear Regression (Age → Salary)

X_train_s, X_test_s, y_train_s, y_test_s =
train_test_split(X_simple, y, test_size=0.2,
random_state=42)

simple_model = LinearRegression()
simple_model.fit(X_train_s, y_train_s)

y_pred_s = simple_model.predict(X_test_s)

print("\n Simple Linear Regression(Age →
Salary)\n")
print("MSE:", mean_squared_error(y_test_s,
y_pred_s))
print("R2 Score:", r2_score(y_test_s, y_pred_s))

# New prediction
new_age = np.array([[35]])
print("Predicted Salary for Age 35:",
simple_model.predict(new_age)[0])

# Multiple Linear Regression (Age + Experience
→ Salary)

X_train_m, X_test_m, y_train_m, y_test_m =
train_test_split(X_multi, y, test_size=0.2,
random_state=42)

multi_model = LinearRegression()
multi_model.fit(X_train_m, y_train_m)

y_pred_m = multi_model.predict(X_test_m)

print("\n Multiple Linear Regression\n ")
print("MSE:", mean_squared_error(y_test_m,
y_pred_m))
print("R2 Score:", r2_score(y_test_m, y_pred_m))

# New prediction (Age, Experience)
new_input = np.array([[35, 10]])
print("Predicted Salary (Age 35, Exp 10):",
multi_model.predict(new_input)[0])

#Comparison Graph (MSE)

models = ["Simple LR", "Multiple LR"]
mse_values = [mean_squared_error(y_test_s,
```

```
Predicted Salary (Age 35, Exp 10):
112343.36592824249
```





```
y_pred_s), mean_squared_error(y_test_m,
y_pred_m)]

plt.figure(figsize=(6,4))
plt.bar(models, mse_values, color=["blue", "green"])
plt.title("Model Comparison (MSE)")
plt.ylabel("Error")
plt.show()

# Visualization

plt.figure(figsize=(12,5))

# Simple Regression (Age vs Salary)
plt.subplot(1,2,1)
plt.scatter(X_test_s, y_test_s, color="blue",
label="Actual")
plt.plot(X_test_s, y_pred_s, color="red",
label="Predicted")
plt.title("Simple Regression (Age → Salary)")
plt.xlabel("Age")
plt.ylabel("Salary")
plt.legend()

#Multiple Regression
plt.subplot(1,2,2)
plt.plot(y_test_m, marker="o", label="Actual
Salary")
plt.plot(y_pred_m, marker="x", label="Predicted
Salary")
plt.title("Multiple Regression (Age + Exp →
Salary)")
plt.legend()

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

Sample code for Logistic regression –

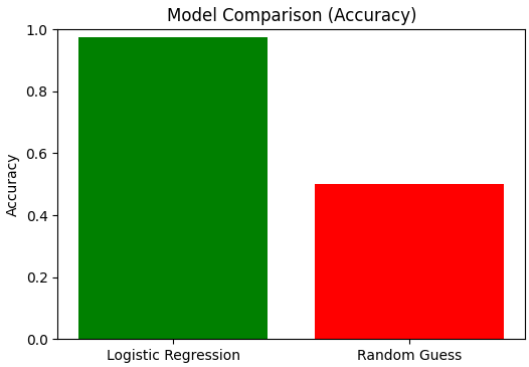
Predicts product purchase using age and salary with Logistic Regression

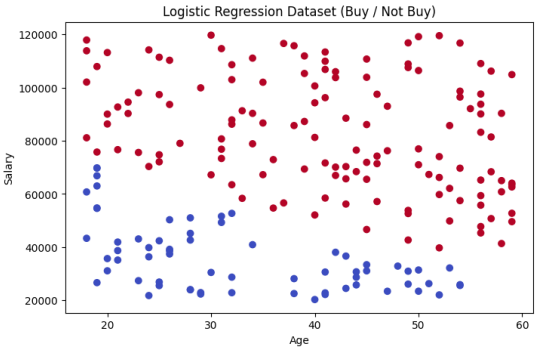
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,
confusion_matrix, classification_report

#Create Dataset
np.random.seed(42)
age = np.random.randint(18, 60, 200)
salary = np.random.randint(20000, 120000, 200)
X = np.column_stack((age, salary))
```

Output

```
Logistic Regression Results
Accuracy: 0.975

Confusion Matrix:
 [[11  1]
 [ 0 28]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.92      0.96        12
           1       0.97      1.00      0.98        28

    accuracy                           0.97        40
   macro avg       0.98      0.96      0.97        40
weighted avg       0.98      0.97      0.97        40

Prediction for Age=35, Salary=60000: 1
```

```
y = ((age * 1000 + salary) > 90000).astype(int)

#Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

#Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#Model Training
model = LogisticRegression()
model.fit(X_train, y_train)

#Prediction
y_pred = model.predict(X_test)

#Evaluation
accuracy = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("\nLogistic Regression Results")
print("Accuracy:", accuracy)
print("\nConfusion Matrix:\n", cm)
print("\nClassification Report:\n", report)

#New Prediction
new_data = np.array([[35, 60000]])
new_data_scaled = scaler.transform(new_data)
prediction = model.predict(new_data_scaled)
print("Prediction for Age=35, Salary=60000:",
prediction[0])

#Comparison Graph
models = ["Logistic Regression", "Random Guess"]
accuracy_values = [accuracy, 0.5]
plt.figure(figsize=(6,4))
plt.bar(models, accuracy_values, color=["green",
"red"])
plt.title("Model Comparison (Accuracy)")
plt.ylabel("Accuracy")
plt.ylim(0, 1)
plt.show()

#Visualization
plt.figure(figsize=(8,5))
plt.scatter(age, salary, c=y, cmap="coolwarm")
plt.title("Logistic Regression Dataset (Buy / Not Buy)")
plt.xlabel("Age")
plt.ylabel("Salary")
plt.show()
```



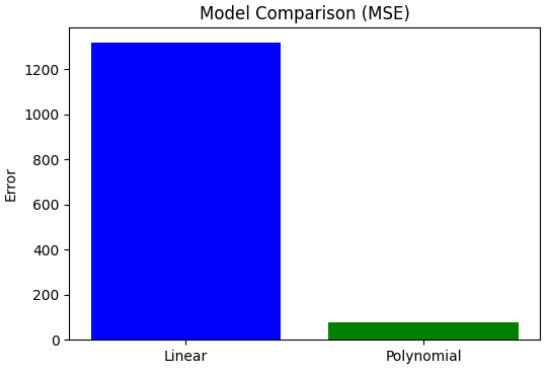


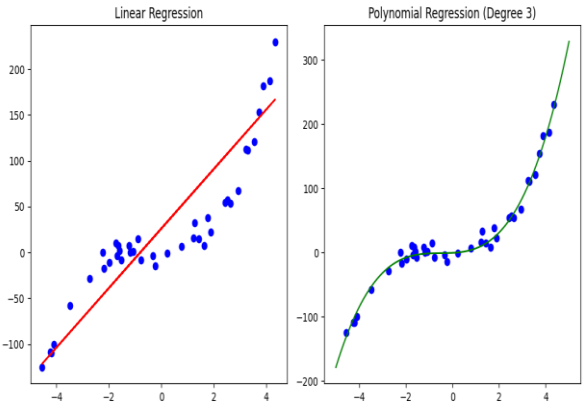
Sample code for polynomial regression –

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import
PolynomialFeatures
from sklearn.metrics import mean_squared_error,
r2_score

#Create Non-Linear Dataset
np.random.seed(42)
X = np.linspace(-5, 5, 200).reshape(-1, 1)
y = 2 * (X**3) + 3 * (X**2) + X +
np.random.randn(200, 1) * 10

#Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

#Linear Regression Model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
y_pred_linear = linear_model.predict(X_test)

#Polynomial Regression Model
poly = PolynomialFeatures(degree=3)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

poly_model = LinearRegression()
poly_model.fit(X_train_poly, y_train)
y_pred_poly = poly_model.predict(X_test_poly)

# Evaluation
print("\nLinear Regression Results")
print("MSE:", mean_squared_error(y_test,
y_pred_linear))
print("R2 Score:", r2_score(y_test, y_pred_linear))

print("\nPolynomial Regression Results")
print("MSE:", mean_squared_error(y_test,
y_pred_poly))
print("R2 Score:", r2_score(y_test, y_pred_poly))

# New Prediction
new_value = np.array([[2]])
print("\nPrediction (Linear):",
linear_model.predict(new_value)[0])
print("Prediction (Polynomial):",
poly_model.predict(poly.transform(new_value))[0])

#Comparison Graph (Mse)
models = ["Linear", "Polynomial"]
```

Output

```
Linear Regression Results
MSE: 1318.458719003272
R2 Score: 0.7762947760293312

Polynomial Regression Results
MSE: 78.27726776351318
R2 Score: 0.9867185574607245

Prediction (Linear): [90.88775015]
Prediction (Polynomial): [30.77541048]
```





```
mse_values = [mean_squared_error(y_test,
y_pred_linear), mean_squared_error(y_test,
y_pred_poly)]

plt.figure(figsize=(6,4))
plt.bar(models, mse_values, color=["blue", "green"])
plt.title("Model Comparison (MSE)")
plt.ylabel("Error")
plt.show()

#Visualization
plt.figure(figsize=(10,5))

# Linear plot
plt.subplot(1,2,1)
plt.scatter(X_test, y_test, color="blue")
plt.plot(X_test, y_pred_linear, color="red")
plt.title("Linear Regression")

# Polynomial curve
plt.subplot(1,2,2)
X_curve = np.linspace(-5, 5, 200).reshape(-1, 1)
X_curve_poly = poly.transform(X_curve)
y_curve = poly_model.predict(X_curve_poly)

plt.scatter(X_test, y_test, color="blue")
plt.plot(X_curve, y_curve, color="green")
plt.title("Polynomial Regression (Degree 3)")

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

Lab Assignment

SET A

1. Apply Linear Regression on a YouTube Video dataset to predict video views using relevant video attributes. Visualize the relationship between the predictor and target variable using a scatter plot and regression line.
2. Apply Logistic Regression on an Online Gaming dataset to classify players as Casual or Professional gamers based on features such as daily play time, achievements unlocked, in-game purchases, and gaming level. Evaluate the model using Accuracy, Precision, Recall, F1-Score, and Confusion Matrix.
3. Download the Salary Dataset. Write a Python program to read the dataset and display its information. Preprocess the data if required and split it into training and testing sets. Apply Simple Linear Regression to predict salary based on years of experience. Plot the regression line and evaluate the model using appropriate metrics such as Mean Squared Error.

4. Apply Polynomial Regression to predict song popularity using audio features such as danceability, energy, and tempo. Evaluate the model using MAE, RMSE, and R^2 Score, and visualize the fitted curve.

SET B

1. Apply Logistic Regression to classify smartphones into different price categories using specifications such as RAM, storage, battery capacity, and camera resolution. Assess the model using Accuracy, F1-Score, and Confusion Matrix.
2. Use a Gaming Dataset containing gameplay hours, achievements unlocked, in-game purchases, and friend count. Develop a Multiple Linear Regression model to predict player engagement score. Evaluate the model using MAE, MSE, RMSE, and R^2 Score.
3. Use a Sports Performance dataset containing attributes such as practice hours, fitness score, match experience, and training sessions. Apply Simple Linear Regression and Multiple Linear Regression to predict player performance. Compare the models using MAE, MSE, RMSE, and R^2 Score, and identify the most suitable regression model.

SET C

1. Download a Mobile Phone Specifications dataset. Apply Multiple Linear Regression to predict mobile prices using features such as RAM, storage, battery capacity, and camera resolution. Categorize mobiles into Budget and Premium segments and build a Logistic Regression model for classification. Compare the performance of both models using appropriate regression and classification metrics.
2. Use a YouTube Video Statistics dataset to predict video views using Simple Linear Regression based on the number of likes. Convert the target variable into Popular and Non-Popular categories using a suitable threshold and apply Logistic Regression. Evaluate the regression model using MAE, MSE, RMSE, and R^2 Score, and evaluate the classification model using Accuracy, Precision, Recall, F1-Score, and Confusion Matrix.

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion: __/___/____**

Assignment 5

Supervised Machine Learning Models for Classification (KNN, Random Forest)

No. of slots - 2

Objectives

Students will be able to:

- Understand classification process and its different terminologies
- Understand the working of the K-Nearest Neighbors algorithm based on distance measures
- Comprehend the concept of ensemble learning in machine learning
- Study and implement the working of the Random Forest algorithm for classification tasks

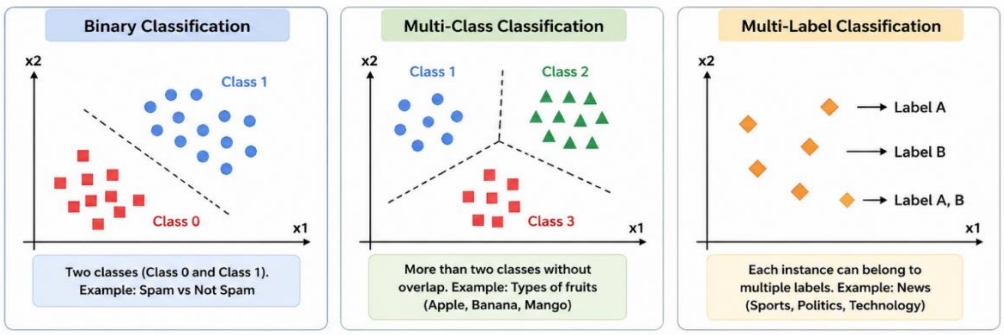
Introduction

Classification is a supervised machine learning technique used to assign input data into predefined categories based on patterns learned from labeled datasets. It predicts the class of new or unseen data points by analyzing previously known examples where the correct output is already defined. This approach is widely used in real-world applications such as identifying emails as spam or non-spam and classifying patients as diseased or healthy.

Types of Classification

Classification in machine learning involves sorting data into categories based on their features or characteristics. The type of classification problem depends on how many classes exist and how the categories are structured.

1. **Binary Classification**-Binary classification is the simplest type of classification where data is divided into two possible categories.
2. **Multiclass Classification**-Multiclass classification is used when data needs to be divided into more than two categories
3. **Multi Label Classification**-Multi label classification allows a single piece of data to belong to multiple categories at the same time. Unlike multiclass classification, where each data point is assigned only one class, this approach allows the model to assign multiple labels to the same input.



Classification Process

Classification involves training a model using labeled data, where each input is associated with a known output class. The model learns to recognize patterns in the data that differentiate between classes. Once trained, the model can predict the class label for new, unseen data.

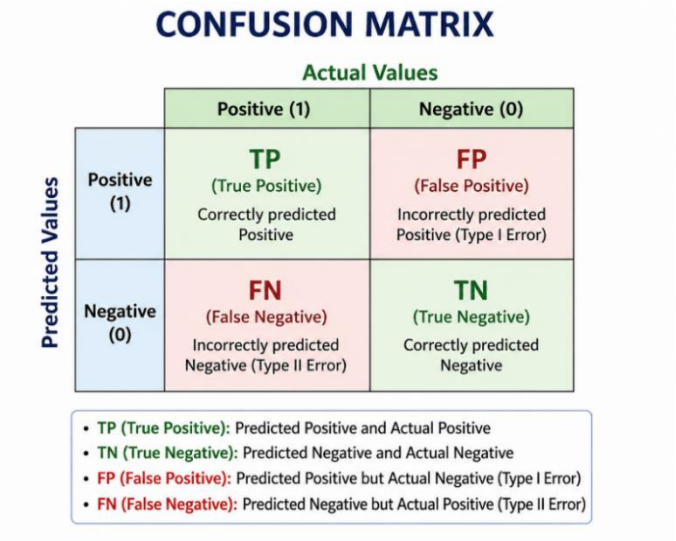
Key Steps in the Classification Process

1. **Data Collection & Labeling:** Gathering data where each item is pre-labeled with the correct class (e.g., fraudulent/legitimate transaction).
2. **Feature Extraction:** Identifying key characteristics (e.g., pixel patterns in an image, words in an email) that help distinguish between classes.
3. **Model Training:** An algorithm (like KNN or Random Forest) analyzes the labeled data to find the underlying patterns connecting features to labels.
4. **Model Evaluation:** The model is tested on new data to measure accuracy, precision, and recall.
5. **Prediction:** The trained model assigns a class label to new, unseen data.

Classification Metrics for model evaluation

In classification problems, model performance is evaluated using different metrics that measure how accurately the model predicts class labels.

1. Confusion Matrix: A confusion matrix is a table (NXN matrix) used to evaluate the performance of a classification model.



2. Accuracy: Measures overall correctness of the model.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

3. Precision: Measures how many predicted positives are actually correct.

$$Precision = \frac{TP}{TP + FP}$$

4. **Recall (Sensitivity):** Measures how many actual positives are correctly identified.

$$Recall = \frac{TP}{FN + TP}$$

5. **F1-Score:** Harmonic mean of Precision and Recall.

$$F1\text{-}Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

6. **Specificity (True Negative Rate):** Measures how many actual negatives are correctly identified.

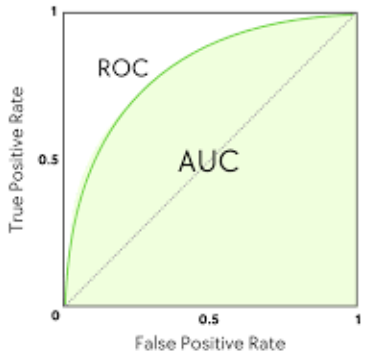
$$\text{Specificity} = \frac{TN}{TN + FP}$$

7. **ROC Curve:**

- Graph between **True Positive Rate (Recall)** and **False Positive Rate**
- Used to evaluate model performance at different thresholds

8. **AUC (Area Under Curve):**

- Measures overall performance of model
- Value ranges from **0 to 1**
  - 1 → Perfect model
  - 0.5 → Random model



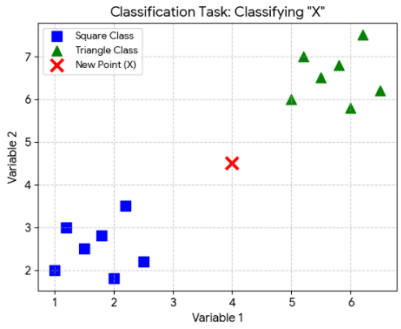
Classification Algorithms

I) K-Nearest Neighbors (KNN) Algorithm

K-Nearest Neighbors (KNN) is a simple supervised learning algorithm used for classification. It does not assume any predefined structure of the data, so it is called non-parametric. It is also known as a lazy learner because it does not train a model in advance.

When a new data point is given, KNN finds the distance between this point and all the training data points using distance measures like Euclidean or Manhattan. It then chooses the K closest points. The new data point is assigned to the class that is most common among these K nearest neighbors.

Let’s see this algorithm in action with the help of simple example. Suppose you have a dataset with two variables, which when plotted, looks like the following figure.



To classify a new data point represented by "X" into either the "Square" class or the "Triangle" class, we typically look at the proximity of the new point to the existing data clusters. This is a fundamental concept in supervised learning known as Classification.

In the generated plot:

- The **Blue Squares** represent the "Square" class, clustered in the lower-left region.
- The **Green Triangles** represent the "Triangle" class, clustered in the upper-right region.
- The **Red 'X'** represents the new, unlabeled data point that needs to be classified.

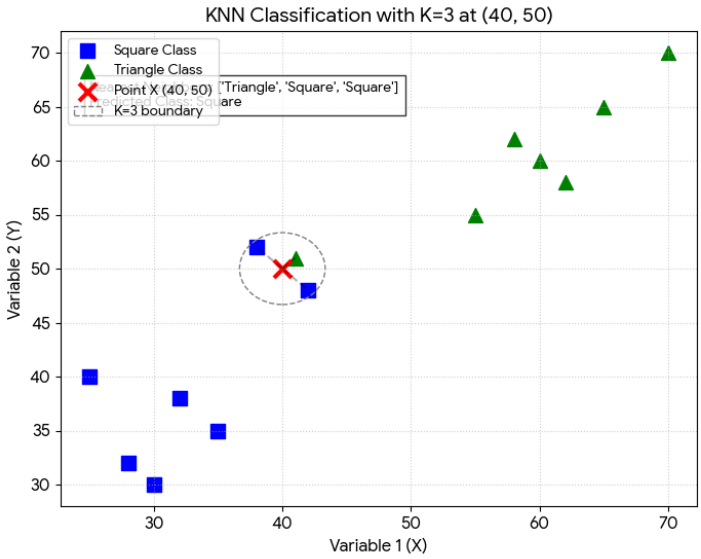
To classify "X", the KNN algorithm follows these steps:

1. **Calculate Distance:** It calculates the distance (usually Euclidean distance) between "X" and all other points in the dataset. The Euclidean distance between two points P1(x1, y1) and P2(x2, y2) is given by:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

2. **Find Neighbors:** It identifies the *k* closest points (neighbors) to "X".
3. **Majority Vote:** "X" is assigned to the class that is most common among its *k* nearest neighbors.

For example, if we choose *k*=3 and the three closest points to "X" are all Triangles, "X" would be classified into the Triangle class. Based on the visual layout in the plot, "X" appears to be positioned closer to the Triangle cluster, suggesting it would likely be classified as a Triangle depending on the specific value of *k* and the distribution of the data.



Based on the coordinates X=40, Y=50 and using *K*=3, the algorithm has identified the three closest data points. To complete the classification, we look at the labels of these three nearest neighbors.

The Final Step: Majority Voting

Once the *K* nearest points are found, the algorithm performs a majority vote. In this specific case:

1. **Identify Neighbors:**

   The 3 nearest points consist of:

   - 2 points from the Square class.
   - 1 point from the Triangle class.

2. **Calculate Majority:** Since the Square class has the most representatives among the neighbors (2 out of 3), it wins the vote.

3. **Result:** The new data point "X" is officially classified into the Square class.

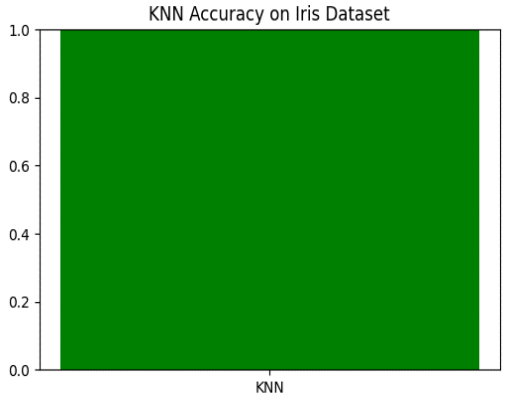
   This demonstrates why the choice of K is so important; a different *K* value might include more triangles and potentially change the result. Typically, an odd number for *K* is chosen to avoid ties in binary classification tasks.

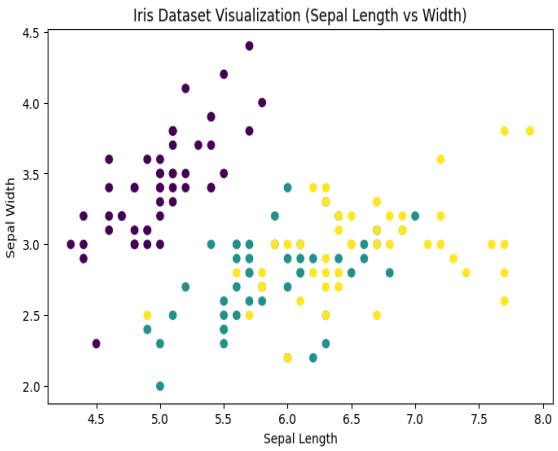
Python code for the implementation of KNN algorithm

import numpy as np import matplotlib.pyplot as plt from sklearn.datasets import load_iris from sklearn.model_selection import train_test_split from sklearn.preprocessing import StandardScaler from sklearn.neighbors import KNeighborsClassifier from sklearn.metrics import accuracy_score,	**Output**

```
confusion_matrix, classification_report
# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Feature scaling (important for KNN)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Prediction
y_pred = knn.predict(X_test)

# Evaluation
print("\nKNN on Iris Dataset")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n",
confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n",
classification_report(y_test, y_pred))

# New prediction
new_flower = np.array([[5.1, 3.5, 1.4, 0.2]])
new_flower_scaled = scaler.transform(new_flower)
prediction = knn.predict(new_flower_scaled)
print("\nPrediction for new flower:",
iris.target_names[prediction][0])
```

```
KNN on Iris Dataset
Accuracy: 1.0

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30


Prediction for new flower: setosa
```





```
# Accuracy comparison graph
plt.figure(figsize=(6,4))
plt.bar(["KNN"], [accuracy_score(y_test, y_pred)],
color="green")
plt.title("KNN Accuracy on Iris Dataset")
plt.ylim(0, 1)
plt.show()

# Visualization (2D plot using first two features)
plt.figure(figsize=(8,5))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap="viridis")
plt.title("Iris Dataset Visualization (Sepal Length vs
Width)")
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.show()
```

The model is evaluated using accuracy, classification report, and confusion matrix. The error rate analysis for different values of K helps in selecting the optimal number of neighbors. It is observed that KNN performs effectively for this dataset when an appropriate value of K is chosen.

II) Random Forest Algorithm

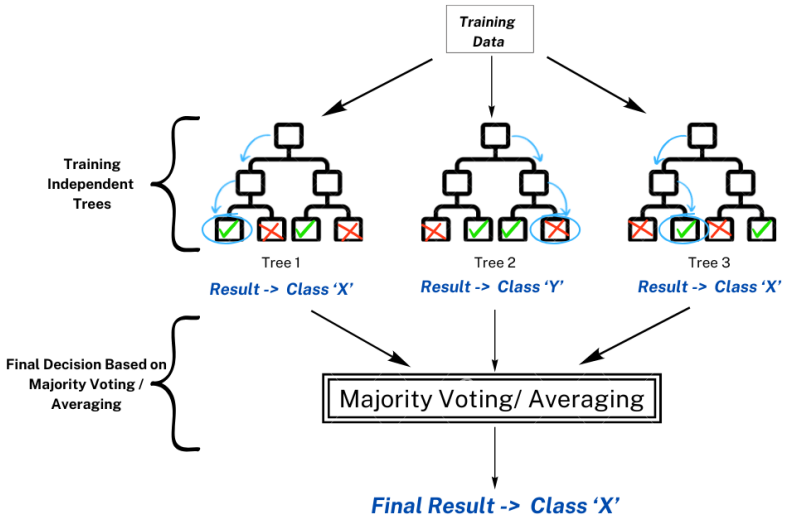
Random Forest is a supervised machine learning algorithm used for both classification and regression tasks. It is an ensemble learning technique that combines multiple decision trees to improve accuracy and reduce overfitting. Instead of relying on a single model, Random Forest builds many trees and combines their outputs to make a final prediction.

Random Forest is a collection (forest) of multiple decision trees where:

- Each tree is trained on a **random subset of data**
- Each tree uses a **random subset of features**
- Final prediction is made using **majority voting**

Working of Random Forest Algorithm:

1. Select random samples from dataset (**Bootstrap Sampling**)
2. Build multiple decision trees
3. Each tree gives a prediction
4. Combine predictions using **majority voting**



Before implementing random forest classifier in Python let's first understand its parameters

- *n_estimators* : Number of trees in the forest
- *max_depth* : Maximum depth of each tree
- *max_features* :Number of features considered for splitting at each node
- *criterion*: Function used to measure split quality('gini' or 'entropy')
- *min_samples_split* : Minimum samples required to split a node
- *min_samples_leaf*: Minimum samples required to be at a leaf node
- *bootstrap*: whether to use bootstrap sampling when building trees(True or False)

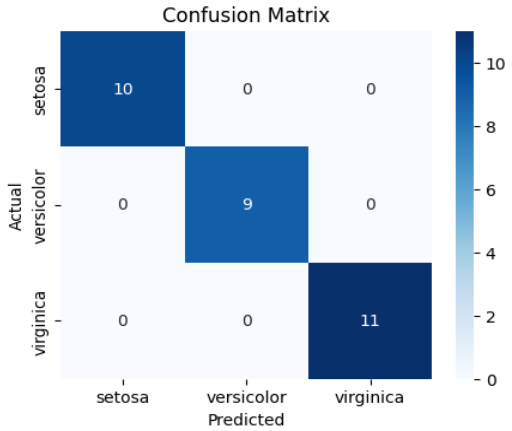
Random Forest Classification on Iris Dataset

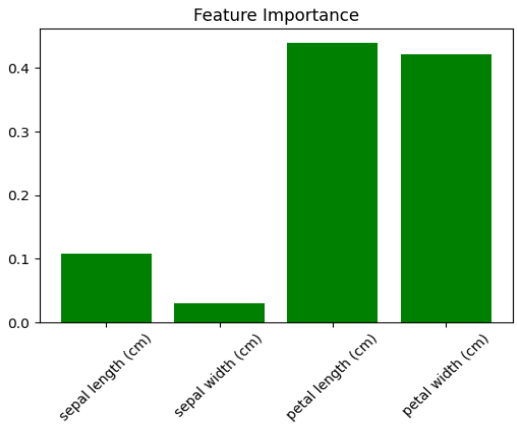
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import
RandomForestClassifier
from sklearn.metrics import accuracy_score,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Model
model = RandomForestClassifier(n_estimators=100,
random_state=42)
model.fit(X_train, y_train)

# Prediction
```

Output

```
Accuracy: 100.0 %
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Prediction: setosa
```

```
y_pred = model.predict(X_test)

# Accuracy in percentage
accuracy = accuracy_score(y_test, y_pred) * 100

print("Accuracy:", round(accuracy, 2), "%")
print("Confusion Matrix:\n",
confusion_matrix(y_test, y_pred))

# New prediction
sample = [[5.1, 3.5, 1.4, 0.2]]
print("Prediction:",
iris.target_names[model.predict(sample)[0]])

# Confusion Matrix Graph
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, cmap="Blues",
            xticklabels=iris.target_names,
            yticklabels=iris.target_names)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Feature Importance Graph
plt.figure(figsize=(6,4))
plt.bar(iris.feature_names,
model.feature_importances_, color="green")
plt.title("Feature Importance")
plt.xticks(rotation=45)
plt.show()
```





From the graph we can see that petal width (cm) is the most important feature followed closely by petal length (cm).The sepal width (cm) and sepal length(cm) have lower importance in determining the model's predictions. This indicates that the classifier relies more on the petal measurements to make predictions about the flower species.

Random Forest Classifiers are useful for classification tasks offering high accuracy and robustness.

They are easy to use, provide insights into feature importance and can handle complex datasets.

Lab Assignment

SET A:

1. Implement the K-Nearest Neighbors (KNN) algorithm on the Wine dataset. Perform data preprocessing, model training, testing, and performance evaluation. Compare the results for different values of K.
2. Apply the Random Forest algorithm on the Heart Disease dataset to predict the presence of heart disease. Preprocess the data, evaluate the model using accuracy and confusion matrix, and identify important features affecting the prediction.
3. Use the KNN algorithm to classify emails as spam or non-spam based on selected features. Evaluate the classifier using suitable performance metrics.
   Attributes: [World_Counr, Links_Count, Special_Characters, Sender_Reputation, Spam]
4. Apply the Random Forest algorithm to predict house prices based on property attributes. Evaluate the model performance and analyze the importance of features influencing the prediction.

SET B:

1. Implement a KNN classifier on a Movie dataset. Preprocess the data, evaluate the model using accuracy and confusion matrix.
2. Develop a Random Forest model to classify mobile phones into different price ranges based on their specifications. Evaluate the classifier and identify the most influential features.

SET C:

1. Apply KNN and Random Forest algorithms to classify Instagram influencers into different engagement categories based on followers, likes, comments, and posts. Compare the performance of both classifiers.
2. Implement KNN and Random Forest algorithms to predict product rating categories based on customer reviews, purchase history, and product features. Compare the effectiveness of both classifiers.

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion: __/___/____**

Assignment No-6

Unsupervised Machine Learning Models for Clustering (K-means clustering) and Association Rule Mining (Apriori Algorithm)

No. of slots - 2

Objectives

Students will be able to:

- Understand the concept of unsupervised learning
- Learn the working of K-Means clustering algorithm
- Understand association rule mining and frequent itemsets
- Implement Apriori algorithm to generate association rules

Introduction

Unsupervised Machine Learning is a type of machine learning in which the model learns patterns from data without any predefined labels. It is mainly used to discover hidden structures, relationships, or groupings within the data. Two important techniques in unsupervised learning are clustering and association rule mining.

Clustering is the process of grouping similar data points together based on their characteristics. One of the most widely used clustering algorithms is K-Means clustering, which partitions the dataset into a predefined number of clusters by minimizing the distance between data points and their respective cluster centroids. It is commonly used in applications such as customer segmentation, image processing, and pattern recognition.

Association Rule Mining is used to identify interesting relationships or patterns among variables in large datasets. The Apriori algorithm is a popular method for generating frequent itemsets and discovering association rules based on measures such as support, confidence, and lift. It is widely applied in market basket analysis, recommendation systems, and business analytics.

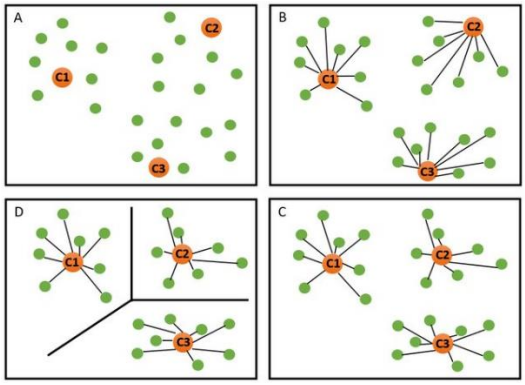
Overall, these techniques help in extracting meaningful insights from data without prior knowledge of output labels, making them essential tools in data analysis and decision-making.

I. K-means Clustering Algorithm

K-Means is a popular unsupervised machine learning algorithm used for partitioning a dataset into K clusters based on the similarity of their features. It works by minimizing the variance within each cluster and is widely used in applications such as market segmentation, image compression, and pattern recognition.

K means clustering, assigns data points to one of the K clusters depending on their distance from the center of the clusters. Here K defines the number of pre-defined clusters that need to be created in the

process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.It starts by randomly assigning the clusters centroid in the space. Then each data point assign to one of the cluster based on its distance from centroid of the cluster. After assigning each point to one of the cluster, new cluster centroids are assigned. This process runs iteratively until it finds good cluster. In the analysis we assume that number of cluster is given in advanced and we have to put points in one of the group.



(A) Three centroids are randomly chosen

(B) Objects are assigned to clusters

(C) Objects are assigned to clusters

(D) The centroids have moved to new positions

ALGORITHM:

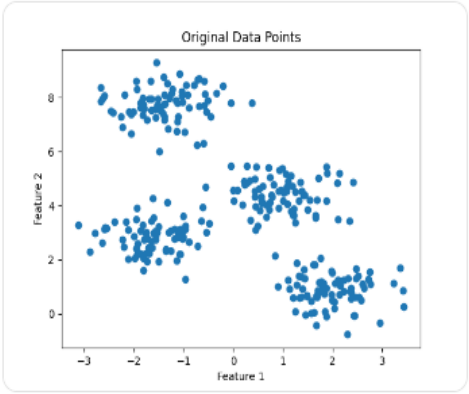
- Choose the number of clusters (K)
- Select K random data points as initial centroids
- Assign each data point to the nearest centroid based on distance (usually Euclidean distance)
- Recalculate centroids by taking the mean of all points in each cluster
- Repeat assignment and centroid update steps until centroids do not change or reach maximum iterations
- Final clusters are formed when convergence is achieved

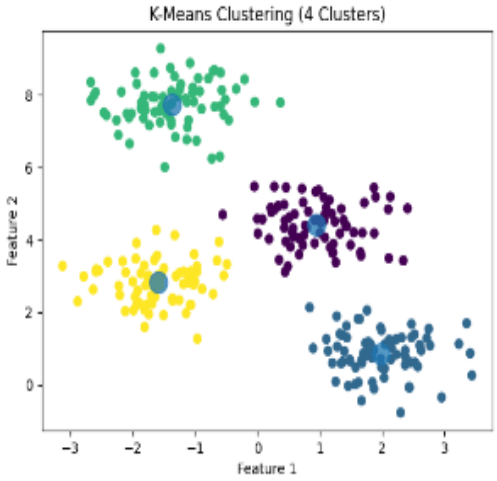
Python code for K-means Clustering (4 Clusters)-

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
X, _ =
make_blobs(n_samples=300,centers=4,cluster_std=0.60,
random_state=0)
plt.figure()
plt.scatter(X[:, 0], X[:, 1])
plt.title("Original Data Points")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
kmeans = KMeans(n_clusters=4, n_init=10,
random_state=0)
kmeans.fit(X)

centers = kmeans.cluster_centers_
labels = kmeans.labels_
plt.figure()
plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.scatter(centers[:, 0], centers[:, 1], s=200, alpha=0.75)

plt.title("K-Means Clustering (4 Clusters)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
print("K-Means clustering applied successfully with 4
clusters.")
```

Output-

```
K-Means clustering applied successfully with 4 clusters.
```





II. Association Rule Mining (Apriori Algorithm)

Frequent Itemset Mining: Finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories.

Association Mining searches for frequent items in the data-set. In frequent mining usually the interesting associations and correlations between item sets in transactional and relational databases are found.

If there are 2 items X and Y purchased frequently then it is good to put them together in stores or provide some discount offer on one item on purchase of other item. This can really increase the sales. For example it is likely to find that if a customer buys Milk and bread he/she also buys Butter. So the association rule is [‘milk]^[‘bread’]=>[‘butter’].

Applications:

Market Basket Analysis is one of the key techniques used by large retailers to uncover associations between item, catalog design, loss-leader analysis, clustering, classification, recommendation systems, etc.

Consider the following Transaction database:

D= {{butter, bread, milk, sugar};

{butter, flour, milk, sugar};

{butter, eggs, milk, salt};

{eggs};

{butter, flour, milk, salt, sugar}}

Question of interest: Which items are bought together frequently?

Apriori is an algorithm for frequent item set mining and association rule learning over relational databases. The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemset properties. Apriori employs an iterative approach known as a levelwise search, where k-itemsets are used to explore (k+1)-itemsets.

To construct association rules between items, the algorithm considers 3 important factors which are, support, confidence and lift. Each of these factors is explained as follows:

Support:

The support of item I is defined as the ratio between the number of transactions containing the item I by the total number of transactions expressed as :

$$support(I) = \frac{Number\ of\ transactions\ containing\ I}{Total\ number\ of\ transactions}$$

Confidence:

This is measured by the proportion of transactions with item I1, in which item I2 also appears.

$$confidence(I1 \rightarrow I2) = \frac{Number\ of\ transactions\ containing\ items\ I1\ and\ I2}{Total\ number\ of\ transactions\ containing\ I1}$$

Given that the item on the left hand side (antecedent) is purchased then the item on the right hand side(consequent) would also be purchased.

Lift:

Lift is the ratio between the confidence and support expressed as :

$$lift(I1 \rightarrow I2) = \frac{confidence(I1 \rightarrow I2)}{support(I2)}$$

Lift (antecedent => consequent) = 1 means that there is no correlation within the itemset, > 1 means that there is a positive correlation within the itemset, i.e., products in the itemset, antecedent, and consequent, are more likely to be bought together, < 1 means that there is a negative correlation within the itemset, i.e., products in itemset, antecedent, and consequent, are unlikely to be bought together.

The steps of the apriori algorithm can be given as:-

1. Define the minimum support and confidence for the association rule
2. Take all the subsets in the transactions with higher support than the minimum support
3. Take all the rules of these subsets with higher confidence than minimum confidence
4. Sort the association rules in the decreasing order of lift.
5. Visualize the rules along with confidence and support.

In this assignment you will analyze collections of market baskets and will determine frequent itemsets and association rules present in the collections.

Python libraries

Python has many libraries for apriori implementation.

i. Mlxtend (apriori)

ii. Apyori (apriori)

iii. pypi (efficient_apriori)

The apriori module from mlxtend library provides fast and efficient apriori implementation.

Usage:

apriori(df, min_support=0.5, use_colnames=False, max_len=None, verbose=0, low_memory=False)

Parameters

• *df* : One-Hot-Encoded DataFrame or DataFrame that has 0 and 1 or True and False as values

• *min_support* : Floating point value between 0 and 1 that indicates the minimum support required for an itemset to be selected.

# of observation with item / total observation# of observation with item / total observation

• *use_colnames* : This allows to preserve column names for itemset making it more readable.

• *max_len* : Max length of itemset generated. If not set, all possible lengths are evaluated.

• *verbose* : Shows the number of iterations if >= 1 and low_memory is True. If =1 and low_memory is False , shows the number of combinations.

• *low_memory* :

• If True, uses an iterator to search for combinations above min_support. Note that while low_memory=True should only be used for large dataset if memory resources are limited,because this implementation is approx. 3–6x slower than the default.

The function returns a pandas DataFrame with columns ['support', 'itemsets'] of all itemsets that are >= min_support and < than max_len (if max_len is not None).

Mining Association Rules

Frequent if-then associations called association rules which consists of an antecedent (if) and a consequent (then). The following function returns the association rules from the frequent itemsets which satisfy the given metric threshold. Metric can be set to confidence, lift, support,leverage and conviction.

association_rules(frequent_items, metric='confidence', min_threshold=0.5, support_only = False)

Leverage computes the difference between the observed frequency of A and C appearing together and the frequency that would be expected if A and C were independent. A leverage value of 0 indicates independence. A high conviction value means that the consequent is highly depending on the antecedent.

```
# Install mlxtend if not already installed
# !pip install mlxtend
import pandas as pd
from mlxtend.preprocessing import
TransactionEncoder
from mlxtend.frequent_patterns import
apriori, association_rules

# Sample transaction dataset
transactions = [
    ['eggs', 'milk', 'bread'],
    ['eggs', 'apple'],
    ['milk', 'bread'],
    ['apple', 'milk'],
    ['milk', 'apple', 'bread']]

#Convert transactions into one-hot
encoded DataFrame
te = TransactionEncoder()
te_array =
te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_array,
columns=te.columns_)
```

Output:

Transaction Dataset (Encoded Table)

Transaction	Apple	Bread	Eggs	Milk
T1	False	True	True	True
T2	True	False	True	False
T3	False	True	False	True
T4	True	False	False	True
T5	True	True	False	True

Frequent Itemsets (min_support = 0.4)

Support	Itemset
0.60	{Apple}
0.60	{Bread}
0.40	{Eggs}
0.80	{Milk}
0.40	{Apple, Milk}
0.60	{Bread, Milk}

```
print("Transaction Dataset:")
print(df)
# Generate frequent itemsets
freq_items = apriori(df, min_support=0.4,
use_colnames=True)
print("\nFrequent Itemsets:")
print(freq_items)
# Generate association rules
rules = association_rules(freq_items,
                    metric="confidence",
                    min_threshold=0.5)

print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents',
          'support', 'confidence', 'lift']])
```

Association Rules

Antecedent	Consequent	Support	Confidence	Lift
{Apple}	{Milk}	0.40	0.667	0.833
{Milk}	{Apple}	0.40	0.500	0.833
{Bread}	{Milk}	0.60	1.000	1.250
{Milk}	{Bread}	0.60	0.750	1.250

Lab Assignment

SET A:

1. Generate a dataset using built-in libraries. Plot the data points and observe their distribution. Apply the K-Means clustering algorithm with a predefined number of clusters (K=3 or K=4). Visualize the clustered data along with cluster centroids.
2. Create the following dataset in python

TID	Items
1	Python, DBMS
2	Python, Java, AI
3	DBMS, Data Science
4	Python, DBMS, Java
5	AI, Data Science
6	Python, AI
7	Java, DBMS
8	Python, Data Science
9	Python, Java, DBMS, AI
10	DBMS, Data Science, AI

   Create the following dataset in Python. Apply the Apriori algorithm to generate frequent itemsets and association rules. Repeat the process with different minimum support values (0.2, 0.3, 0.4, and 0.5).

3. Consider a dataset containing students' attendance percentage and examination marks. Apply the **K-Means Clustering** algorithm to group students into clusters based on their academic performance. Visualize the clusters using a scatter plot, identify the cluster centroids, and interpret the characteristics of each cluster.

4. Apply the Apriori algorithm to discover frequently used learning resources and generate association rules among them.

TID	Items
1	Video Lecture, Quiz
2	Video Lecture, Assignment
3	Quiz, Discussion Forum
4	Video Lecture, Quiz, Assignment
5	Assignment, Discussion Forum
6	Video Lecture, Certificate
7	Quiz, Certificate
8	Video Lecture, Quiz
9	Assignment, Certificate
10	Video Lecture, Quiz, Certificate

SET B:

1. Download a real-world Customer Segmentation dataset (Mall Customers dataset). Preprocess the data by handling missing values and performing feature scaling. Apply the K-Means clustering algorithm to segment customers based on their annual income and spending score.
2. Download a Hotel Booking dataset. Perform preprocessing by handling missing values and selecting relevant numerical features such as booking duration, number of guests, and stay cost. Apply the K-Means clustering algorithm to group bookings into different categories.
3. Download the groceries dataset. Write a python program to read the dataset and display its information. Preprocess the data (drop null values etc.) Convert the categorical values into numeric format. Apply the apriori algorithm on the above dataset to generate the frequent itemsets and association rules.

SET C:

1. Collect a stock market dataset containing features such as price, volume, and volatility. Preprocess the data by handling missing values and normalizing the features. Apply the K-Means clustering algorithm to group stocks based on performance patterns. Additionally, apply the Apriori algorithm to discover frequent patterns and relationships among stock attributes.

Assignment Evaluation

0: Not Done	[ ]	1: Incomplete	[ ]	2: Late Complete	[ ]
3: Needs Improvement	[ ]	4: Complete	[ ]	5: Well Done	[ ]

Signature of the Instructor **Date of Completion: __/___/____**