EXPERIMENT -1

AIM : To implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

THEORY : Find-S (Find Specific) is a concept learning algorithm used in machine learning to find the most specific hypothesis that fits all positive training examples. It starts with the most specific hypothesis and gradually generalizes it when a positive example does not match. It ignores all negative examples. A hypothesis is represented using attribute values, and "?" is used to represent any value. The goal is to find a hypothesis h that correctly matches all positive examples and is consistent with the target concept.

ALGORITHM:

Step 1: Initialize the hypothesis h to the most specific form (all attributes as Ø).

Step 2: For each training example:
- If the example is negative, ignore it.
- If the example is positive:
  - For each attribute i:
    - If h[i] = Ø, then set h[i] = value of attribute i in example.
    - Else if h[i] ≠ value of attribute i, then set h[i] = ?.
    - Else keep h[i] unchanged.

Step 3: After all examples, output the final hypothesis h.

PROS AND CONS:

PROS:

1. Simple and easy to understand.
2. Fast and efficient.
3. Good for beginners in machine learning.
4. Produces a clear and specific hypothesis.

CONS:

1. Uses only positive examples.
2. Cannot handle noisy or incorrect data.
3. Ignores negative examples.
4. Finds only the most specific hypothesis.
5. Limited to simple problems.

CODE:

```
import pandas as pd

df=pd.read_csv("Sports_data.csv")

df2=df[df['enjoy sport']=='yes']
df2=df2.drop(columns=['enjoy sport'])

h1=['Ø']*(len(df2.columns))
k=0
for i in df2.columns:
    for j in df2[i]:
        if(h1[k]=='Ø'):
            h1[k]=j

        elif(j!=h1[k]):
            h1[k]='?'
    k+=1
print("Final Hypothesis")
print(h1)
```

RESULTS:

Dataset 1 : Sports Dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   sky          4 non-null      object
 1   air temp     4 non-null      object
 2   humidity     4 non-null      object
 3   wind         4 non-null      object
 4   water        4 non-null      object
 5   forecast     4 non-null      object
 6   enjoy sport  4 non-null      object
dtypes: object(7)
memory usage: 352.0+ bytes

{'sky': ['sunny', 'rainy'], 'air temp': ['warm', 'cold'], 'humidity': ['normal', 'high'], 'wind': ['strong'],
'water': ['warm', 'cool'], 'forecast': ['same', 'change'], 'enjoy sport': ['yes', 'no']}

Final Hypothesis
['sunny', 'warm', '?', 'strong', '?', '?']
```

Dataset 2 : Restraunt Dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Veg      6 non-null      object
 1   Spicy    6 non-null      object
 2   Cheap    6 non-null      object
 3   Street   6 non-null      object
 4   Popular  6 non-null      object
 5   Yes      6 non-null      object
dtypes: object(6)
memory usage: 416.0+ bytes
None

{'Veg': ['Veg', 'Non-Veg'], 'Spicy': ['Normal', 'Spicy'], 'Cheap': ['Cheap', 'Expensive'], 'Street':
['Restaurant', 'Street'], 'Popular': ['Popular', 'New'], 'Yes': ['Yes', 'No']}

Final Hypothesis
['Veg', '?', 'Cheap', '?', '?']
```

RESULT ANALYSIS:
DATASET 1-

- **Attributes Used:**
  sky, air temperature, humidity, wind, water, forecast, enjoy sport
- **Final Hypothesis Obtained:**
  ['sunny', 'warm', '?', 'strong', '?', '?']
- **Analysis:**
  - The algorithm starts with the first positive example, so the hypothesis becomes exactly like it.
  - As more positive examples are processed, mismatched attribute values are replaced with "?".
  - Negative examples are ignored.
  - After all positive examples, only sky = sunny, air temp = warm, and wind = strong remain fixed.
  - This means sport is enjoyed whenever it is **sunny**, **warm**, and **wind is strong**, no matter what the other attributes are.
- **Conclusion:**
  Find-S learned that enjoy sport = **YES** when:
  sky = sunny, air temperature = warm, wind = strong.
  Other values can be anything.

DATASET 2

- **Attributes Used:**
  Veg/Non-Veg, Spicy, Cheap/Expensive, Street/Restaurant, Popular/New, Like Food
- **Final Hypothesis Obtained:**
  ['Veg', '?', 'Cheap', '?', '?']
- **Analysis:**
  - The first positive example makes the initial hypothesis.
  - For every next positive example, attribute mismatches are replaced by "?".
  - Negative examples are ignored as per Find-S algorithm.
  - Only "Veg" and "Cheap" stay consistent across all positive examples.
- **Conclusion:**
  Food is liked when it is **Veg** and **Cheap**, and other attributes can take any value.

Observation: FIND-S correctly identifies the essential attributes affecting the outcome, generalizing only where necessary, while ignoring irrelevant attributes.

REFERENCES :

https://github.com/kevinadhiguna/find-S-algorithm
https://www.geeksforgeeks.org/machine-learning/ml-find-s-algorithm/

EXPERIMENT -2

AIM: For a given set of training data examples stored in a .CSV file, implement and demonstrate the CandidateElimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

THEORY: The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

- You can consider this as an extended form of the Find-S algorithm.
- Consider both positive and negative examples.
- Actually, positive examples are used here as the Find-S algorithm (Basically they are generalizing from the specification).
- While the negative example is specified in the generalizing form.

Terms Used:

- **Concept learning:** Concept learning is basically the learning task of the machine (Learn by Train data)
- **General Hypothesis:** Not Specifying features to learn the machine.
- **G = {'?', '?','?','?'...}:** Number of attributes
- **Specific Hypothesis:** Specifying features to learn machine (Specific feature)
- **S= {'pi','pi','pi'...}: The number** of pi depends on a number of attributes.
- **Version Space:** It is an intermediate of general hypothesis and Specific hypothesis. It not only just writes one hypothesis but a set of all possible hypotheses based on training data-set.

ALGORITHM:

Step1: Load Data set

Step2: Initialize General Hypothesis and Specific Hypothesis.

Step3: For each training example

Step4: If example is positive example

```
if attribute_value == hypothesis_value:
   Do nothing
else:
   replace attribute value with '?' (Basically generalizing it)
```

Step5: If example is Negative example

Make generalize hypothesis more specific.

PROS AND CONS:

PROS:

1. **Improved accuracy:** CEA considers both positive and negative examples to generate the hypothesis, which can result in higher accuracy when dealing with noisy or incomplete data.
2. **Flexibility:** CEA can handle more complex classification tasks, such as those with multiple classes or non-linear decision boundaries.
3. **More efficient:** CEA reduces the number of hypotheses by generating a set of general hypotheses and then eliminating them one by one. This can result in faster processing and improved efficiency.
4. **Better handling of continuous attributes:** CEA can handle continuous attributes by creating boundaries for each attribute, which makes it more suitable for a wider range of datasets.

CONS:

6. **More complex:** CEA is a more complex algorithm than Find-S, which may make it more difficult for beginners or those without a strong background in machine learning to use and understand.

7. **Higher memory requirements:** CEA requires more memory to store the set of hypotheses and boundaries, which may make it less suitable for memory-constrained environments.

8. **Slower processing for large datasets:** CEA may become slower for larger datasets due to the increased number of hypotheses generated.

9. **Higher potential for overfitting:** The increased complexity of CEA may make it more prone to overfitting on the training data, especially if the dataset is small or has a high degree of noise.

CODE:

```
import numpy as np
import pandas as pd

data = pd.read_csv('ceadataset2.csv')

concepts = np.array(data.iloc[:, 0:-1])
target   = np.array(data.iloc[:, -1])

def learn(concepts, target):
    # Take first training example as initial specific hypothesis
    specific_h = concepts[0].copy()
```

```
    print("Initial Specific_h:\n", specific_h)

    general_h = [["?" for _ in range(len(specific_h))]
                 for _ in range(len(specific_h))]
    print("Initial General_h:\n", general_h)
    print("\n")

    for i, h in enumerate(concepts):
        if target[i].lower() == "yes":   # Positive example
            print(f"Instance {i+1} is Positive")
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'

        elif target[i].lower() == "no":  # Negative example
            print(f"Instance {i+1} is Negative")
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Step", i+1)
        print("Specific_h:", specific_h)
        print("General_h:", general_h)
        print("\n")

    empty = ["?"] * len(specific_h)
    general_h = [g for g in general_h if g != empty]

    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print("Final Specific_h:")
print(s_final)

print("\nFinal General_h:")
for g in g_final:
    print(g)
```

RESULTS:

Dataset 1 : Movie Recomendation Dataset

Genre,AgeGroup,Language,ActorFamiliarity,RatingPreference,Platform,Recommended

```
Action,Young,English,High,High,Theatre,yes
Drama,Adult,Hindi,Low,Medium,OTT,no
Comedy,Teen,English,Medium,High,OTT,yes
Horror,Young,Hindi,Low,Low,Theatre,no
Action,Adult,English,Medium,High,OTT,yes
Romance,Teen,Hindi,Low,Medium,OTT,no
Comedy,Adult,English,High,High,Theatre,yes
Horror,Senior,Hindi,Medium,Low,OTT,no

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Genre            8 non-null      object
 1   AgeGroup         8 non-null      object
 2   Language         8 non-null      object
 3   ActorFamiliarity  8 non-null      object
 4   RatingPreference  8 non-null      object
 5   Platform         8 non-null      object
 6   Recommended      8 non-null      object
dtypes: object(7)
memory usage: 576.0+ bytes
None
{'Genre': ['Action', 'Drama', 'Comedy', 'Horror', 'Romance'], 'AgeGroup': ['Young', 'Adult', 'Teen',
'Senior'], 'Language': ['English', 'Hindi'], 'ActorFamiliarity': ['High', 'Low', 'Medium'],
'RatingPreference': ['High', 'Medium', 'Low'], 'Platform': ['Theatre', 'OTT'], 'Recommended': ['yes',
'no']}
Initial Specific_h:
 ['Action' 'Young' 'English' 'High' 'High' 'Theatre']
Initial General_h:
 [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is Positive
Step 1
Specific_h: ['Action' 'Young' 'English' 'High' 'High' 'Theatre']
General_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 2 is Negative
Step 2
Specific_h: ['Action' 'Young' 'English' 'High' 'High' 'Theatre']
General_h: [['Action', '?', '?', '?', '?', '?'], ['?', 'Young', '?', '?', '?', '?'], ['?', '?', 'English', '?', '?', '?'], ['?',
'?', '?', 'High', '?', '?'], ['?', '?', '?', '?', 'High', '?'], ['?', '?', '?', '?', '?', 'Theatre']]
```

Instance 3 is Positive
Step 3
Specific_h: ['?' '?' 'English' '?' 'High' '?']
General_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', 'English', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'High', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 4 is Negative
Step 4
Specific_h: ['?' '?' 'English' '?' 'High' '?']
General_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', 'English', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'High', '?'], ['?', '?', '?', '?', '?', '?']]
Instance 5 is Positive
Step 5
Specific_h: ['?' '?' 'English' '?' 'High' '?']
General_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', 'English', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'High', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 6 is Negative
Step 6
Specific_h: ['?' '?' 'English' '?' 'High' '?']
General_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', 'English', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'High', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 7 is Positive
Step 7
Specific_h: ['?' '?' 'English' '?' 'High' '?']
General_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', 'English', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'High', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 8 is Negative
Step 8
Specific_h: ['?' '?' 'English' '?' 'High' '?']
General_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', 'English', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'High', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:
['?' '?' 'English' '?' 'High' '?']

Final General_h:
['?', '?', 'English', '?', '?', '?']
['?', '?', '?', '?', 'High', '?']

Dataset 2 : Laptop Purchase Dataset
Brand,Processor,RAM,Storage,Graphics,Budget,Purchase
Dell,Intel,i5,SSD,NVIDIA,high,yes
HP,Intel,i5,HDD,NVIDIA,medium,yes

```
Acer,AMD,i3,HDD,Integrated,low,no
Dell,Intel,i7,SSD,NVIDIA,high,yes
Lenovo,Intel,i5,SSD,Integrated,medium,no
HP,AMD,i7,SSD,NVIDIA,high,yes

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Brand      6 non-null      object
 1   Processor  6 non-null      object
 2   RAM        6 non-null      object
 3   Storage    6 non-null      object
 4   Graphics   6 non-null      object
 5   Budget     6 non-null      object
 6   Purchase   6 non-null      object
dtypes: object(7)
memory usage: 464.0+ bytes
None
{'Brand': ['Dell', 'HP', 'Acer', 'Lenovo'], 'Processor': ['Intel', 'AMD'], 'RAM': ['i5', 'i3', 'i7'],
'Storage': ['SSD', 'HDD'], 'Graphics': ['NVIDIA', 'Integrated'], 'Budget': ['high', 'medium', 'low'],
'Purchase': ['yes', 'no']}
Initial Specific_h:
 ['Dell' 'Intel' 'i5' 'SSD' 'NVIDIA' 'high']
Initial General_h:
 [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is Positive
Step 1
Specific_h: ['Dell' 'Intel' 'i5' 'SSD' 'NVIDIA' 'high']
General_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 2 is Positive
Step 2
Specific_h: ['?' 'Intel' 'i5' '?' 'NVIDIA' '?']
General_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 3 is Negative
Step 3
Specific_h: ['?' 'Intel' 'i5' '?' 'NVIDIA' '?']
General_h: [['?', '?', '?', '?', '?', '?'], ['?', 'Intel', '?', '?', '?', '?'], ['?', '?', 'i5', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', 'NVIDIA', '?'], ['?', '?', '?', '?', '?', '?']]
```

Instance 4 is Positive
Step 4
Specific_h: ['?' 'Intel' '?' '?' 'NVIDIA' '?']
General_h: [['?', '?', '?', '?', '?', '?'], ['?', 'Intel', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'NVIDIA', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 5 is Negative
Step 5
Specific_h: ['?' 'Intel' '?' '?' 'NVIDIA' '?']
General_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'NVIDIA', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 6 is Positive
Step 6
Specific_h: ['?' '?' '?' '?' 'NVIDIA' '?']
General_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'NVIDIA', '?'], ['?', '?', '?', '?', '?', '?']]
Final Specific_h:
['?' '?' '?' '?' 'NVIDIA' '?']

Final General_h:
['?', '?', '?', '?', 'NVIDIA', '?']

RESULT ANALYSIS:
DATASET 1-

- **Attributes Used:**
  Genre, AgeGroup, Language, ActorFamiliarity, RatingPreference, Platform, Recommended
- **Final Hypothesis Obtained:**
  [‘?‘,‘?‘,‘English‘,‘?‘,‘High‘,‘?‘]
- **Analysis:**
  The algorithm initially takes the first positive example and assigns its values to the specific hypothesis. As the subsequent positive instances are processed, the attribute values that do not match the current hypothesis are replaced by “?”. Negative examples do not change the specific hypothesis; they only affect the general hypothesis set.
  After all instances are processed, only the attributes Language = English and RatingPreference = High remain constant across all positive examples. The remaining attributes differ among the positive samples and therefore become generalized to “?”.
- **Conclusion:**
  The Candidate Elimination Algorithm concludes that a **movie is recommended (YES)** when:
  Language = English
  Rating Preference = High
  All other attributes (Genre, AgeGroup, ActorFamiliarity, Platform) may take any value without affecting the recommendation decision.

DATASET 2

- **Attributes Used:**
  Brand, Processor, RAM, Storage, Graphics, Budget, Purchase
- **Final Hypothesis Obtained:**
  [‘?‘,‘?‘,‘?‘,‘?‘,‘NVIDIA‘,‘?‘]
- **Analysis:**
  The algorithm begins by using the first positive instance to initialize the specific hypothesis. When further positive examples are processed, any attribute value conflicting with the current hypothesis is generalized to “?”. Negative examples do not modify the specific hypothesis since only positive samples contribute toward learning the most specific consistent hypothesis.
  After all positive instances have been evaluated, the only attribute that remained constant across all positive purchase decisions was **Graphics = NVIDIA**. All remaining attributes showed variation among the positive examples and were therefore replaced with “?”.

- **Conclusion:**
  The Candidate Elimination Algorithm concludes that a **laptop will be purchased (YES)** when:
  Graphics = NVIDIA
  All other characteristics — Brand, Processor, RAM, Storage, and Budget — may take any value without affecting the purchase decision.

Observation: The Candidate Elimination Algorithm gradually generalized the hypothesis with each positive example and restricted the version space using negative examples. Only the attributes common across all positive instances remained in the final hypothesis, while the varying attributes were replaced with “?”. This shows that the algorithm learns the exact conditions required for a positive outcome based solely on consistent patterns in the dataset.

REFERENCES :

https://github.com/DeepakDVallur/Candidate_Elimination
https://www.geeksforgeeks.org/machine-learning/ml-candidate-elimination-algorithm/

Experiment – 3

Aim – Apply Pandas, data visualization, and Data Preprocessing to the given dataset.

Pandas

This code imports the pandas library and loads the CSV file dataset_3.csv into a DataFrame named df, enabling structured data analysis and manipulation.

```
import pandas as pd
df=pd.read_csv("dataset_3.csv")
```

This code checks the shape of the DataFrame, showing that it contains 32 rows and 5 columns, which represents the dataset's dimensions.

```
df.shape
(32, 5)
```

This code retrieves and displays the column names of the DataFrame, indicating that the dataset contains five features: Duration, Date, Pulse, Maxpulse, and Calories.

```
df.columns
Index(['Duration', 'Date', 'Pulse', 'Maxpulse', 'Calories'], dtype='object')
```

This code provides a concise summary of the DataFrame, including the number of entries, column names, data types, and non-null counts for each column.

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  32 non-null     int64
 1   Date      31 non-null     object
 2   Pulse     32 non-null     int64
 3   Maxpulse  32 non-null     int64
 4   Calories  30 non-null     float64
dtypes: float64(1), int64(3), object(1)
memory usage: 1.4+ KB
```

This code displays the first five rows of the DataFrame, giving a quick overview of the dataset's structure and sample values.

df.head()

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0

This code displays the first seven rows of the DataFrame, allowing a slightly broader view of the dataset's initial records and values.

```
df.head(7)
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0

This code displays the last seven rows of the DataFrame, providing an overview of the dataset's ending records and values.

df.tail(7)

	Duration	Date	Pulse	Maxpulse	Calories
25	60	'2020/12/25'	102	126	334.5
26	60	20201226	100	120	250.0
27	60	'2020/12/27'	92	118	241.0
28	60	'2020/12/28'	103	132	NaN
29	60	'2020/12/29'	100	132	280.0
30	60	'2020/12/30'	102	129	380.3
31	60	'2020/12/31'	92	115	243.0

This code selects and displays the Calories column from the DataFrame, returning it as a pandas Series for focused analysis

df["Calories"]

0 409.1
1 479.0
2 340.0
3 282.4
4 406.0
5 300.0
6 374.0
7 253.3
8 195.1
9 269.0
10 329.3
11 250.7
12 250.7
13 345.3
14 379.3
15 275.0
16 215.2

```
17    300.0
18      NaN
19    323.0
20    243.0
21    364.2
22    282.0
23    300.0
24    246.0
...
28      NaN
29    280.0
30    380.3
31    243.0
Name: Calories, dtype: float64
```

This code selects the Calories and Date columns from the DataFrame and displays the first five rows, allowing a focused view of these two features.

```
df[["Calories","Date"]].head()
```

	Calories	Date
0	409.1	'2020/12/01'
1	479.0	'2020/12/02'
2	340.0	'2020/12/03'
3	282.4	'2020/12/04'
4	406.0	'2020/12/05'

This code uses integer-based indexing to select rows from index 1 to 3 of the DataFrame, extracting a specific subset of records.

```
df.iloc[1:4]
```

	Duration	Date	Pulse	Maxpulse	Calories
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4

This code uses integer-based indexing to select rows 1 to 3 and columns 0 to 2, returning a specific subset of the DataFrame.

```
df.iloc[1:4,0:3]
```

	Duration	Date	Pulse
1	60	'2020/12/02'	117
2	60	'2020/12/03'	103
3	45	'2020/12/04'	109

This code selects the rows at indices 0, 8, and 10 from the DataFrame using integer-based indexing, retrieving those specific records.

df.iloc[[0,8,10]]

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
8	30	'2020/12/09'	109	133	195.1
10	60	'2020/12/11'	103	147	329.3

This code generates descriptive statistics for the numerical columns of the DataFrame, including measures such as mean, standard deviation, minimum, and maximum values.

df.describe()

	Duration	Pulse	Maxpulse	Calories
count	32.000000	32.000000	32.000000	30.000000
mean	68.437500	103.500000	128.500000	304.680000
std	70.039591	7.832933	12.998759	66.003779
min	30.000000	90.000000	101.000000	195.100000
25%	60.000000	100.000000	120.000000	250.700000
50%	60.000000	102.500000	127.500000	291.200000
75%	60.000000	106.500000	132.250000	343.975000
max	450.000000	130.000000	175.000000	479.000000

This code removes the Date column from the DataFrame and then calculates the mean of each remaining column across all rows.

```
df2=df.drop("Date",axis=1)
df2.mean(0)
Duration     68.4375
Pulse       103.5000
Maxpulse    128.5000
Calories    304.6800
dtype: float64
```

This code defines a lambda function to check whether the Duration is at least 60 minutes, applies it to create a new Boolean column, and then displays the first five rows of the updated DataFrame.

```
durationfunc=lambda x: x>=60
df2["Duration_mins_above_60"]=df["Duration"].apply(durationfunc)
df2.head()
```

	Duration	Pulse	Maxpulse	Calories	Duration_mins_above_60
0	60	110	130	409.1	True
1	60	117	145	479.0	True
2	60	103	135	340.0	True
3	45	109	175	282.4	False
4	45	117	148	406.0	False

Data Visualization

This code imports the required libraries for data analysis and visualization, and then loads the CSV file dataset_3.csv into a pandas DataFrame for further processing.

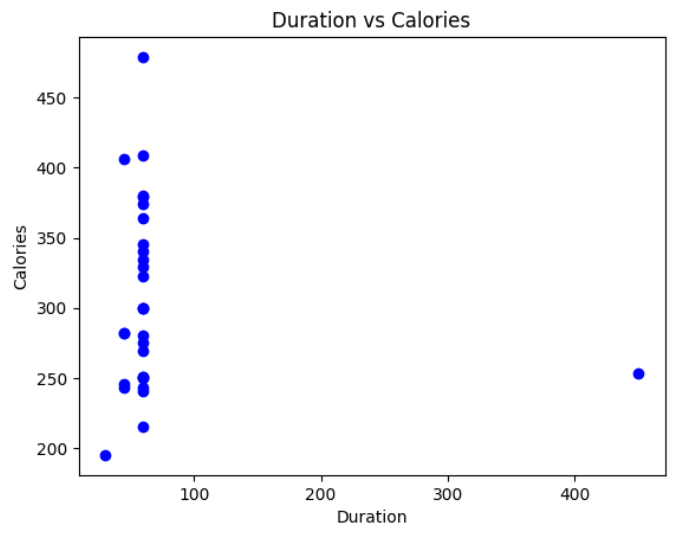
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("dataset_3.csv")
```

This code displays a summary of the DataFrame, showing the total number of records, column names, data types, and the count of non-null values in each column.

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  32 non-null     int64
 1   Date      31 non-null     object
 2   Pulse     32 non-null     int64
 3   Maxpulse  32 non-null     int64
 4   Calories  30 non-null     float64
dtypes: float64(1), int64(3), object(1)
memory usage: 1.4+ KB
```

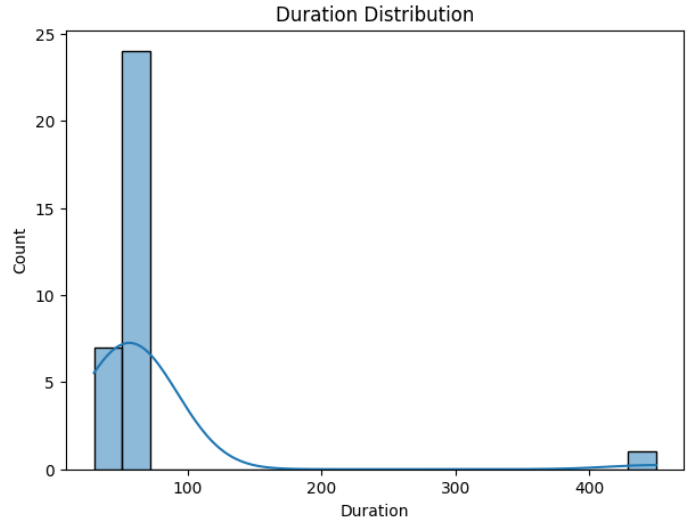
This code creates a scatter plot of Duration versus Calories, labels the axes, adds a title, and displays the visualization to show the relationship between exercise duration and calories burned.

```
plt.plot(df['Duration'], df['Calories'], 'bo')
plt.xlabel('Duration')
plt.ylabel('Calories')
plt.title('Duration vs Calories')
plt.show()
```



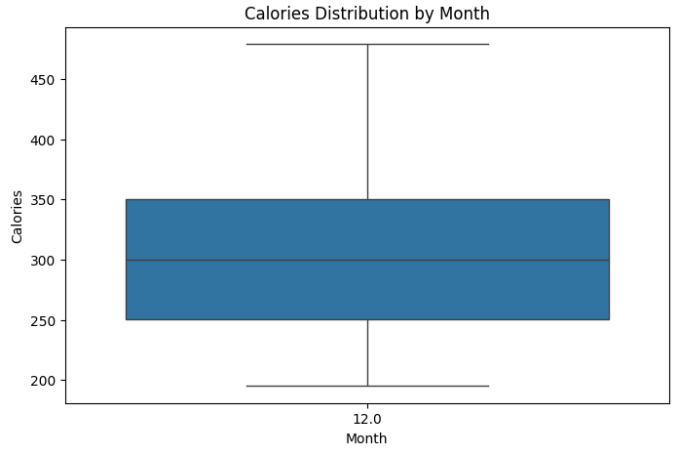
This code creates a histogram of the Duration values with 20 bins and a KDE curve, labels the axes, sets the title, and displays the plot to show the distribution of duration data.

```
plt.figure(figsize=(7,5))
sns.histplot(df['Duration'], bins=20, kde=True)
plt.xlabel("Duration")
plt.ylabel("Count")
plt.title("Duration Distribution")
plt.show()
```



This code converts the Date column to datetime format, extracts the month into a new column, and then creates a boxplot to visualize how Calories are distributed across different months.

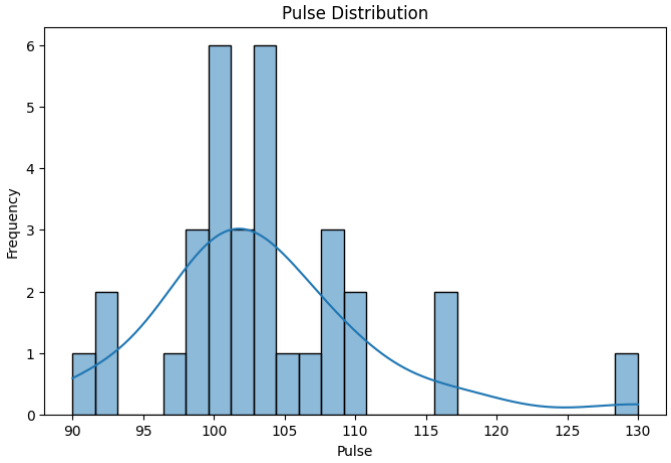
```
df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
df['Month'] = df['Date'].dt.month
plt.figure(figsize=(8,5))
sns.boxplot(x='Month', y='Calories', data=df)
plt.xlabel("Month")
plt.ylabel("Calories")
plt.title("Calories Distribution by Month")
plt.show()
```



This code generates a histogram of the Pulse values with 25 bins and a KDE curve, labels the axes, adds a title, and displays the plot to show the distribution of pulse rates.

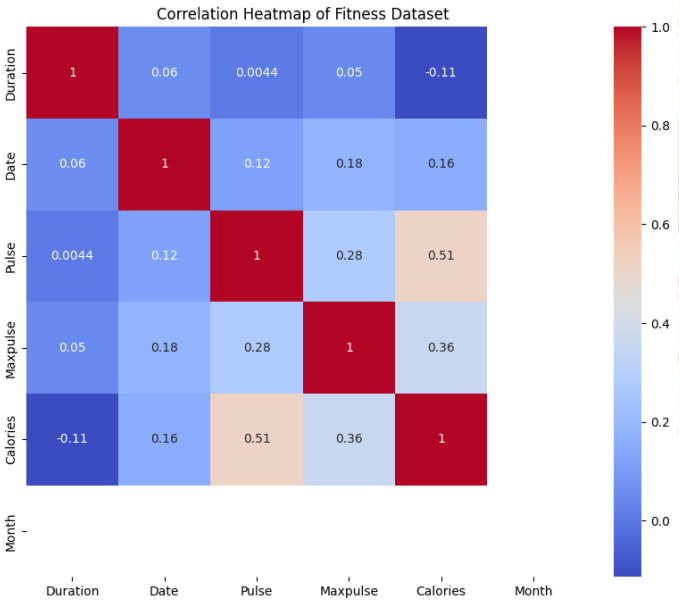
```
plt.figure(figsize=(8,5))
```

```
sns.histplot(df['Pulse'], bins=25, kde=True)
plt.xlabel("Pulse")
plt.ylabel("Frequency")
plt.title("Pulse Distribution")
plt.show()
```



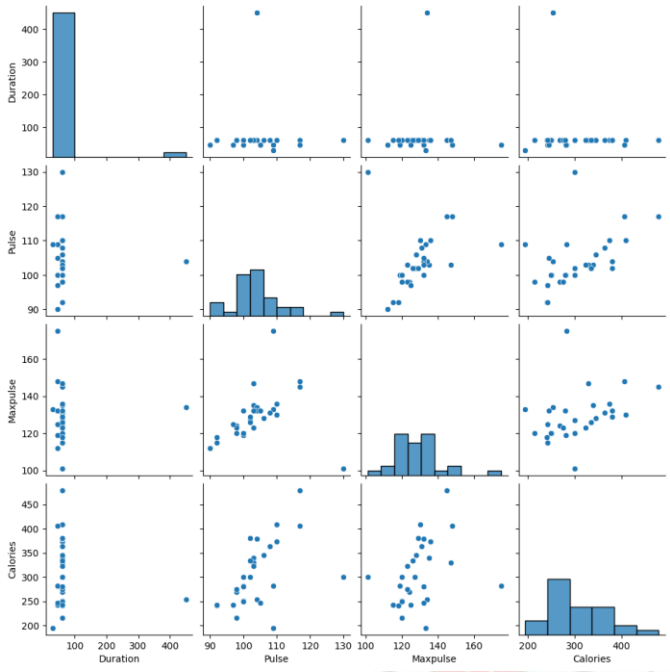
This code computes the correlation matrix of the dataset and visualizes it using a heatmap with annotations, making it easy to understand the relationships between numerical variables.

```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap of Fitness Dataset")
plt.show()
```



This code creates pairwise scatter plots and distributions for Duration, Pulse, Maxpulse, and Calories, helping to visually analyze relationships and patterns among these variables.

```
sns.pairplot(df[['Duration', 'Pulse', 'Maxpulse', 'Calories']])
plt.show()
```



Data Preprocessing

This code imports the pandas library, loads the CSV file dataset_3.csv into a DataFrame, and then displays a summary of the dataset including column types and non-null counts.

```
import pandas as pd
df=pd.read_csv('dataset_3.csv')
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  32 non-null     int64
 1   Date      31 non-null     object
 2   Pulse     32 non-null     int64
 3   Maxpulse  32 non-null     int64
 4   Calories  30 non-null     float64
dtypes: float64(1), int64(3), object(1)
memory usage: 1.4+ KB
```

This code generates descriptive statistics for the numerical columns in the DataFrame, summarizing their central tendency and spread.

```
df.describe()
```

	Duration	**Pulse**	**Maxpulse**	**Calories**
count	32.000000	32.000000	32.000000	30.000000
mean	68.437500	103.500000	128.500000	304.680000
std	70.039591	7.832933	12.998759	66.003779

	Duration	Pulse	Maxpulse	Calories
min	30.000000	90.000000	101.000000	195.100000
25%	60.000000	100.000000	120.000000	250.700000
50%	60.000000	102.500000	127.500000	291.200000
75%	60.000000	106.500000	132.250000	343.975000
max	450.000000	130.000000	175.000000	479.000000

This code displays the first five rows of the DataFrame, providing a quick preview of the dataset's structure and values.

df.head()

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0

This code checks each column for missing values and returns the total count of null entries per column in the DataFrame.

```
df.isnull().sum()
Duration    0
Date        1
Pulse       0
Maxpulse    0
Calories    2
dtype: int64
```

This code calculates the mean of the Calories column, replaces missing values in that column with the mean, and then verifies that no null values remain.

```
calories_mean = df['Calories'].mean()
df['Calories'].fillna(calories_mean, inplace=True)
df.isnull().sum()
Duration    0
Date        1
Pulse       0
Maxpulse    0
Calories    0
dtype: int64
```

This code sorts the DataFrame by the Date column, resets the index, and then filters the rows where the Date value is missing to identify invalid or unparsed dates.

```
df = df.sort_values(by='Date').reset_index(drop=True)
df[df['Date'].isna()]
```

	Duration	Date	Pulse	Maxpulse	Calories
31	45	NaN	100	119	282.0

This code fills missing values in the Date column using forward fill followed by backward fill, ensuring continuity of dates. It then checks the DataFrame to confirm that no missing values remain.

```
df['Date'] = df['Date'].ffill().bfill()
df.isna().sum()
Duration    0
Date        0
Pulse       0
Maxpulse    0
Calories    0
dtype: int64
```

This code calculates the most frequent Duration value, computes the interquartile range to determine an upper bound for outliers, and then filters the dataset to identify records with unusually high duration values.

```
duration_mode = df['Duration'].mode()[0]
duration_mode
Q1 = df['Duration'].quantile(0.25)
Q3 = df['Duration'].quantile(0.75)
IQR = Q3 - Q1
upper_bound = Q3 + 1.5 * IQR
upper_bound
high_duration_outliers = df[
    df['Duration'] > upper_bound
]
high_duration_outliers
```

	Duration	Date	Pulse	Maxpulse	Calories
7	450	'2020/12/08'	104	134	253.3

This code replaces Duration values that exceed the calculated upper bound with the mode of the Duration column, effectively treating high outliers.

```
df.loc[df['Duration'] > upper_bound, 'Duration'] = duration_mode
```

This code removes duplicate rows from the DataFrame in place, ensuring that each record in the dataset is unique.

```
df.drop_duplicates(inplace=True)
```

Scaling is a data preprocessing technique that adjusts the range of numerical features so they are on a similar scale. It prevents features with larger values from dominating the learning process and improves the performance and convergence of many machine learning algorithms, especially distance-based and gradient-based models.

Standard scaling (Z-score normalization) rescales numerical features so that they have a mean of 0 and a standard deviation of 1. This helps algorithms treat all features equally and improves model stability, especially for methods sensitive to feature magnitude.

This code creates a StandardScaler, makes a copy of the dataset, and applies standard scaling to the selected numerical columns. The transformed values replace the original ones, and the first few rows of the scaled DataFrame are displayed.

```
from sklearn.preprocessing import StandardScaler
standard_scaler = StandardScaler()
df_standard = df.copy()
df_standard[['Duration','Pulse','Maxpulse','Calories']] = (
   standard_scaler.fit_transform(df_standard[['Duration','Pulse','Maxpulse','Calories']])
)
df_standard.head()
```

	Duration	Date	Pulse	Maxpulse	Calories
0	0.510061	'2020/12/01'	0.818140	0.094980	1.627893
1	0.510061	'2020/12/02'	1.714788	1.257241	2.736104
2	0.510061	'2020/12/03'	-0.078508	0.482401	0.532365
3	-1.466426	'2020/12/04'	0.690047	3.581761	-0.380839
4	-1.466426	'2020/12/05'	1.714788	1.489693	1.578745

Min–Max scaling rescales numerical features to a fixed range, usually between 0 and 1. It preserves the original distribution shape while ensuring all values lie within the same bounded interval.
This code initializes a MinMaxScaler, creates a copy of the DataFrame, and applies min–max scaling to the selected numerical columns. The scaled values replace the originals, and the first few rows of the transformed dataset are displayed.

```
from sklearn.preprocessing import MinMaxScaler
minmax_scaler = MinMaxScaler()
df_minmax = df.copy()
df_minmax[['Duration','Pulse','Maxpulse','Calories']] = (
   minmax_scaler.fit_transform(df_minmax[['Duration','Pulse','Maxpulse','Calories']])
)
df_minmax.head()
```

	Duration	Date	Pulse	Maxpulse	Calories
0	1.0	'2020/12/01'	0.500	0.391892	0.753787
1	1.0	'2020/12/02'	0.675	0.594595	1.000000
2	1.0	'2020/12/03'	0.325	0.459459	0.510391
3	0.5	'2020/12/04'	0.475	1.000000	0.307503
4	0.5	'2020/12/05'	0.675	0.635135	0.742867

Robust scaling scales features using the median and interquartile range (IQR), making it less sensitive to outliers compared to standard and min–max scaling.
This code creates a RobustScaler, copies the dataset, and applies robust scaling to the selected numerical columns. The transformed values replace the original ones, and the first few rows of the scaled DataFrame are displayed.

```
from sklearn.preprocessing import RobustScaler
```

```
robust_scaler = RobustScaler()
df_robust = df.copy()
df_robust[['Duration','Pulse','Maxpulse','Calories']] = (
   robust_scaler.fit_transform(df_robust[['Duration','Pulse','Maxpulse','Calories']])
)
df_robust.head()
```

	Duration	Date	Pulse	Maxpulse	Calories
0	0.0	'2020/12/01'	1.000000	0.16	1.203530
1	0.0	'2020/12/02'	2.000000	1.36	1.974628
2	0.0	'2020/12/03'	0.000000	0.56	0.441258
3	-15.0	'2020/12/04'	0.857143	3.76	-0.194153
4	-15.0	'2020/12/05'	2.000000	1.60	1.169333

MaxAbs scaling scales features by dividing each value by the maximum absolute value of that feature, resulting in values within the range −1 to 1. It preserves sparsity and does not shift the data's center.

This code initializes a MaxAbsScaler, creates a copy of the DataFrame, and applies max-absolute scaling to the selected numerical columns. The scaled values replace the original ones, and the first few rows of the transformed dataset are displayed.

```
from sklearn.preprocessing import MaxAbsScaler
maxabs_scaler = MaxAbsScaler()
df_maxabs = df.copy()
df_maxabs[['Duration','Pulse','Maxpulse','Calories']] = (
   maxabs_scaler.fit_transform(df_maxabs[['Duration','Pulse','Maxpulse','Calories']])
)
df_maxabs.head()
```

	Duration	Date	Pulse	Maxpulse	Calories
0	1.00	'2020/12/01'	0.846154	0.742857	0.854071
1	1.00	'2020/12/02'	0.900000	0.828571	1.000000
2	1.00	'2020/12/03'	0.792308	0.771429	0.709812
3	0.75	'2020/12/04'	0.838462	1.000000	0.589562
4	0.75	'2020/12/05'	0.900000	0.845714	0.847599

Encoding is a data preprocessing technique used to convert categorical or non-numeric data into numerical form so that it can be understood by machine learning algorithms. Most ML models work only with numbers, so encoding ensures that categorical information (such as dates, labels, or categories) is represented in a meaningful numeric format without losing essential patterns. The choice of encoding method depends on whether the categories have an order, are purely nominal, or represent structured information like time.

This code applies label encoding to the Date column by converting each unique date into a unique integer value. A copy of the dataset is used to preserve the original data, and the first few rows are displayed to show the original dates alongside their encoded numerical labels.

```
from sklearn.preprocessing import LabelEncoder
df_label = df.copy()
le = LabelEncoder()
df_label['Date_encoded'] = le.fit_transform(df_label['Date'].astype(str))
df_label[['Date', 'Date_encoded']].head()
```

	Date	Date_encoded
0	'2020/12/01'	0
1	'2020/12/02'	1
2	'2020/12/03'	2
3	'2020/12/04'	3
4	'2020/12/05'	4

This code performs one-hot encoding on the Date column by creating a separate binary column for each unique date. The encoded features are converted into a DataFrame, combined with the original numerical columns, and the first few rows of the fully encoded dataset are displayed.

```
from sklearn.preprocessing import OneHotEncoder
df_ohe = df.copy()
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
encoded = ohe.fit_transform(df_ohe[['Date']].astype(str))
encoded_df = pd.DataFrame(encoded, columns=ohe.get_feature_names_out(['Date']))
df_ohe_encoded = pd.concat([df_ohe.drop(columns=['Date']), encoded_df], axis=1)
df_ohe_encoded.iloc[:, -5:].head()  # Displaying last 5 columns for brevity
```

	Date_'2020/12/28'	Date_'2020/12/29'	Date_'2020/12/30'	Date_'2020/12/31'	Date_20201226
0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0

References-

https://www.kaggle.com/datasets/themrityunjaypathak/pandas-practice-dataset

Experiment – 4

Aim – To implement various Feature Selection Techniques and Train Models based on the selected features.

Logistic Regression Model:

Import data and data preprocessing –

```
import pandas as pd
df=pd.read_csv('breast-cancer.csv')
X=df.drop('diagnosis',axis=1)
Y=df['diagnosis']
from sklearn.preprocessing import LabelEncoder
X = X.drop(columns=['Unnamed: 32'])
le = LabelEncoder()
Y = le.fit_transform(Y)
```

Import packages –

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif, chi2, mutual_info_classif
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import roc_curve, roc_auc_score
```

F-classif (ANOVA F-test) –

F-Classif is a univariate feature selection method used in classification problems and is based on the ANOVA (Analysis of Variance) F-test. It evaluates each feature independently by measuring how strongly it is related to the target class. The method works by comparing the variation of feature values between different classes with the variation within the same class. Features that show large differences in mean values across classes receive higher F-scores and are considered more relevant for classification tasks.

The F-statistic is calculated as the ratio of between-class variance to within-class variance, and a corresponding p-value indicates the statistical significance of each feature. A high F-score and a low p-value suggest that the feature has a strong influence on class separation. F-Classif works best with numerical features and categorical targets, does not require feature scaling, and is computationally efficient. However, since it evaluates features individually, it does not capture interactions or non-linear relationships between features.x

```
# 1. Split first
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.3, random_state=42
)
# 2. Feature selection
selector = SelectKBest(score_func=f_classif, k=7)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)
# # Get the selected feature columns (boolean mask)
selected_columns = X.columns[selector.get_support()]
```

```
# # Print the names of the selected features
print("Selected Features:", selected_columns)
Selected Features: Index(['radius_mean', 'perimeter_mean', 'concave points_mean', 'radius_worst',
       'perimeter_worst', 'area_worst', 'concave points_worst'],
      dtype='object')
```

 # 3. Scaling

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_selected)
X_test_scaled = scaler.transform(X_test_selected)
```

 # 4. Train model

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
y_pred
array([0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0])
```

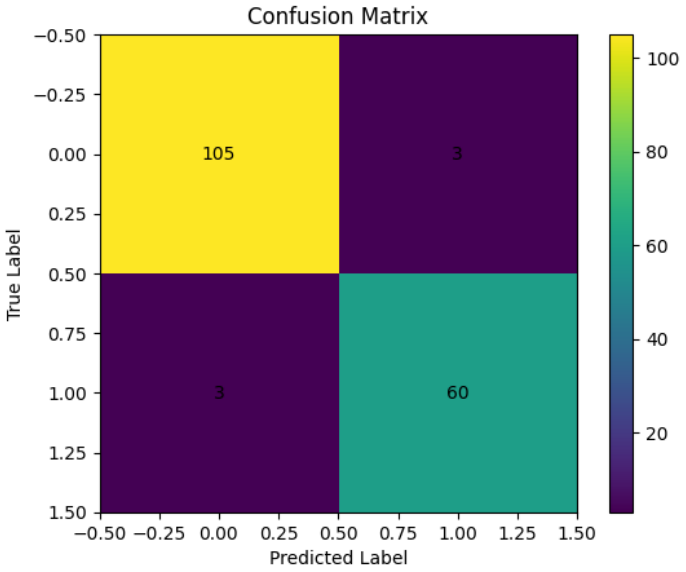
#5. Metrics

```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(f'Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
Accuracy: 0.9649122807017544
Confusion Matrix:
[[105   3]
 [  3  60]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       108
           1       0.95      0.95      0.95        63

    accuracy                           0.96       171
   macro avg       0.96      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171
```

#Confusion Matrix

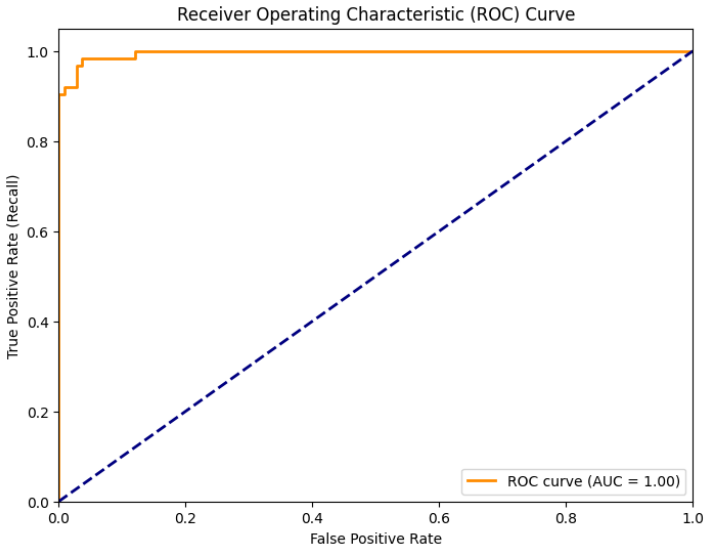
```
cm = confusion_matrix(y_test, y_pred)
plt.figure()
plt.imshow(cm)
```

```
plt.title("Confusion Matrix")
plt.colorbar()
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
# Show values inside cells
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, cm[i, j], ha="center", va="center")
plt.show()
```



 # ROC curve

```
y_prob = model.predict_proba(X_test_scaled)[:, 1]
# Compute the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
# Calculate the AUC (Area Under the Curve)
roc_auc = roc_auc_score(y_test, y_prob)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  # Diagonal line (random classifier)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate (Recall)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
print(f"AUC (Area Under the Curve): {roc_auc:.2f}")
```



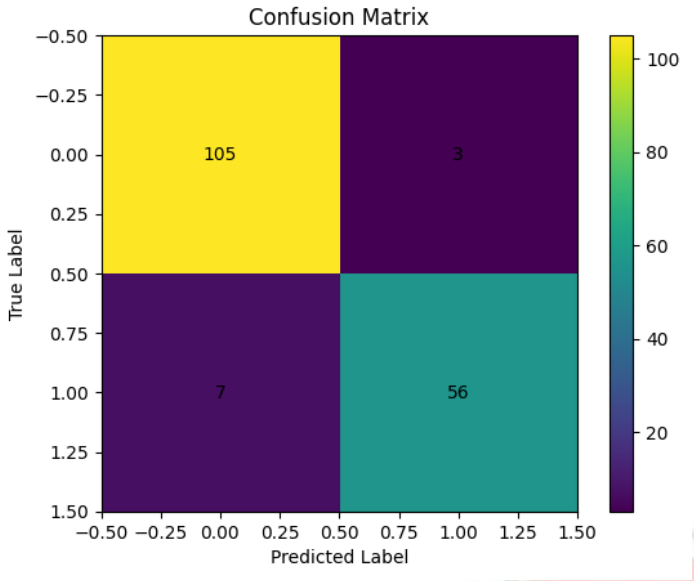
Chi2 Test

The Chi-square (χ^2) test is a statistical feature selection method used in classification problems to measure the dependence between each input feature and the target class. It evaluates whether the observed frequency distribution of a feature is significantly different across different class labels. If a feature and the target variable are independent, the Chi-square value will be low; a high Chi-square value indicates that the feature is strongly associated with the class and is therefore important for classification.

The Chi-square test works only with non-negative feature values, which is why scaling methods like Min–Max scaling are commonly applied before using it. It is especially suitable for categorical data or discretized numerical features and is often used in text classification and count-based datasets. While the Chi-square test is simple and computationally efficient, it evaluates each feature independently and cannot capture interactions between features or handle negative values directly.

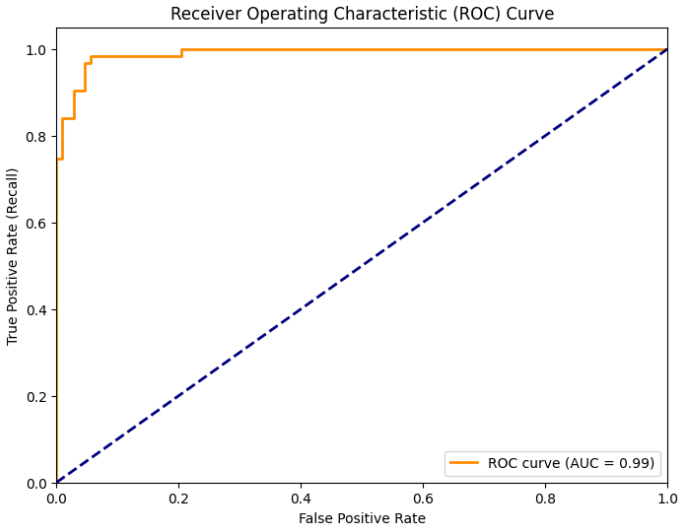
```
# 1. Split first
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.3, random_state=42
)
# 2. Scaling
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Select
selector = SelectKBest(score_func=chi2, k=7)
X_train_selected = selector.fit_transform(X_train_scaled, y_train)
X_test_selected = selector.transform(X_test_scaled)
# 3. Feature names
selected_columns = X_train.columns[selector.get_support()]
print("Selected Features:", selected_columns)
Selected Features: Index(['concavity_mean', 'concave points_mean', 'radius_worst',
       'perimeter_worst', 'area_worst', 'concavity_worst',
```

```
       'concave points_worst'],
      dtype='object')
# 4. Train model
model = LogisticRegression(max_iter=1000)
model.fit(X_train_selected, y_train)
y_pred = model.predict(X_test_selected)
#5. Metrics
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
Accuracy: 0.9415204678362573
Confusion Matrix:
[[105   3]
 [  7  56]]
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.95       108
           1       0.95      0.89      0.92        63

    accuracy                           0.94       171
   macro avg       0.94      0.93      0.94       171
weighted avg       0.94      0.94      0.94       171
#Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure()
plt.imshow(cm)
plt.title("Confusion Matrix")
plt.colorbar()
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
# Show values inside cells
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, cm[i, j], ha="center", va="center")
plt.show()
```



 # ROC curve

```
y_prob = model.predict_proba(X_test_selected)[:, 1]
# Compute the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
# Calculate the AUC (Area Under the Curve)
roc_auc = roc_auc_score(y_test, y_prob)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  # Diagonal line (random classifier)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate (Recall)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
print(f"AUC (Area Under the Curve): {roc_auc:.2f}")
```



Mutual_info_classif –

Mutual Information for Classification (mutual_info_classif) is a feature selection method that measures the amount of information a feature provides about the target class. It is based on information theory and quantifies how much knowing the value of a feature reduces the uncertainty of the class label. If a feature contains no useful information about the target, its mutual information score will be close to zero; higher values indicate stronger dependency between the feature and the class.

Unlike statistical tests such as F-test or Chi-square, mutual_info_classif can capture both linear and non-linear relationships between features and the target variable. It does not require feature scaling and works with continuous as well as discrete features. However, since it estimates probabilities from data, it can be computationally more expensive and may be sensitive to sample size. Like other univariate methods, it evaluates features independently and does not account for interactions between multiple features.

```
# 1. Train–test split
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.3, random_state=42
)
# 2. Feature selection (NO scaling here)
selector = SelectKBest(score_func=mutual_info_classif, k=7)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)
selected_columns = X_train.columns[selector.get_support()]
print("Selected Features:", selected_columns)
# 3. Scaling (ONLY because Logistic Regression needs it)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_selected)
X_test_scaled = scaler.transform(X_test_selected)
Selected Features: Index(['perimeter_mean', 'concavity_mean', 'concave points_mean',
       'radius_worst', 'perimeter_worst', 'area_worst',
       'concave points_worst'],
      dtype='object')
```

#4. Train model

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
```

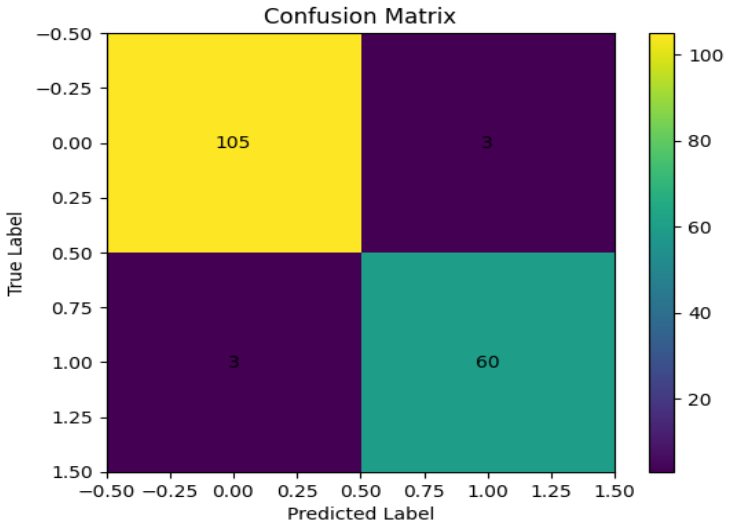
#5. Metrics

```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
```

```
Accuracy: 0.9649122807017544
Confusion Matrix:
[[105   3]
 [  3  60]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       108
           1       0.95      0.95      0.95        63

    accuracy                           0.96       171
   macro avg       0.96      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171
```

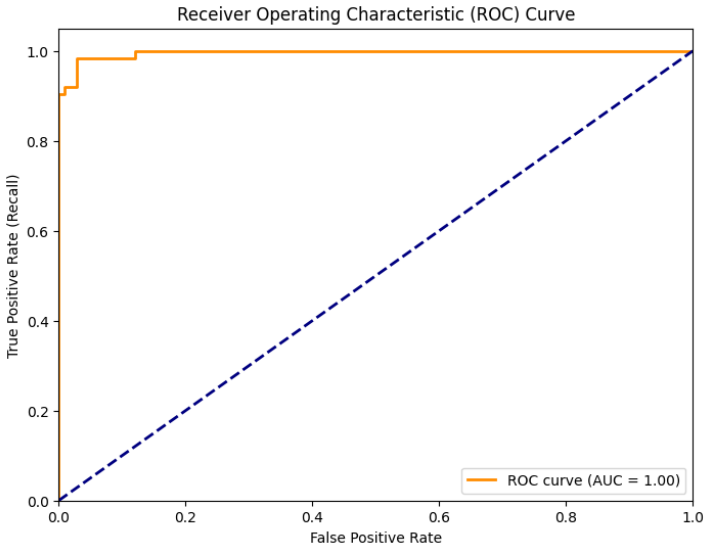
#Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
plt.figure()
plt.imshow(cm)
plt.title("Confusion Matrix")
plt.colorbar()
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
# Show values inside cells
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, cm[i, j], ha="center", va="center")
plt.show()
```



 # ROC curve

```
y_prob = model.predict_proba(X_test_scaled)[:, 1]
# Compute the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
# Calculate the AUC (Area Under the Curve)
roc_auc = roc_auc_score(y_test, y_prob)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  # Diagonal line (random classifier)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate (Recall)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
print(f"AUC (Area Under the Curve): {roc_auc:.2f}")
```



Decision Tree Model:

Import data and data preprocessing –

```
import pandas as pd
df=pd.read_csv('breast-cancer.csv')
X=df.drop('diagnosis',axis=1)
Y=df['diagnosis']
from sklearn.preprocessing import LabelEncoder
X = X.drop(columns=['Unnamed: 32'])
le = LabelEncoder()
Y = le.fit_transform(Y)
```

Import Packages –

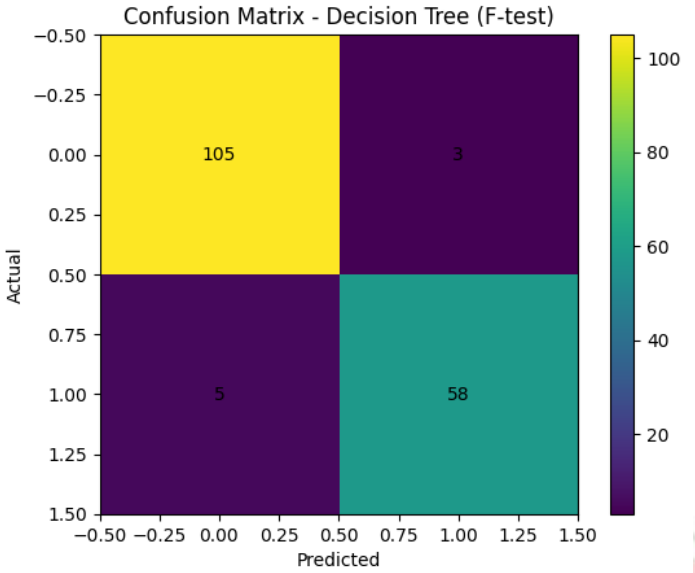
```
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif, chi2, mutual_info_classif
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import MinMaxScaler
```

F-classif (ANOVA)-

```
# 1. Split
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.3, random_state=42
)
# 2. Feature Selection
selector = SelectKBest(score_func=f_classif, k=7)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)
print("Selected Features:", X.columns[selector.get_support()])
```

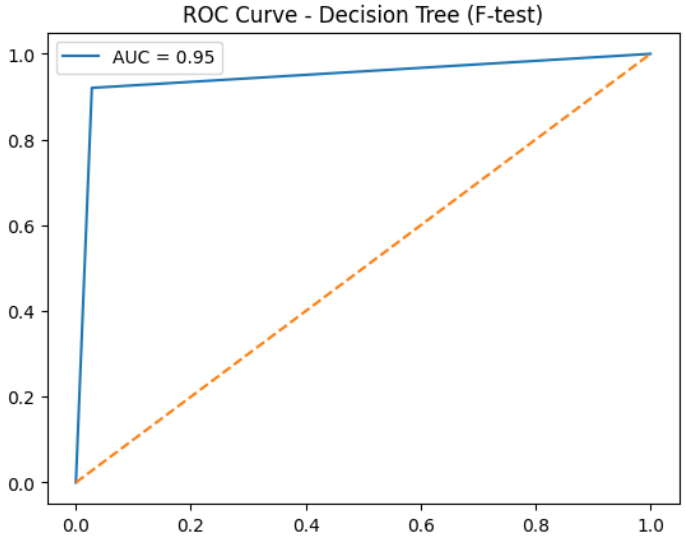
Selected Features: Index(['radius_mean', 'perimeter_mean', 'concave points_mean', 'radius_worst',

```
      'perimeter_worst', 'area_worst', 'concave points_worst'],
     dtype='object')
# 3. Train Decision Tree
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train_selected, y_train)
# 4. Prediction
y_pred = model.predict(X_test_selected)
# 5. Metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
Accuracy: 0.9532163742690059
Confusion Matrix:
 [[105   3]
 [  5  58]]
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       108
           1       0.95      0.92      0.94        63

    accuracy                           0.95       171
   macro avg       0.95      0.95      0.95       171
weighted avg       0.95      0.95      0.95       171
cm = confusion_matrix(y_test, y_pred)
plt.imshow(cm)
plt.title("Confusion Matrix - Decision Tree (F-test)")
plt.colorbar()
plt.xlabel("Predicted")
plt.ylabel("Actual")
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, cm[i, j], ha="center", va="center")
plt.show()
#Confusion Matrix
```



#ROC Curve

```
y_prob = model.predict_proba(X_test_selected)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
plt.plot([0,1],[0,1],'--')
plt.legend()
plt.title("ROC Curve - Decision Tree (F-test)")
plt.show()
```



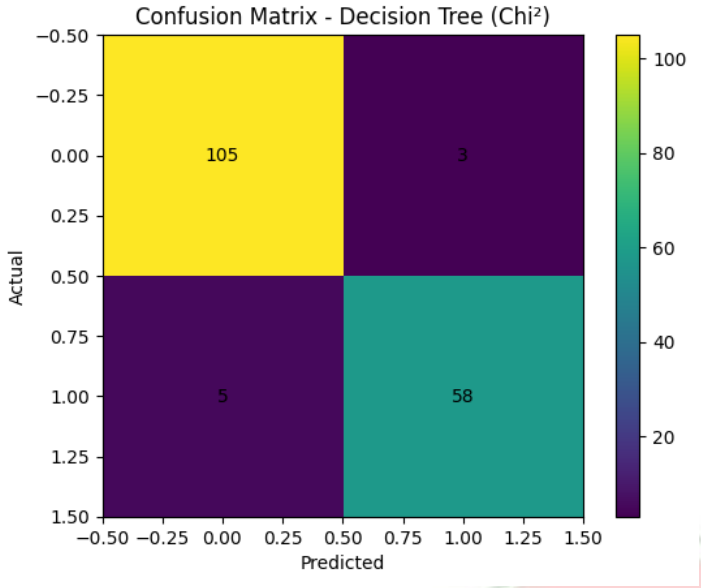
Chi2 –

 # 1. Split

```
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.3, random_state=42
)
```

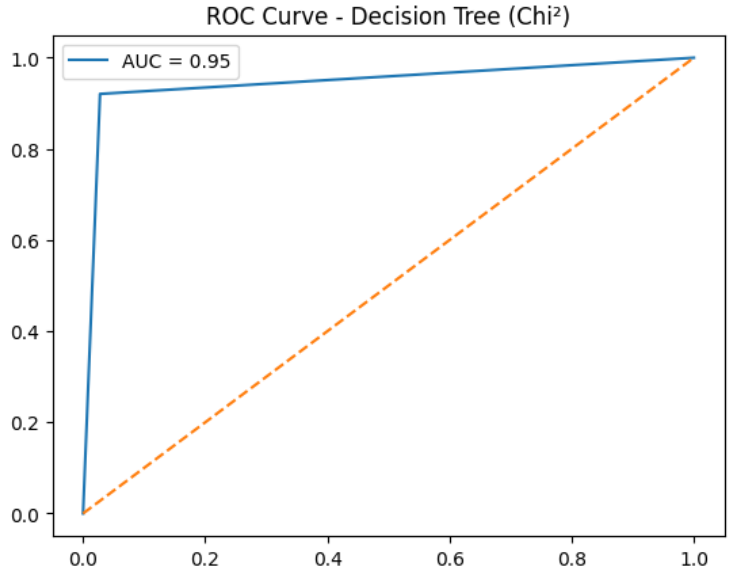
 # 2. Scaling (MANDATORY for chi2)

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# 3. Feature Selection
selector = SelectKBest(score_func=chi2, k=7)
X_train_selected = selector.fit_transform(X_train_scaled, y_train)
X_test_selected = selector.transform(X_test_scaled)
print("Selected Features:", X.columns[selector.get_support()])
Selected Features: Index(['concavity_mean', 'concave points_mean', 'radius_worst',
       'perimeter_worst', 'area_worst', 'concavity_worst',
       'concave points_worst'],
      dtype='object')

# 4. Train Decision Tree
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train_selected, y_train)
# 5. Prediction
y_pred = model.predict(X_test_selected)
# 6. Metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
Accuracy: 0.9532163742690059
Confusion Matrix:
 [[105   3]
 [  5  58]]
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       108
           1       0.95      0.92      0.94        63

    accuracy                           0.95       171
   macro avg       0.95      0.95      0.95       171
weighted avg       0.95      0.95      0.95       171
#Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.imshow(cm)
plt.title("Confusion Matrix - Decision Tree (Chi²)")
plt.colorbar()
plt.xlabel("Predicted")
plt.ylabel("Actual")
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, cm[i, j], ha="center", va="center")
plt.show()
```



#ROC Curve

```
y_prob = model.predict_proba(X_test_selected)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
plt.plot([0,1],[0,1],'--')
plt.legend()
plt.title("ROC Curve - Decision Tree (Chi²)")
plt.show()
```

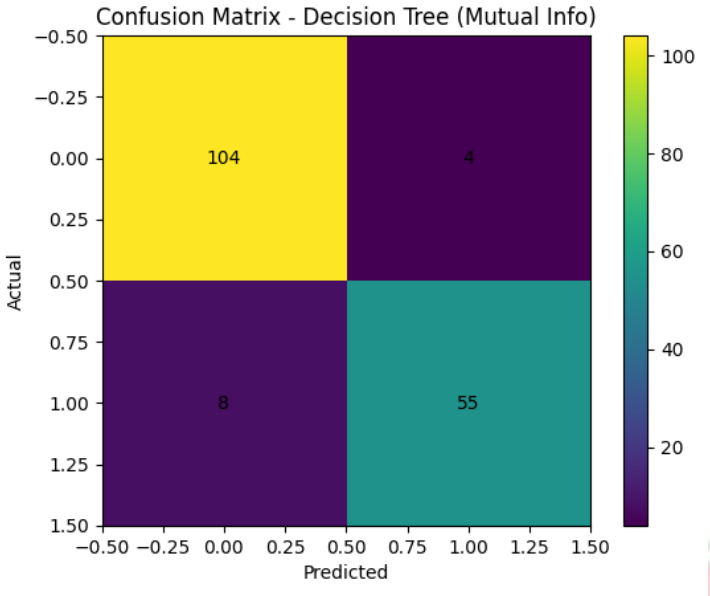


Mutual_info_classif –

 # 1. Split

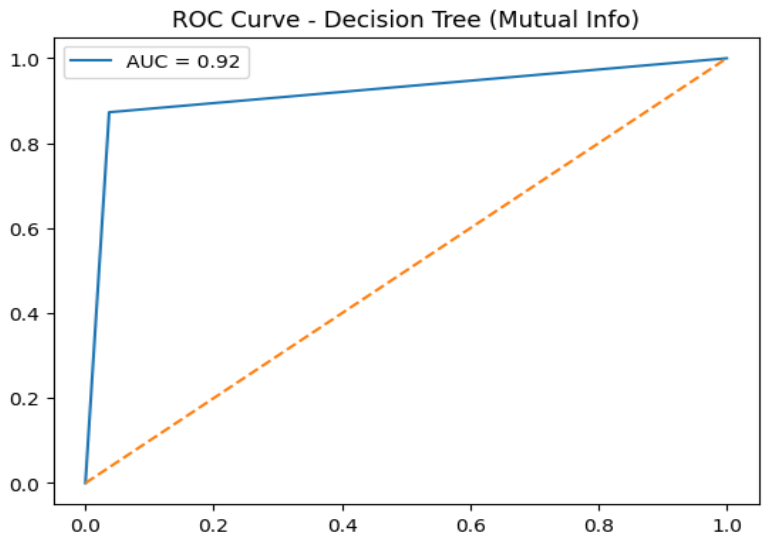
```
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.3, random_state=42
```

```
)
# 2. Feature Selection
selector = SelectKBest(score_func=mutual_info_classif, k=7)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)
print("Selected Features:", X.columns[selector.get_support()])
Selected Features: Index(['perimeter_mean', 'concavity_mean', 'concave points_mean',
       'radius_worst', 'perimeter_worst', 'area_worst',
       'concave points_worst'],
      dtype='object')

# 3. Train Decision Tree
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train_selected, y_train)
# 4. Prediction
y_pred = model.predict(X_test_selected)
# 5. Metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
Accuracy: 0.9298245614035088
Confusion Matrix:
 [[104   4]
 [  8  55]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.96      0.95       108
           1       0.93      0.87      0.90        63

    accuracy                           0.93       171
   macro avg       0.93      0.92      0.92       171
weighted avg       0.93      0.93      0.93       171
#Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.imshow(cm)
plt.title("Confusion Matrix - Decision Tree (Mutual Info)")
plt.colorbar()
plt.xlabel("Predicted")
plt.ylabel("Actual")
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, cm[i, j], ha="center", va="center")
plt.show()
```



#ROC Curve

```
y_prob = model.predict_proba(X_test_selected)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
plt.plot([0,1],[0,1],'--')
plt.legend()
plt.title("ROC Curve - Decision Tree (Mutual Info)")
plt.show()
```



Experiment – 5
ID3

Aim – A program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Theory - A decision tree is a supervised learning technique used for classification, where knowledge is represented in the form of a tree structure consisting of decision nodes, branches, and leaf nodes. Each internal node tests an attribute, each branch corresponds to an outcome of the test, and each leaf node represents a class label. The ID3 (Iterative Dichotomiser 3) algorithm, proposed by J. Ross Quinlan, builds a decision tree using a top-down, greedy approach by selecting the most informative attribute at each stage of tree construction.

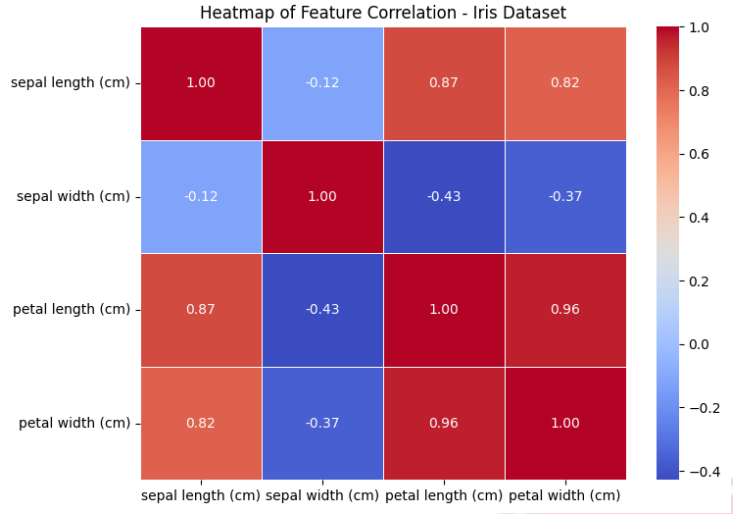
ID3 uses the concepts of entropy and information gain to select the best attribute for splitting the dataset. Entropy measures the level of uncertainty or impurity in a dataset, while information gain measures the reduction in entropy achieved by partitioning the data based on a specific attribute. The attribute with the highest information gain is chosen as the splitting attribute, ensuring that the data is classified as effectively as possible at each level of the tree.

Once the decision tree is constructed, it can be used to classify new and unseen samples by traversing the tree from the root node to a leaf node based on the attribute values of the input sample. ID3 works best with categorical attributes and produces interpretable models, but it may suffer from overfitting and does not directly handle continuous attributes without preprocessing. Despite these limitations, ID3 is widely used for understanding the fundamental principles of decision tree learning.

Code –

```
#Load Dataset
import pandas as pd
from sklearn.datasets import load_iris
data=load_iris()
df=pd.DataFrame(load_iris().data,columns=load_iris().feature_names)
x=df
y=load_iris().target
#import libraries
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
corr = df.corr()

# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(
    corr,
    annot=True,
    cmap="coolwarm",
    linewidths=0.5,
    fmt=".2f"
)
```

```
plt.title("Heatmap of Feature Correlation - Iris Dataset")
plt.show()
```



#Split data

```
X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.3, random_state=42
)
```

#Model

```
model = DecisionTreeClassifier(criterion='entropy',random_state=42)
model.fit(X_train, y_train)
```

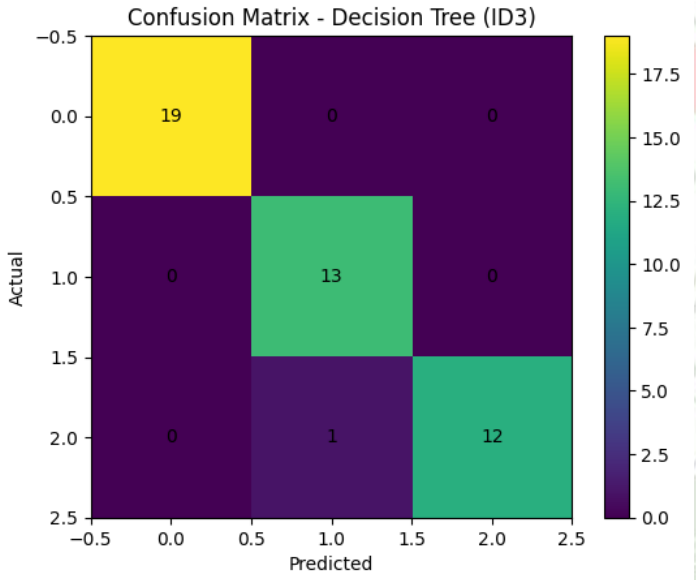
#Predict

```
y_pred = model.predict(X_test)
y_pred
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 1, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 0, 0, 2, 1, 0, 0, 0, 2, 1, 1, 0,
       0])
```

#Metrics

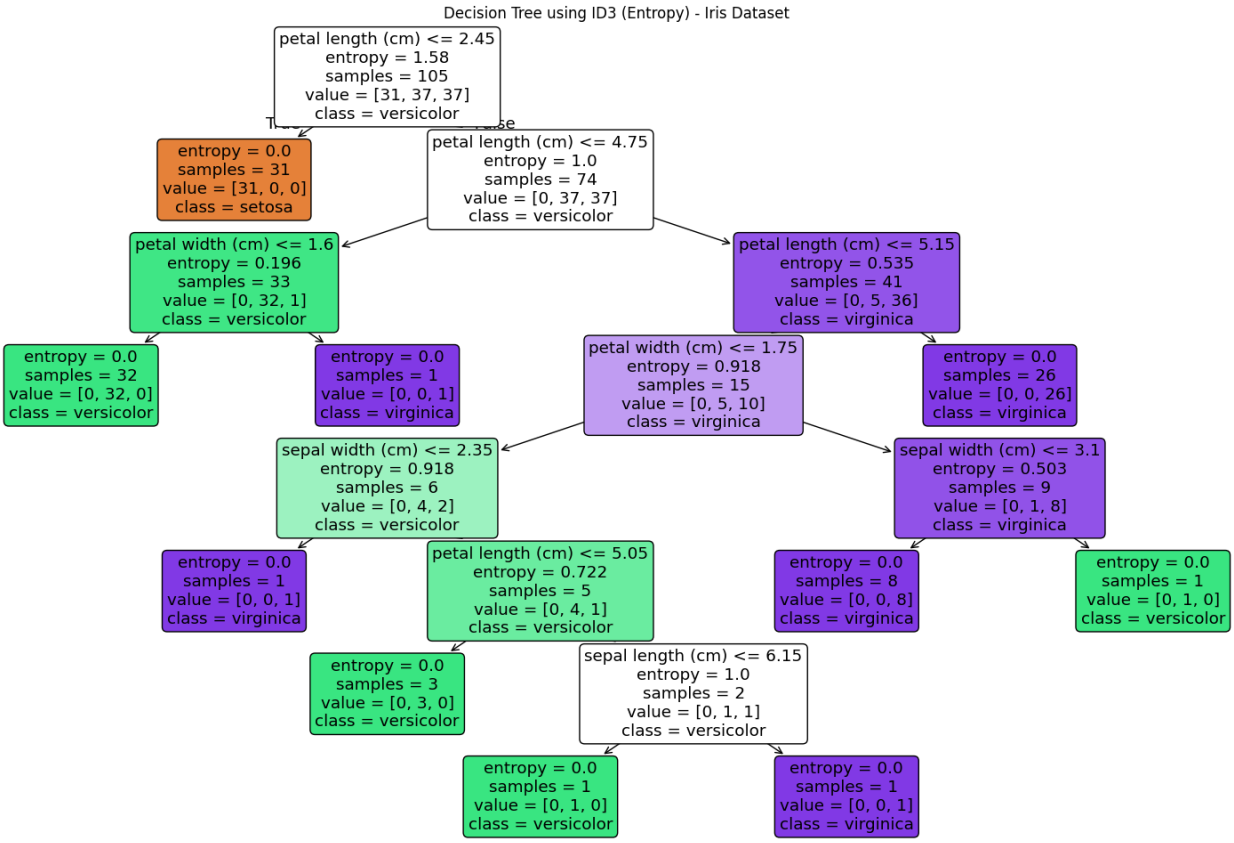
```
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
Accuracy: 0.9777777777777777
Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  1 12]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.93      1.00      0.96        13
           2       1.00      0.92      0.96        13

    accuracy                           0.98        45
```

```
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45
cm = confusion_matrix(y_test, y_pred)

plt.imshow(cm)
plt.title("Confusion Matrix - Decision Tree (ID3)")
plt.colorbar()
plt.xlabel("Predicted")
plt.ylabel("Actual")

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, cm[i, j], ha="center", va="center")
plt.show()
```



#**Plot Decision tree**

```
from sklearn.tree import DecisionTreeClassifier, plot_tree

plt.figure(figsize=(20, 12))
plot_tree(
    model,
    feature_names=data.feature_names,
    class_names=data.target_names,
    filled=True,
    rounded=True
)
plt.title("Decision Tree using ID3 (Entropy) - Iris Dataset")
plt.show()
```



K-Nearest

Aim - Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

Theory - The k-Nearest Neighbour (k-NN) algorithm is a supervised machine learning technique used for classification and regression. It is a lazy learning algorithm, meaning it does not build an explicit model during training. Instead, all training instances are stored, and classification is performed only when a new sample needs to be classified. The algorithm works by finding the k closest data points (nearest neighbours) to the new sample based on a distance metric such as Euclidean distance.

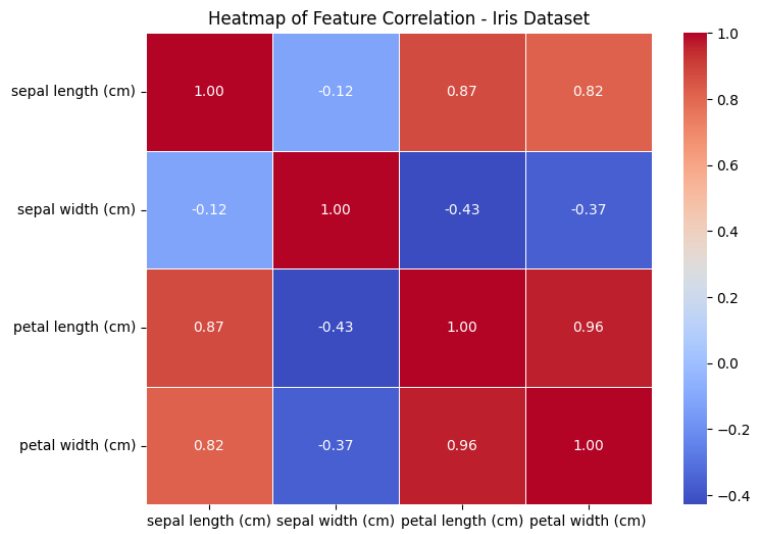
In classification problems like the Iris dataset, k-NN determines the class of a test sample by identifying the k nearest training samples and assigning the class that occurs most frequently among them (majority voting). The value of k plays an important role: a small value of k may make the model sensitive to noise, while a larger value provides smoother decision boundaries but may reduce accuracy. Feature scaling is often applied before using k-NN, as distance calculations are affected by the magnitude of feature values.

After classification, the predicted class labels are compared with the actual class labels to evaluate performance. Predictions that match the true class are considered correct, while mismatches are considered wrong predictions. The k-NN algorithm is simple, intuitive, and effective for small datasets, but it can be computationally expensive for large datasets and sensitive to irrelevant features.

Code –

```
import pandas as pd
```

```
from sklearn.datasets import load_iris
data=load_iris()
df=pd.DataFrame(load_iris().data,columns=load_iris().feature_names)
x=df
y=load_iris().target
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
corr = df.corr()

# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(
    corr,
    annot=True,
    cmap="coolwarm",
    linewidths=0.5,
    fmt=".2f"
)
plt.title("Heatmap of Feature Correlation - Iris Dataset")
plt.show()
```




```
X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.3, random_state=42
)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Correct Predictions:")
```

```
for i in range(len(y_test)):
    if y_test[i] == y_pred[i]:
        print(
            f"Actual: {data.target_names[y_test[i]]}, "
            f"Predicted: {data.target_names[y_pred[i]]}"
        )

print("\nWrong Predictions:")
for i in range(len(y_test)):
    if y_test[i] != y_pred[i]:
        print(
            f"Actual: {data.target_names[y_test[i]]}, "
            f"Predicted: {data.target_names[y_pred[i]]}"
        )
```

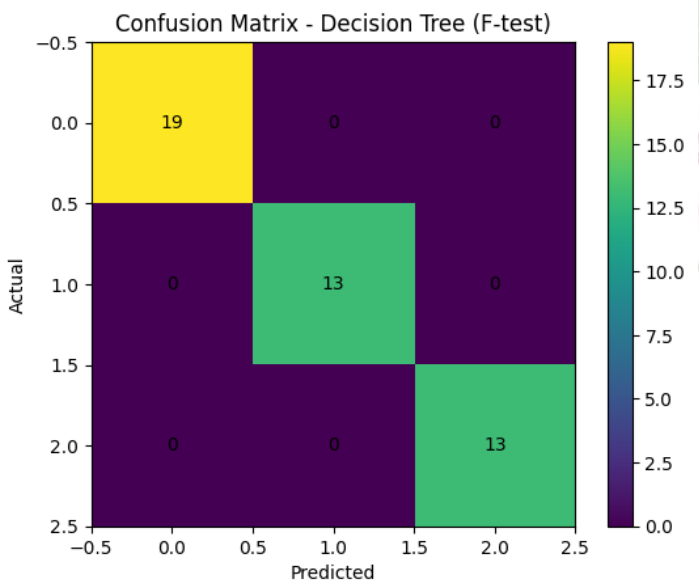
```
Correct Predictions:
Actual: versicolor, Predicted: versicolor
Actual: setosa, Predicted: setosa
Actual: virginica, Predicted: virginica
Actual: versicolor, Predicted: versicolor
Actual: versicolor, Predicted: versicolor
Actual: setosa, Predicted: setosa
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: virginica
Actual: versicolor, Predicted: versicolor
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: virginica
Actual: setosa, Predicted: setosa
Actual: setosa, Predicted: setosa
Actual: setosa, Predicted: setosa
Actual: setosa, Predicted: setosa
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: virginica
Actual: versicolor, Predicted: versicolor
Actual: versicolor, Predicted: versicolor
Actual: virginica, Predicted: virginica
Actual: setosa, Predicted: setosa
Actual: virginica, Predicted: virginica
Actual: setosa, Predicted: setosa
Actual: virginica, Predicted: virginica
...
Actual: setosa, Predicted: setosa
Actual: setosa, Predicted: setosa

Wrong Predictions:
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
print("Classification Report:\n", classification_report(y_test, y_pred))
```

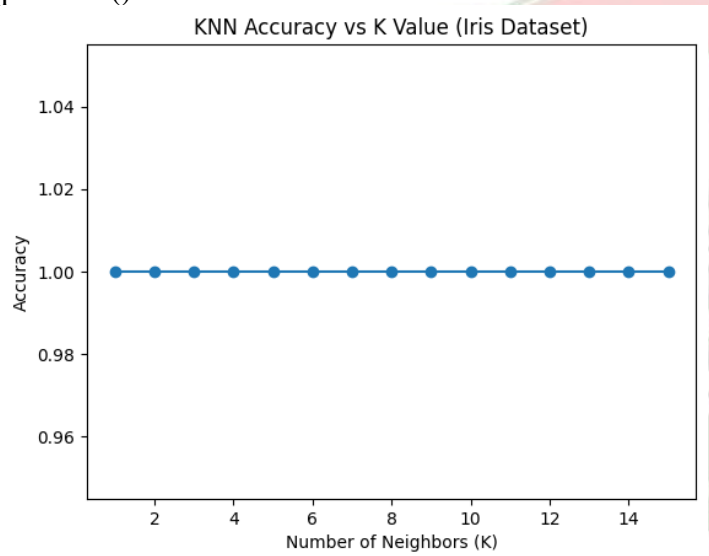
```
Accuracy: 1.0
Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

```
cm = confusion_matrix(y_test, y_pred)

plt.imshow(cm)
plt.title("Confusion Matrix - Decision Tree (F-test)")
plt.colorbar()
plt.xlabel("Predicted")
plt.ylabel("Actual")

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, cm[i, j], ha="center", va="center")
plt.show()
```




```
k_values = range(1, 16)
accuracies = []
```

```
# Train KNN for different K values
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)

# Plot graph
plt.figure()
plt.plot(k_values, accuracies, marker='o')
plt.xlabel("Number of Neighbors (K)")
plt.ylabel("Accuracy")
plt.title("KNN Accuracy vs K Value (Iris Dataset)")
plt.show()
```



Em vs K-means

Aim - Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using kMeans algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

Theory - Clustering is an unsupervised learning technique used to group similar data points without prior class labels. The Expectation–Maximization (EM) algorithm is a probabilistic clustering approach that assumes the data is generated from a mixture of probability distributions, typically Gaussian distributions. EM works iteratively in two steps: the Expectation (E) step, where the probability of each data point belonging to each cluster is calculated, and the Maximization (M) step, where the model parameters (means, variances, and mixing coefficients) are updated to maximize the likelihood of the data. This results in soft clustering, where each data point can belong to multiple clusters with different probabilities.

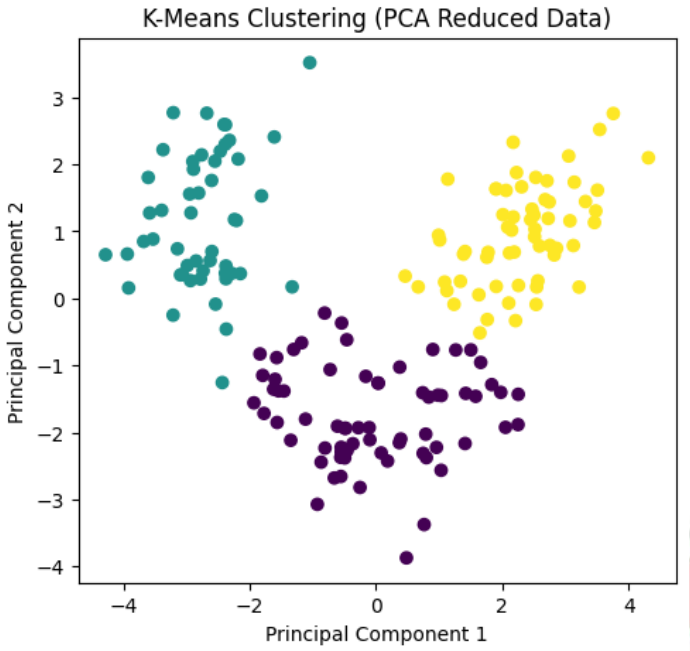
The K-Means algorithm is a distance-based clustering method that partitions the dataset into k clusters by minimizing the within-cluster sum of squared distances. It works by initially selecting cluster centroids, assigning each data point to the nearest centroid, and then updating the centroids based on the

mean of assigned points. This process is repeated until convergence. K-Means performs hard clustering, meaning each data point belongs to exactly one cluster, and it assumes clusters are spherical and equally sized.

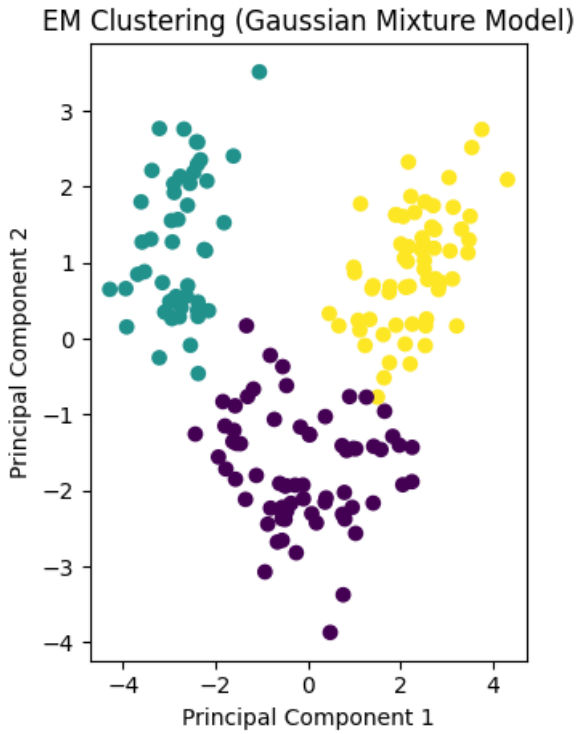
When the same dataset is clustered using both EM and K-Means, differences in clustering quality can be observed. K-Means is simpler and faster but may perform poorly when clusters overlap or have complex shapes. EM generally provides better clustering quality for such datasets because it models the underlying data distribution and handles overlapping clusters more effectively. Therefore, EM often produces more accurate and flexible clustering results compared to K-Means, especially for real-world datasets with varying cluster structures.

Code –

```
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
wine = load_wine()
X = wine.data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X_pca)
gmm = GaussianMixture(n_components=3, random_state=42)
gmm_labels = gmm.fit_predict(X_pca)
kmeans_score = silhouette_score(X_pca, kmeans_labels)
gmm_score = silhouette_score(X_pca, gmm_labels)

print("Silhouette Score (K-Means):", round(kmeans_score, 3))
print("Silhouette Score (EM/GMM):", round(gmm_score, 3))
Silhouette Score (K-Means): 0.56
Silhouette Score (EM/GMM): 0.559
plt.figure(figsize=(12, 5))
# K-Means plot
plt.subplot(1, 2, 1)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels)
plt.title("K-Means Clustering (PCA Reduced Data)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
```




```
plt.subplot(1, 2, 2)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=gmm_labels)
plt.title("EM Clustering (Gaussian Mixture Model)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.tight_layout()
plt.show()
```



ANN

Aim - Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

Theory - An Artificial Neural Network (ANN) is a computational model inspired by the structure and functioning of the human brain. It consists of interconnected processing units called neurons, organized into input, hidden, and output layers. Each connection between neurons has an associated weight, and learning in an ANN involves adjusting these weights so that the network can correctly map input data to output classes or values. ANNs are widely used for classification and prediction tasks due to their ability to model complex nonlinear relationships.

Backpropagation is a supervised learning algorithm used to train artificial neural networks. During training, the input data is passed forward through the network to produce an output, which is then compared with the actual target output to compute an error. This error is propagated backward through the network, and the weights are updated using gradient descent in order to minimize the loss function. The process of forward propagation, error calculation, and backward weight adjustment is repeated iteratively until the network converges to an optimal solution.

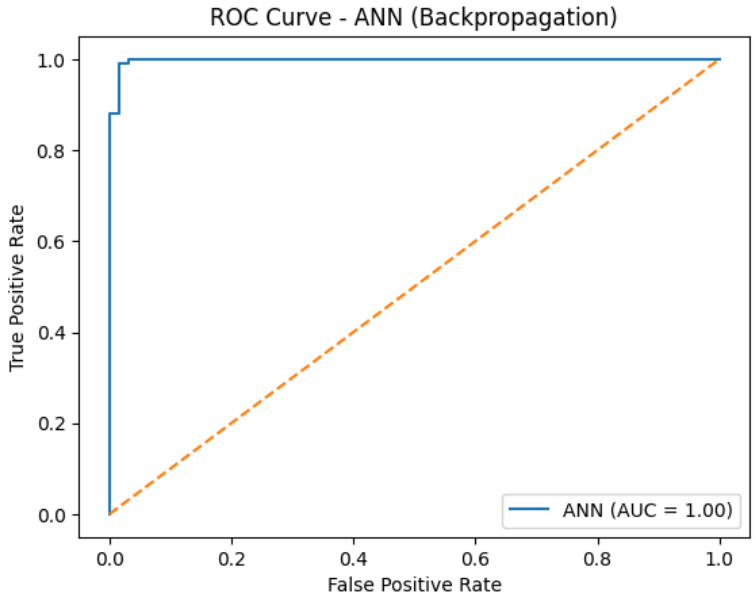
Once trained, the ANN can be tested using unseen data to evaluate its performance. The effectiveness of the network is measured using metrics such as accuracy, loss, or error rate. ANNs trained using backpropagation are powerful tools for pattern recognition and classification, but their performance depends on factors such as network architecture, learning rate, and quality of the dataset.

Code –

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer()
df=pd.DataFrame(data.data, columns=data.feature_names)
X = data.data
y = data.target
print("Feature shape:", X.shape)
print("Target shape:", y.shape)
Feature shape: (569, 30)
Target shape: (569,)
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
from sklearn.neural_network import MLPClassifier

ann = MLPClassifier(
```

```
    hidden_layer_sizes=(20, 10),
    activation='relu',
    solver='adam',        # Uses backpropagation
    max_iter=1000,
    random_state=42
)
ann.fit(X_train, y_train)
y_pred = ann.predict(X_test)

from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
Accuracy: 0.9766081871345029
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

Confusion Matrix:
 [[ 62   1]
 [  3 105]]
from sklearn.metrics import classification_report

print(classification_report(
    y_test,
    y_pred,
    target_names=data.target_names
))
              precision    recall  f1-score   support

   malignant       0.95      0.98      0.97        63
      benign       0.99      0.97      0.98       108

    accuracy                           0.98       171
   macro avg       0.97      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
from sklearn.metrics import roc_curve, auc

# Predict probabilities for positive class
y_prob = ann.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
```

```
# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, label=f"ANN (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - ANN (Backpropagation)")
plt.legend(loc="lower right")
plt.show()
```



Experiment – 6
Agglomerative Clustering

Aim- To Demonstrate Agglomerative Hierarchical Clustering on the given dataset and analyze cluster formation.

Theory- Agglomerative Hierarchical Clustering is a bottom-up clustering technique where each data point initially starts as its own individual cluster. The algorithm then repeatedly merges the two most similar clusters until a stopping condition is met—usually when all points form a single cluster or when a predefined number of clusters is reached. Similarity between clusters is calculated using a distance metric such as Euclidean distance, and the way distances are measured between clusters depends on the chosen linkage method (single, complete, average, or Ward's method). The entire merging process is represented using a dendrogram, a tree-like diagram that visually shows how clusters are combined at different distance levels.

One major advantage of agglomerative hierarchical clustering is that it does not require the number of clusters to be specified in advance, unlike k-means. This makes it useful for exploratory data analysis where the underlying structure of the data is unknown. Additionally, the dendrogram provides interpretability, allowing analysts to choose a suitable cut-off level to obtain meaningful clusters. However, the algorithm is computationally expensive for large datasets and is sensitive to noise and outliers, since early incorrect merges cannot be undone. Despite these limitations, agglomerative clustering is widely used in domains like bioinformatics, text analysis, and social network analysis where hierarchical relationships are important.

Code-

```
import pandas as pd
df=pd.read_csv('income.csv')
df.head()
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week	income_level
0	39	77516	13	2174	0	40	0
1	50	83311	13	0	0	13	0
2	38	215646	9	0	0	40	0
3	53	234721	7	0	0	40	0
4	28	338409	13	0	0	40	0

```
df.isna().sum()
age             0
fnlwgt          0
education_num   0
capital_gain    0
capital_loss    0
hours_per_week  0
```

```
income_level      0
dtype: int64
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             48842 non-null  int64
 1   fnlwgt          48842 non-null  int64
 2   education_num   48842 non-null  int64
 3   capital_gain    48842 non-null  int64
 4   capital_loss    48842 non-null  int64
 5   hours_per_week  48842 non-null  int64
 6   income_level    48842 non-null  int64
dtypes: int64(7)
memory usage: 2.6 MB
df.columns
Index(['age', 'fnlwgt', 'education_num', 'capital_gain', 'capital_loss',
       'hours_per_week', 'income_level'],
      dtype='object')
x=df.drop('income_level',axis=1)
y=df['income_level']
x.head()
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week
0	39	77516	13	2174	0	40
1	50	83311	13	0	0	13
2	38	215646	9	0	0	40
3	53	234721	7	0	0	40
4	28	338409	13	0	0	40

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x=scaler.fit_transform(x)
x
array([[ 0.02599598, -1.06197924,  1.136512  ,  0.14693247, -0.2171271 ,
        -0.03408696],
       [ 0.82830842, -1.00710388,  1.136512  , -0.14480353, -0.2171271 ,
        -2.21303208],
       [-0.04694151,  0.24603353, -0.41933527, -0.14480353, -0.2171271 ,
        -0.03408696],
       ...,
       [-0.04694151,  1.75486457,  1.136512  , -0.14480353, -0.2171271 ,
```

```
  0.77292975],
 [ 0.39068346, -1.00161161,  1.136512  ,  0.58722034, -0.2171271 ,
  -0.03408696],
 [-0.26575399, -0.07117353,  1.136512  , -0.14480353, -0.2171271 ,
   1.57994645]])
from sklearn.cluster import AgglomerativeClustering
from sklearn.model_selection import train_test_split

model = AgglomerativeClustering(n_clusters=3)
X_train, X_test, y_train, y_test=train_test_split(x,y,test_size=0.3,random_state=42)
model.fit(X_train)
```

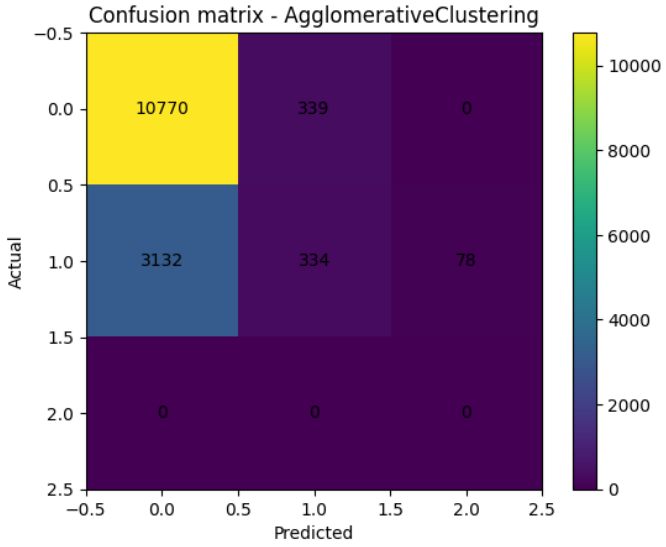
AgglomerativeClustering

▼ Parameters

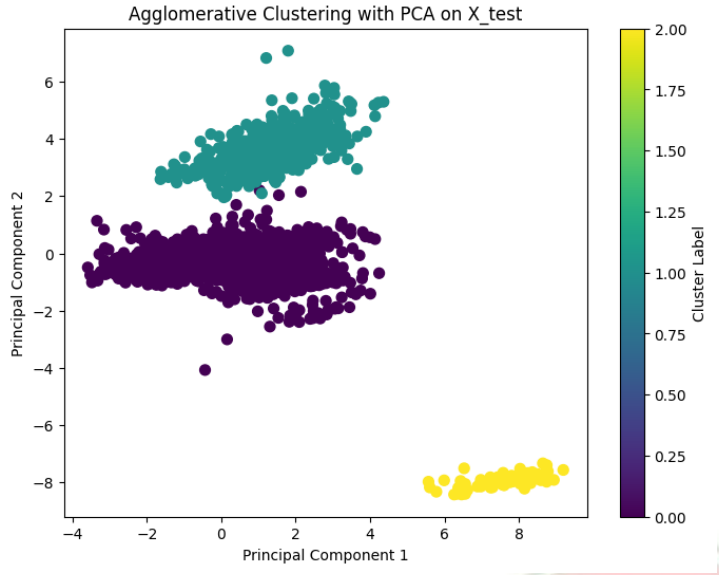
n_clusters	3
metric	'euclidean'
memory	None
connectivity	None
compute_full_tree	'auto'
linkage	'ward'
distance_threshold	None
compute_distances	False

```
y_pred=model.fit_predict(X_test)
from sklearn.metrics import accuracy_score, classification_report,confusion_matrix
print("accuracy_score=",accuracy_score(y_pred,y_test))
print("classification_report",classification_report(y_pred,y_test))
print("confussion matrix",confusion_matrix(y_pred,y_test))
accuracy_score= 0.7577970381491844
classification_report              precision    recall  f1-score   support

           0       0.97      0.77      0.86     13902
           1       0.09      0.50      0.16       673
           2       0.00      0.00      0.00        78

    accuracy                           0.76     14653
   macro avg       0.35      0.42      0.34     14653
weighted avg       0.92      0.76      0.82     14653

confussion matrix [[10770  3132     0]
 [  339   334     0]
 [    0    78     0]]
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.imshow(cm)
```

```
plt.title("Confusion matrix - AgglomerativeClustering")
plt.colorbar()
plt.xlabel("Predicted")
plt.ylabel("Actual")

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j,i,cm[i,j],ha="center",va="center")
plt.show()
```

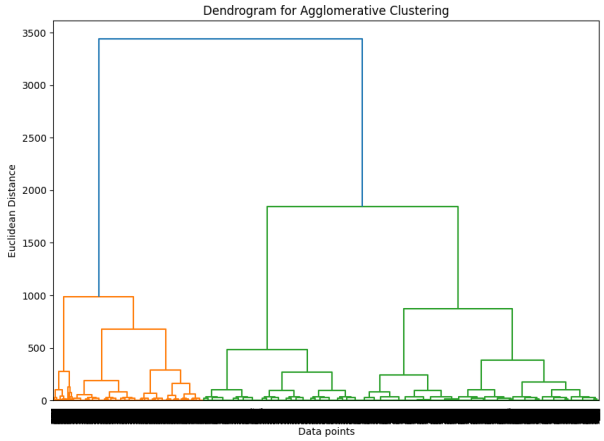



```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)  # Reducing to 2 components (2D)
X_reduced = pca.fit_transform(X_test)  # Apply PCA on test data (X_test)

# Plotting the results
plt.figure(figsize=(8, 6))
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y_pred, cmap='viridis', s=50)

# Adding labels and title
plt.title('Agglomerative Clustering with PCA on X_test')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

# Add color bar to indicate the cluster labels
plt.colorbar(label='Cluster Label')

# Display the plot
plt.show()
```




```
from scipy.cluster.hierarchy import dendrogram, linkage
agg_clustering = AgglomerativeClustering(n_clusters=3, linkage='ward')
df['Cluster'] = agg_clustering.fit_predict(df)
print(df)
       age  fnlwgt  education_num  capital_gain  capital_loss  hours_per_week  \
0       39   77516             13          2174             0              40
1       50   83311             13             0             0              13
2       38  215646              9             0             0              40
3       53  234721              7             0             0              40
4       28  338409             13             0             0              40
...    ...     ...            ...           ...           ...             ...
48837   39  215419             13             0             0              36
48838   64  321403              9             0             0              40
48839   38  374983             13             0             0              50
48840   44   83891             13          5455             0              40
48841   35  182148             13             0             0              60

       income_level  Cluster
0                 0        2
1                 0        2
2                 0        1
3                 0        1
4                 0        0
...             ...      ...
48837             0        1
48838             0        0
48839             0        0
48840             0        2
48841             1        1

[48842 rows x 8 columns]
```

```
# Plot Dendrogram
linked = linkage(df[['income_level', 'age']], method='ward')

plt.figure(figsize=(10, 7))
dendrogram(linked)
plt.title('Dendrogram for Agglomerative Clustering')
plt.xlabel('Data points')
plt.ylabel('Euclidean Distance')
plt.show()
```



DBSCAN

Aim- To implement DBSCAN clustering and identify dense clusters and noise points in the dataset.

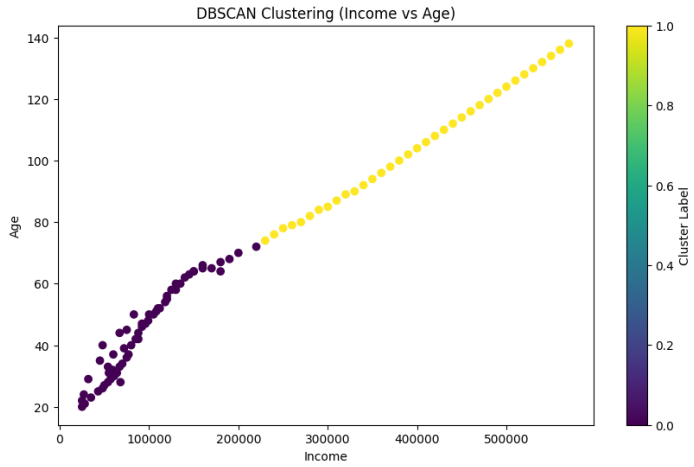
Theory- DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm that groups data points based on how closely they are packed together. It uses two key parameters: ε (epsilon), which defines the radius of a neighborhood around a point, and MinPts, the minimum number of points required to form a dense region. Points are classified as core points (enough neighbors within ε), border points (close to a core point but with fewer neighbors), or noise/outliers (points that do not belong to any cluster). Clusters are formed by connecting core points that are density-reachable from one another, allowing DBSCAN to discover clusters of arbitrary shape.

A major strength of DBSCAN is its ability to identify noise and outliers explicitly, which many clustering algorithms fail to do. It also does not require specifying the number of clusters in advance and works well with non-spherical cluster structures. However, DBSCAN is sensitive to the choice of ε and MinPts—poor parameter selection can lead to missed clusters or excessive noise. Additionally, it struggles with datasets where cluster densities vary significantly. Despite these challenges, DBSCAN is widely used in applications such as spatial data analysis, anomaly detection, and image processing due to its robustness and flexibility.

Code-

```
import pandas as pd
df=pd.read_csv("dbscan.csv")
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 95 entries, 0 to 94
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
```

```
--- ------    -------------- -----
 0   Income    95 non-null     int64
 1   Age       95 non-null     int64
 2   Location  95 non-null     int32
 3   Cluster   95 non-null     int64
dtypes: int32(1), int64(3)
memory usage: 2.7 KB
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import DBSCAN
le = LabelEncoder()
df['Location'] = le.fit_transform(df['Location'])

# Select features for clustering
X = df[['Income', 'Age', 'Location']]

# Run DBSCAN clustering
db = DBSCAN(eps=30000, min_samples=5).fit(X)

# Add the cluster labels to the data
df['Cluster'] = db.labels_

# Display the resulting clusters
print(df[['Income', 'Age', 'Location', 'Cluster']])
    Income  Age  Location  Cluster
0    25000   22         0        0
1    27000   24         1        0
2    32000   29         0        0
3    45000   35         2        0
4    48000   40         1        0
..     ...  ...       ...      ...
90  530000  130         0        1
91  540000  132         1        1
92  550000  134         2        1
93  560000  136         3        1
94  570000  138         4        1

[95 rows x 4 columns]
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.scatter(df['Income'], df['Age'], c=df['Cluster'], cmap='viridis', marker='o')
plt.title('DBSCAN Clustering (Income vs Age)')
plt.xlabel('Income')
plt.ylabel('Age')
plt.colorbar(label='Cluster Label')
plt.show()
```



Boosting

Aim-To implement Boosting techniques and evaluate their performance on a classification dataset.

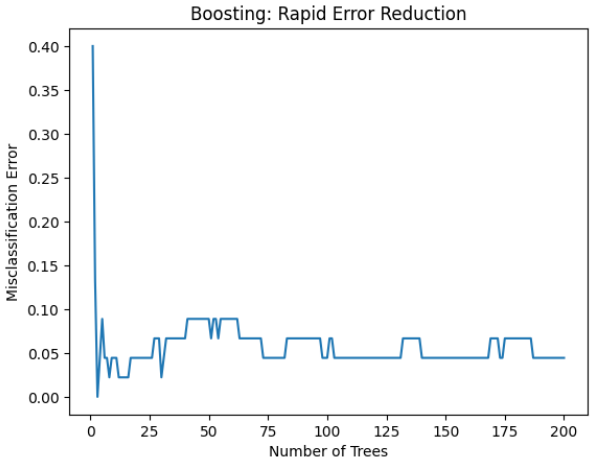
Theory -Boosting is an ensemble learning technique that improves model performance by combining multiple weak learners to form a strong predictive model. Instead of training models independently, boosting trains them sequentially, where each new model focuses more on the data points that were misclassified by previous models. During this process, higher importance (weights) is given to difficult observations, forcing subsequent learners to pay extra attention to them. Common weak learners used in boosting are shallow decision trees (decision stumps), which on their own perform only slightly better than random guessing.

Popular boosting algorithms such as AdaBoost, Gradient Boosting, and XGBoost differ in how they adjust weights and optimize errors, but all follow the same core idea of reducing bias and improving accuracy through iteration. Boosting is especially effective for complex datasets and often achieves high predictive performance in classification and regression tasks. However, it can be sensitive to noisy data and outliers, since misclassified points may receive increasing emphasis. Despite this limitation, boosting remains one of the most powerful and widely used techniques in machine learning competitions and real-world applications.

Code-

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
wine = load_wine()
X = wine.data
y = wine.target
le = LabelEncoder()
y = le.fit_transform(y)
```

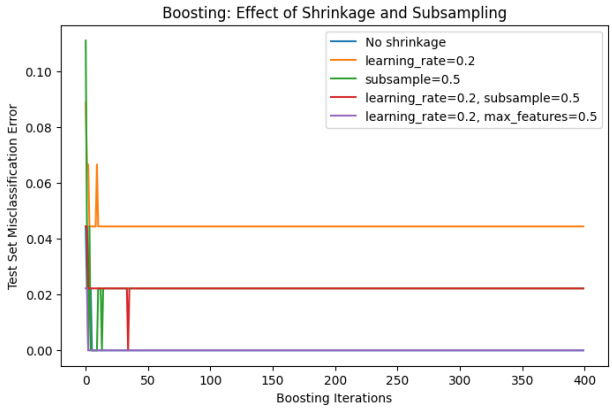
```
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
)
boost_errors = []
trees = range(1, 201)
for n in trees:
    boost = AdaBoostClassifier(
        estimator=DecisionTreeClassifier(max_depth=1),
        n_estimators=n,
        learning_rate=0.5,
        random_state=42
    )
    boost.fit(X_train, y_train)
    pred = boost.predict(X_test)
    boost_errors.append(1 - accuracy_score(y_test, pred))
plt.figure()
plt.plot(trees, boost_errors)
plt.xlabel("Number of Trees")
plt.ylabel("Misclassification Error")
plt.title("Boosting: Rapid Error Reduction")
plt.show()
```




```
boost_configs = {
    "No shrinkage": GradientBoostingClassifier(
        learning_rate=1.0,
        subsample=1.0,
        max_features=None,
        random_state=42
    ),

    "learning_rate=0.2": GradientBoostingClassifier(
        learning_rate=0.2,
        subsample=1.0,
```

```
        max_features=None,
        random_state=42
    ),

    "subsample=0.5": GradientBoostingClassifier(
        learning_rate=1.0,
        subsample=0.5,
        max_features=None,
        random_state=42
    ),

    "learning_rate=0.2, subsample=0.5": GradientBoostingClassifier(
        learning_rate=0.2,
        subsample=0.5,
        max_features=None,
        random_state=42
    ),

    "learning_rate=0.2, max_features=0.5": GradientBoostingClassifier(
        learning_rate=0.2,
        subsample=1.0,
        max_features=0.5,
        random_state=42
    )
}
n_estimators = 400
boosting_errors = {}

for name, model in boost_configs.items():
    model.set_params(n_estimators=n_estimators)
    model.fit(X_train, y_train)

    errors = []
    for y_pred in model.staged_predict(X_test):
        errors.append(1 - accuracy_score(y_test, y_pred))

    boosting_errors[name] = errors
plt.figure(figsize=(8, 5))

for name, errors in boosting_errors.items():
    plt.plot(errors, label=name)

plt.xlabel("Boosting Iterations")
plt.ylabel("Test Set Misclassification Error")
plt.title("Boosting: Effect of Shrinkage and Subsampling")
```

```
plt.legend()
plt.show()
```



.

Bagging

Aim-To implement Bagging ensemble techniques and analyze classification performance.

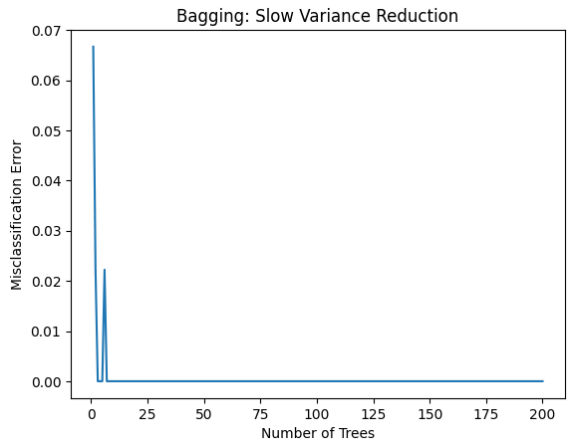
Theory- Bagging (Bootstrap Aggregating) is an ensemble learning technique designed to improve model stability and accuracy by reducing variance. It works by creating multiple different training datasets using bootstrap sampling, where samples are drawn randomly with replacement from the original dataset. A separate model (often a decision tree) is trained on each bootstrap sample, and the final prediction is obtained by aggregating the outputs of all models—using majority voting for classification or averaging for regression. Since each model sees a slightly different version of the data, their errors tend to cancel out.

The main advantage of bagging is its effectiveness in handling high-variance models, especially decision trees, which are sensitive to small changes in data. Unlike boosting, bagging trains all models independently and does not focus on misclassified points, making it more robust to noise and outliers. A well-known example of bagging is the Random Forest algorithm, which further improves performance by introducing randomness in feature selection. Overall, bagging is simple, scalable, and widely used when overfitting is a concern.

Code-

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

from sklearn.datasets import load_wine
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score
wine = load_wine()
X = wine.data
y = wine.target
le = LabelEncoder()
y = le.fit_transform(y)
```

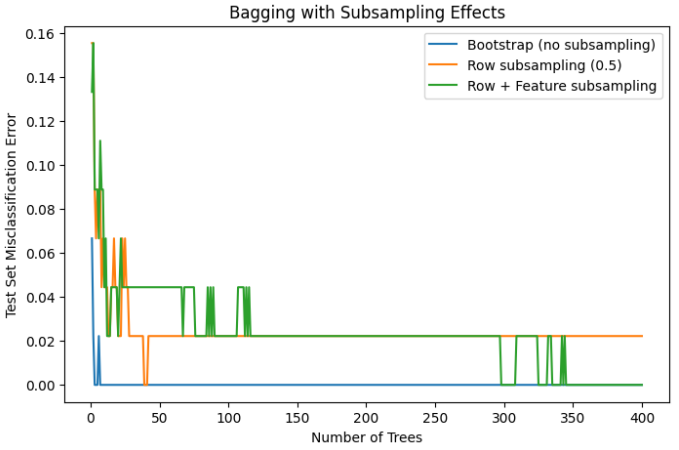
```
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
)
bag_errors = []
trees = range(1, 201)

for n in trees:
    bag = BaggingClassifier(
        estimator=DecisionTreeClassifier(),
        n_estimators=n,
        random_state=42,
        n_jobs=-1
    )
    bag.fit(X_train, y_train)
    pred = bag.predict(X_test)
    bag_errors.append(1 - accuracy_score(y_test, pred))
plt.figure()
plt.plot(trees, bag_errors)
plt.xlabel("Number of Trees")
plt.ylabel("Misclassification Error")
plt.title("Bagging: Slow Variance Reduction")
plt.show()
```




```
trees = range(1, 401)

bag_full = []
bag_subsample = []
bag_row_feature = []

for n in trees:

    # 1 Classic Bagging (Bootstrap)
    model_full = BaggingClassifier(
```

```
        estimator=DecisionTreeClassifier(),
        n_estimators=n,
        bootstrap=True,
        max_samples=1.0,
        random_state=42,
        n_jobs=-1
    )
    model_full.fit(X_train, y_train)
    bag_full.append(
        1 - accuracy_score(y_test, model_full.predict(X_test))
    )

    # 2️⃣ Row Subsampling (like subsample=0.5)
    model_sub = BaggingClassifier(
        estimator=DecisionTreeClassifier(),
        n_estimators=n,
        bootstrap=True,
        max_samples=0.5,
        random_state=42,
        n_jobs=-1
    )
    model_sub.fit(X_train, y_train)
    bag_subsample.append(
        1 - accuracy_score(y_test, model_sub.predict(X_test))
    )

    # 3️⃣ Row + Feature Subsampling
    model_rf_like = BaggingClassifier(
        estimator=DecisionTreeClassifier(),
        n_estimators=n,
        bootstrap=True,
        max_samples=0.5,
        max_features=0.5,
        random_state=42,
        n_jobs=-1
    )
    model_rf_like.fit(X_train, y_train)
    bag_row_feature.append(
        1 - accuracy_score(y_test, model_rf_like.predict(X_test))
    )
plt.figure(figsize=(8, 5))
plt.plot(trees, bag_full, label="Bootstrap (no subsampling)")
plt.plot(trees, bag_subsample, label="Row subsampling (0.5)")
plt.plot(trees, bag_row_feature, label="Row + Feature subsampling")

plt.xlabel("Number of Trees")
```

```
plt.ylabel("Test Set Misclassification Error")
plt.title("Bagging with Subsampling Effects")
plt.legend()
plt.show()
```



Experiment – 7
Locally Weighted Regression

Aim- To Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Theory- Locally Weighted Regression (LWR), also known as Locally Weighted Linear Regression, is a non-parametric supervised learning algorithm used for regression problems. Unlike parametric models that learn a single global function for the entire dataset, LWR fits a separate local model for each query point, making it highly effective for modeling non-linear relationships in data.

The key idea behind LWR is that points closer to the query point should have a greater influence on the prediction than points farther away. This is achieved using a distance-based weighting function, commonly a Gaussian kernel, which assigns higher weights to nearby data points and lower weights to distant ones.

Since LWR does not assume any fixed functional form for the data and does not learn global parameters during training, it is classified as a non-parametric and instance-based learning method. The model is constructed dynamically at prediction time, which allows it to adapt flexibly to local variations in the dataset.

For a given query point, LWR constructs a local regression model by minimizing a weighted least squares error function. The weights are computed using a kernel function that depends on the distance between the query point and each training sample. The Gaussian kernel is widely used due to its smooth weighting behavior.

The bandwidth parameter (τ) controls the width of the neighborhood considered:

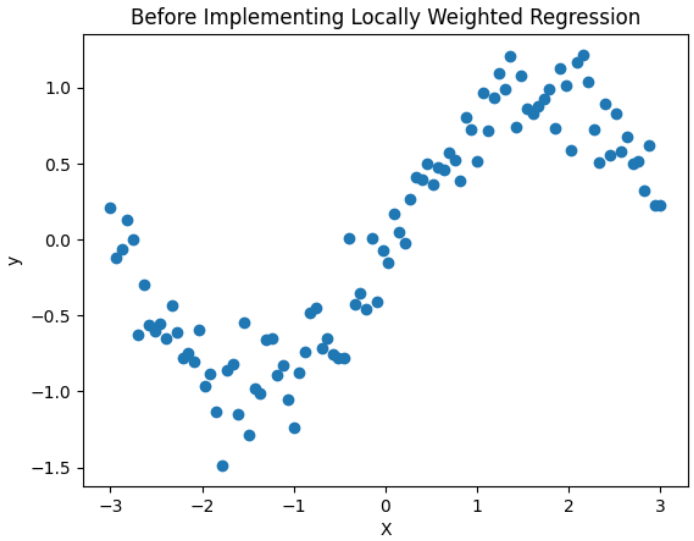
A small τ results in a narrow neighborhood, leading to high variance and possible overfitting.

A large τ results in a broader neighborhood, leading to smoother predictions but possible underfitting.

Once the weights are calculated, a local linear regression model is fitted, and the predicted value is obtained for the query point. This process is repeated independently for each query point.

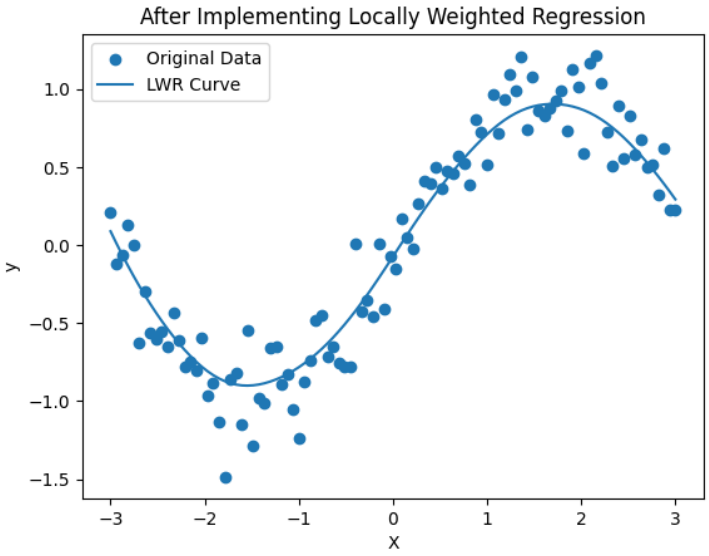
Code-

```
import numpy as np
import matplotlib.pyplot as plt
# Generate synthetic non-linear dataset
np.random.seed(0)
X = np.linspace(-3, 3, 100)
y = np.sin(X) + np.random.normal(0, 0.2, 100)
# Reshape for consistency
X = X.reshape(-1, 1)
y = y.reshape(-1, 1)
plt.figure()
plt.scatter(X, y)
plt.title("Before Implementing Locally Weighted Regression")
plt.xlabel("X")
plt.ylabel("y")
plt.show()
```




```
def locally_weighted_regression(X, y, x_query, tau):
    """
    X       : Training data (n x 1)
    y       : Target values (n x 1)
    x_query : Point to predict
    tau     : Bandwidth parameter
    """
    m = X.shape[0]

    # Add bias term
    X_b = np.hstack((np.ones((m, 1)), X))
    x_q = np.array([1, x_query])

    # Weight matrix
    W = np.zeros((m, m))
    for i in range(m):
        diff = X[i] - x_query
        W[i, i] = np.exp(-(diff ** 2) / (2 * tau ** 2))

    # Closed-form solution
    theta = np.linalg.pinv(X_b.T @ W @ X_b) @ X_b.T @ W @ y

    return x_q @ theta
tau = 0.4  # Bandwidth
y_pred = []

for x in X:
    y_pred.append(locally_weighted_regression(X, y, x[0], tau))

y_pred = np.array(y_pred)
```

```
plt.figure()
plt.scatter(X, y, label="Original Data")
plt.plot(X, y_pred, label="LWR Curve")
plt.title("After Implementing Locally Weighted Regression")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```



Feature Selection Using Kernels

Aim- To select the most relevant features from a dataset using kernel-based techniques (Feature selection using Kernels)

Theory- Feature selection is an important preprocessing step in machine learning that aims to identify the most relevant features from a dataset while removing redundant and irrelevant ones. Kernel-based feature selection techniques are particularly effective because they can capture non-linear relationships between input features and the target variable. Unlike traditional linear methods, kernel-based approaches implicitly map data into a higher-dimensional feature space, enabling better discrimination between important and unimportant features.

In kernel-based feature selection, kernel functions such as linear, polynomial, and Gaussian (RBF) kernels are used to measure the dependency between individual features and the output variable. A commonly used criterion is the Hilbert–Schmidt Independence Criterion (HSIC), which quantifies the statistical dependence between a feature and the target using kernel matrices. Features with higher kernel-based dependency scores are considered more relevant, as they contribute more significantly to the prediction task.

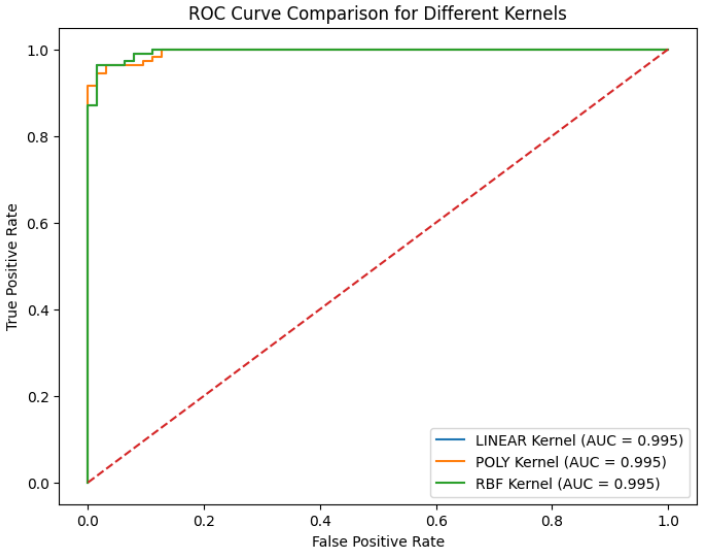
By selecting features based on kernel-based relevance measures, the dimensionality of the dataset can be reduced while preserving important information. This leads to improved model performance, reduced computational complexity, and better generalization. Kernel-based feature selection is especially useful in real-world datasets where complex, non-linear interactions exist between features and class labels.

Code-

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

```
import matplotlib.pyplot as plt

from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import linear_kernel, polynomial_kernel, rbf_kernel
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
data = load_breast_cancer()

X = data.data
y = data.target
feature_names = data.feature_names

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
def compute_hsic_kernel(X_feature, y, kernel_type="rbf"):
    X_feature = X_feature.reshape(-1, 1)
    y = y.reshape(-1, 1)
    n = X_feature.shape[0]

    if kernel_type == "linear":
        Kx = linear_kernel(X_feature)
        Ky = linear_kernel(y)
    elif kernel_type == "poly":
        Kx = polynomial_kernel(X_feature, degree=3)
        Ky = polynomial_kernel(y, degree=3)
    elif kernel_type == "rbf":
        Kx = rbf_kernel(X_feature, gamma=0.5)
        Ky = rbf_kernel(y, gamma=0.5)

    H = np.eye(n) - (1/n) * np.ones((n, n))
    hsic = np.trace(Kx @ H @ Ky @ H) / ((n - 1) ** 2)
    return hsic
kernels = ["linear", "poly", "rbf"]
selected_features = {}

for kernel in kernels:
    scores = []
    for i in range(X_scaled.shape[1]):
        score = compute_hsic_kernel(X_scaled[:, i], y, kernel)
        scores.append(score)

    df_scores = pd.DataFrame({
        "Feature": feature_names,
        "HSIC Score": scores
```

```
    }).sort_values(by="HSIC Score", ascending=False)

    selected_features[kernel] = df_scores.head(10)["Feature"].values

roc_results = {}

for kernel in kernels:
    feature_mask = [f in selected_features[kernel] for f in feature_names]
    X_selected = X_scaled[:, feature_mask]

    X_train, X_test, y_train, y_test = train_test_split(
        X_selected, y, test_size=0.3, random_state=42
    )

    model = LogisticRegression(max_iter=500)
    model.fit(X_train, y_train)

    y_scores = model.predict_proba(X_test)[:, 1]

    fpr, tpr, _ = roc_curve(y_test, y_scores)
    roc_auc = auc(fpr, tpr)

    roc_results[kernel] = (fpr, tpr, roc_auc)
plt.figure(figsize=(8, 6))

for kernel in kernels:
    fpr, tpr, roc_auc = roc_results[kernel]
    plt.plot(fpr, tpr, label=f"{kernel.upper()} Kernel (AUC = {roc_auc:.3f})")

plt.plot([0, 1], [0, 1], linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison for Different Kernels")
plt.legend()
plt.show()
```



Experiment – 8
Naïve Bayesian Classifier

Aim - Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a.CSV file. Compute the accuracy of the classifier, considering few test data sets

Theory - Naïve Bayes is a probabilistic classification technique based on Bayes' theorem, which provides a way to calculate the probability of a class given a set of input features. The classifier is termed naïve because it assumes that all features are conditionally independent of each other given the class label. Despite this simplifying assumption, Naïve Bayes performs remarkably well for many real-world problems, especially in cases involving high-dimensional data. It is widely used in applications such as text classification, spam detection, and medical diagnosis due to its simplicity and efficiency.

In this experiment, a Naïve Bayesian classifier is implemented using a training dataset stored in a CSV file. During the training phase, the algorithm learns the prior probabilities of each class and the conditional probabilities of each feature value given a class. For a new test instance, the classifier applies Bayes' theorem to compute the posterior probability for each class and assigns the class with the highest probability to that instance. The implementation handles multiple attributes and uses the learned probability distributions to perform classification on unseen data.

The performance of the classifier is evaluated by testing it on a few test datasets and comparing the predicted class labels with the actual labels. The accuracy of the classifier is computed as the ratio of correctly classified instances to the total number of test instances. This evaluation helps in understanding how well the Naïve Bayes classifier generalizes to new data and demonstrates its effectiveness as a simple yet powerful machine learning algorithm for classification tasks.

Code –

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,precision_score,recall_score
from sklearn.preprocessing import StandardScaler
df=pd.read_csv("./datasets/diabetes.csv")
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB  X=df.drop('Outcome',axis=1)
```

```
Y=df['Outcome']
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.2)
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)  model=GaussianNB()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print("Predicted:", y_pred.tolist())
print("Actual:   ", y_test.tolist())

print("Accuracy :", accuracy)
print("Precision:", precision)
print("Recall   :", recall)
```

```
Predicted: [1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1,
0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1,
0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1]
Actual:    [0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1,
1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1]
Accuracy : 0.7597402597402597
Precision: 0.6071428571428571
Recall   : 0.6938775510204082
```

Aim - Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

Theory - Naïve Bayes is a statistical text classification technique based on **Bayes' theorem**, used to predict the class of a document given its contents. In document classification, each document is treated as a collection of words (features), and the classifier assumes that the presence of a particular word in a document is **independent** of the presence of other words, given the class label. Although this independence assumption is a simplification, Naïve Bayes performs effectively for many document classification tasks such as spam filtering, sentiment analysis, and topic categorization due to its probabilistic foundation and low computational complexity.

In this experiment, a Naïve Bayesian classifier is implemented using built-in Java APIs to classify a given set of documents. The training phase involves computing prior probabilities for each document class and conditional probabilities of words given a class, usually using frequency-based estimates. During testing, the classifier calculates posterior probabilities for each class using Bayes' theorem and assigns the class with the highest probability to the document. Preprocessing steps such as tokenization and stop-word removal may be applied to improve classification performance.

The performance of the classifier is evaluated using **accuracy, precision, and recall** metrics. Accuracy measures the overall correctness of the model, while precision indicates how many of the documents predicted as a particular class are actually correct. Recall measures the ability of the classifier to correctly identify all documents belonging to a given class. These metrics together provide a comprehensive evaluation of the Naïve Bayes classifier's effectiveness in document classification tasks.

Code –

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
df = pd.read_csv("./datasets/emails.csv")
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5728 entries, 0 to 5727
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    5728 non-null   object
 1   spam    5728 non-null   int64
dtypes: int64(1), object(1)
memory usage: 89.6+ KB
X = df["text"].astype(str)
y = df['spam']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
from sklearn.feature_extraction.text import CountVectorizer
vectorizer=CountVectorizer(stop_words="english",ngram_range=(1,2),max_features=20)
x_train_vec=vectorizer.fit_transform(X_train)
x_test_vec=vectorizer.transform(X_test)
feature_name=vectorizer.get_feature_names_out()
print(feature_name)
print(feature_name[1:10])
print(feature_name.shape)
print("\nDocument-Term Matrix:")
print(x_train_vec.toarray())
['10' '2000' '2001' 'cc' 'com' 'ect' 'ect ect' 'enron' 'group' 'hou'
 'hou ect' 'kaminski' 'know' 'pm' 'research' 'subject' 'thanks' 'time'
 'vince' 'vince kaminski']
['2000' '2001' 'cc' 'com' 'ect' 'ect ect' 'enron' 'group' 'hou']
(20,)

Document-Term Matrix:
[[0 2 0 ... 0 4 2]
 [0 0 3 ... 0 4 3]
 [0 0 0 ... 0 1 0]
```

```
 ...
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 1 1 0]
 [0 0 0 ... 0 0 0]]
print("x_train_vec shape:", x_train_vec.shape)
print("x_test_vec shape :", x_test_vec.shape)
x_train_vec shape: (4582, 20)
x_test_vec shape : (1146, 20)
model = MultinomialNB()
model.fit(x_train_vec, y_train)
y_pred = model.predict(x_test_vec)
spam_prob=model.feature_log_prob_[1]
nonspam_prob=model.feature_log_prob_[0]
top_indices_spam = spam_prob.argsort()[-15:]
top_indices_nonspam = nonspam_prob.argsort()[-15:]

print("Most Important Spam Words | No Spam Words:")
for i, j in zip(reversed(top_indices_spam), reversed(top_indices_nonspam)):
    print(feature_name[i], "    |     ", feature_name[j])
Most Important Spam Words | No Spam Words:
subject     |      enron
com     |      ect
time     |      subject
10     |      vince
know     |      hou
group     |      hou ect
research     |      2000
thanks     |      ect ect
2000     |      kaminski
pm     |      com
2001     |      vince kaminski
hou     |      cc
cc     |      pm
vince     |      2001
ect     |      research
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("\nAccuracy :", acc)
print("Precision:", prec)
print("Recall   :", rec)
print("F1 Score :", f1)

Accuracy : 0.9075043630017452
```

Precision: 0.7383419689119171
Recall : 0.9827586206896551
F1 Score : 0.8431952662721893

Bayesian Network

Aim - Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

Theory - A Bayesian Network is a probabilistic graphical model that represents a set of medical variables and their conditional dependencies using a directed acyclic graph (DAG). Each node in the network corresponds to a clinical attribute (such as age, cholesterol, blood pressure, or chest pain), while the edges represent probabilistic relationships between these variables. Bayesian networks are particularly effective in medical diagnosis because they explicitly model uncertainty and allow reasoning even when some patient information is missing or incomplete.

In this experiment, a Bayesian network is constructed using a standard Heart Disease dataset (for example, the UCI Heart Disease dataset). Important medical features such as age, sex, chest pain type, resting blood pressure, serum cholesterol, fasting blood sugar, ECG results, and maximum heart rate are modeled as nodes in the network, with the target node representing the presence or absence of heart disease. Using Java or Python machine learning libraries, the structure of the network and the conditional probability tables (CPTs) are learned from the dataset during the training phase.

Once the Bayesian network is trained, it is used to perform heart disease diagnosis by computing the posterior probability of heart disease given observed patient attributes. For a new patient, the model infers the likelihood of disease based on available clinical evidence. This probabilistic approach helps in decision-making by providing not just a classification outcome, but also a confidence level, demonstrating the usefulness of Bayesian networks in medical data analysis and clinical diagnosis systems.

Code –

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianNetwork
from pgmpy.inference import VariableElimination
heartDisease = pd.read_csv('./datasets/heart.csv')
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
```

```
 Attributes and datatypes
age          int64
sex          int64
cp           int64
trestbps       int64
chol         int64
fbs          int64
restecg        int64
```

```
thalach          int64
exang            int64
oldpeak        float64
slope            int64
ca              object
thal            object
heartdisease     int64
dtype: object
print(heartDisease[['age','sex','exang','cp','restecg','chol']])
     age  sex  exang  cp  restecg  chol
0     63    1      0   1        2   233
1     67    1      1   4        2   286
2     67    1      1   4        2   229
3     37    1      0   3        0   250
4     41    0      0   2        2   204
..   ...  ...    ...  ..      ...   ...
298   45    1      0   1        0   264
299   68    1      0   4        0   193
300   57    1      1   4        0   131
301   57    0      0   2        2   236
302   38    1      0   3        0   175

[303 rows x 6 columns]
heartDisease = heartDisease.replace('?',np.nan)
print('Sample instances from the dataset are given below')
print(heartDisease.head())
Sample instances from the dataset are given below
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   1       145   233    1        2      150      0      2.3      3
1   67    1   4       160   286    0        2      108      1      1.5      2
2   67    1   4       120   229    0        2      129      1      2.6      2
3   37    1   3       130   250    0        0      187      0      3.5      3
4   41    0   2       130   204    0        2      172      0      1.4      1

   ca thal  heartdisease
0  0    6             0
1  3    3             2
2  2    7             1
3  0    3             0
4  0    3             0
from sklearn.model_selection import train_test_split
from pgmpy.models import DiscreteBayesianNetwork
#Creat Model- Bayesian Network
train_df, test_df = train_test_split(heartDisease, test_size=0.2, random_state=42,
stratify=heartDisease['heartdisease'])
model = DiscreteBayesianNetwork([('age','heartdisease'),('sex','heartdisease'),(
```

```
'exang','heartdisease'),('cp','heartdisease'),('restecg','heartdisease'),('chol','heartdisease')])
# other estimators are BayesianEstimator, HillClimbSearch
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
INFO:pgmpy: Datatype (N=numerical, C=Categorical Unordered, O=Categorical Ordered) inferred
from data:
 {'age': 'N', 'sex': 'N', 'cp': 'N', 'trestbps': 'N', 'chol': 'N', 'fbs': 'N', 'restecg': 'N', 'thalach': 'N', 'exang': 'N',
'oldpeak': 'N', 'slope': 'N', 'ca': 'C', 'thal': 'C', 'heartdisease': 'N'}

 Learning CPD using Maximum likelihood estimators
# Inference engine. Other Inference Engines are BeliefPropagation, GibbsSampling
infer = VariableElimination(model)
example_patient = {'restecg':1}

result = infer.query(variables=['heartdisease'], evidence=example_patient)
print("\nDiagnosis Probability for example patient:")
print(result)

Diagnosis Probability for example patient:
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.2000 |
+-----------------+---------------------+
| heartdisease(1) |              0.2000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2000 |
+-----------------+---------------------+
| heartdisease(3) |              0.2000 |
+-----------------+---------------------+
| heartdisease(4) |              0.2001 |
+-----------------+---------------------+  # ----------------------------
# Predict on test set (basic evaluation)
# We'll predict target by taking whichever probability is higher
y_true = test_df['heartdisease'].tolist()
y_pred = []

for i in range(len(test_df)):
    row = test_df.iloc[i]
    evidence = {
        'age': int(row['age']),
        'sex': int(row['sex']),
        'restecg': int(row['restecg']),
        'chol': int(row['chol']),
        'exang':int(row['exang']),
        'cp':int(row['cp'])
```

```
    }
    q = infer.query(variables=['heartdisease'], evidence=evidence)
    # q.values[0] = P(target=0), q.values[1] = P(target=1)
    pred_class = 1 if q.values[1] > q.values[0] else 0
    y_pred.append(pred_class)
from sklearn.metrics import accuracy_score, precision_score, recall_score
acc = accuracy_score(y_true, y_pred)
prec = precision_score(y_true, y_pred,average='weighted')
rec = recall_score(y_true, y_pred, average='weighted')

print("\n=== Evaluation on Test Data ===")
print("Accuracy :", acc)
print("Precision:", prec)
print("Recall   :", rec)
```

```
=== Evaluation on Test Data ===
Accuracy : 0.7213114754098361
Precision: 0.5373770491803279
Recall   : 0.721311475409836
```