SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

FACULTY OF SCIENCE AND HUMANITIES

DEPARTMENT OF COMPUTER SCIENCE

KATTANKULATHUR – 603 203

PRACTICAL LAB RECORD

NAME :

REGISTER NUMBER :

CLASS & SECTION :

DEPARTMENT : **Computer Science**

SUBJECT CODE : **UCS20D07J**

SUBJECT NAME : **MACHINE LEARNING**

APRIL 2025

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

FACULTY OF SCIENCE AND HUMANITIES

DEPARTMENT OF COMPUTER SCIENCE

KATTANKULATHUR – 603 203

CERTIFICATE

Certified to be the bonafide of record of practical work done by

____

Register No.____________________ of B.Sc Degree course for **UCS20D07J - Machine Learning** in the Computer Science laboratory in SRM Institute of Science and Technology during the academic year 2024-2025.

STAFF INCHARGE **H.O.D**

Submitted for Semester Practical Examination held on _________________

INTERNAL EXAMINER I **INTERNAL EXAMINER II**

Table of Contents

Ex No: 1 **Date:** 6-12-24	**Creating and Loading A Data Frame**

Aim: Using Pandas to Create a Data Frame.

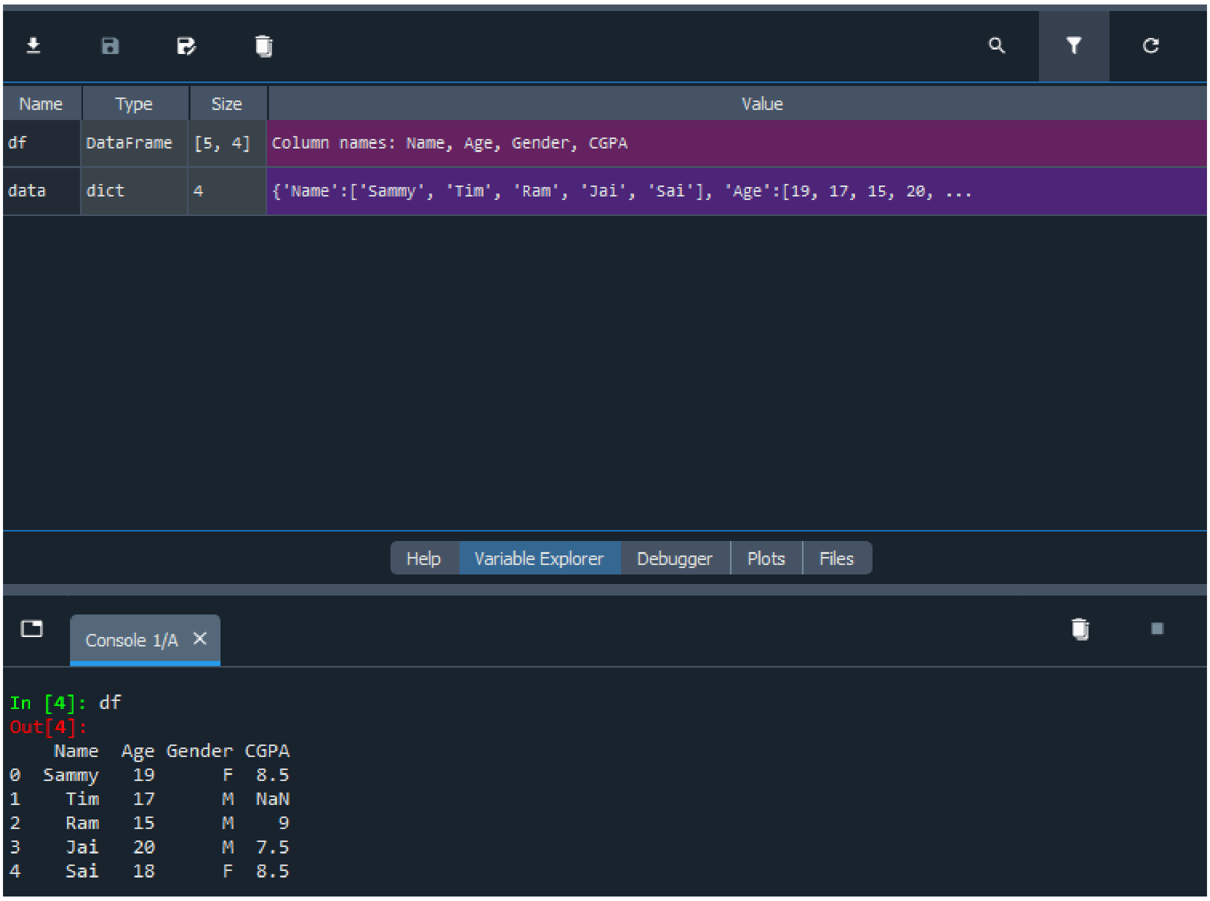
Procedure:

1. **Import Pandas Library** — Imports the `pandas` library to enable DataFrame operations.
2. **Create a Dictionary** — Defines a dictionary named `data` with keys representing column names and lists representing corresponding values.
3. **Create a DataFrame** — Uses the `pd.DataFrame()` function to convert the dictionary into a structured DataFrame.
4. **Display the DataFrame** — Displays the DataFrame, showing all the data in tabular format.
5. **Observation** — The `'NaN'` value in the `'CGPA'` column is treated as a **string**, not as a proper missing value.

Program :

```
# Working with Data Frames
import pandas as pd
data = { 'Name': ['Sammy','Tim', 'Ram', 'Jai', 'Sai'],
        'Age': [19, 17, 15, 20, 18],
        'Gender': ['F', 'M', 'M', 'M', 'F'],
        'CGPA' : [8.5, 'NaN',9, 7.5,8.5]}
df = pd.DataFrame(data)
#Printing the Dataframe
df
```

Output:


Name
Type
Size
Value
df
DataFrame
[5, 4]
Column names: Name, Age, Gender, CGPA
data
dict
4
{'Name':['Sammy', 'Tim', 'Ram', 'Jai', 'Sai'], 'Age':[19, 17, 15, 20, ...
Help
Variable Explorer
Debugger
Plots
Files
Console 1/A
In [4]: df
Out[4]:
    Name Age Gender CGPA
0 Sammy   19   F 8.5
1   Tim   17   M NaN
2   Ram   15   M   9
3   Jai   20   M 7.5
4   Sai   18   F 8.5

Ex No: 2 **Date:** 13-12-24	**Import a Dataset**

Aim: To import dataset that is a part of python library

Procedure:

1. **Import Required Libraries** — Imports the `fetch_california_housing` dataset from `sklearn.datasets` and the `pandas` library for data manipulation.
2. **Load the California Housing Dataset** — Uses `fetch_california_housing()` to load the dataset into the variable `housing`.
3. **Print Dataset Description** — Displays the dataset's description using `print(housing.DESCR)`, which provides details about the dataset's features, target values, and other relevant information.
4. **Create a DataFrame** — Constructs a DataFrame named `housing_df` using `pd.DataFrame()`, where:
   - `data=housing.data` assigns the dataset's feature values.
   - `columns=housing.feature_names` assigns the feature names as column headers.
5. **Display the First Few Rows** — Uses `print(housing_df.head())` to display the first five rows of the DataFrame for an initial overview of the data.

Program:

```
from sklearn.datasets import fetch_california_housing

import pandas as pd

housing = fetch_california_housing()

#Print Dataset Description

print(housing.DESCR)

# Create a DataFrame for easier data manipulation

housing_df = pd.DataFrame(data= housing.data, columns= housing.feature_names)

# Display the first few rows

print(housing_df.head())
```

Output:

```
- Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
  Statistics and Probability Letters, 33 (1997) 291-297

   MedInc  HouseAge  AveRooms  ...  AveOccup  Latitude  Longitude
0  8.3252      41.0  6.984127  ...  2.555556     37.88    -122.23
1  8.3014      21.0  6.238137  ...  2.109842     37.86    -122.22
2  7.2574      52.0  8.288136  ...  2.802260     37.85    -122.24
3  5.6431      52.0  5.817352  ...  2.547945     37.85    -122.25
4  3.8462      52.0  6.281853  ...  2.181467     37.85    -122.25

[5 rows x 8 columns]
```

Ex. No: 3

Date: 08-01-25

Plotting a Graph

Aim : Working with the plot() function in Mathplot lib

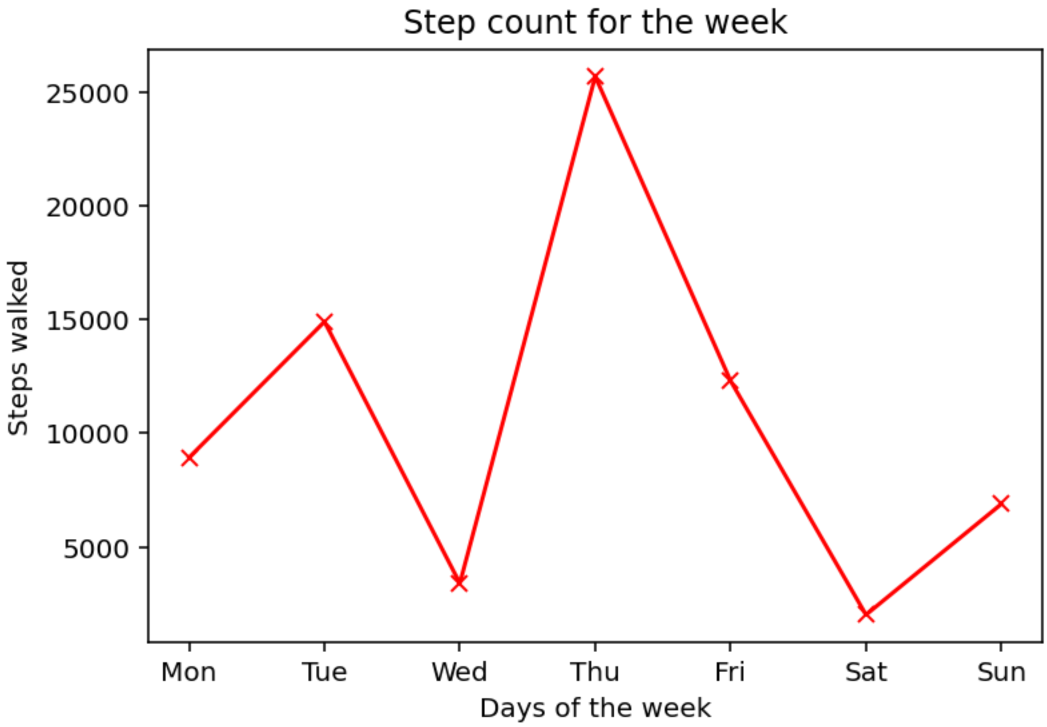
Procedure:

1. **Import Library** — Imports the `matplotlib.pyplot` library as `plt` for data visualization.
2. **Define Data** — Creates two lists:
   - `days` — Represents the days of the week.
   - `steps_walked` — Represents the number of steps walked each day.
3. **Plot the Data** — Uses `plt.plot()` to plot `days` on the x-axis and `steps_walked` on the y-axis with the format `"x-r"` (red line with 'x' markers).
4. **Add Title and Labels** —
   - `plt.title()` sets the chart title.
   - `plt.xlabel()` labels the x-axis as "Days of the week".
   - `plt.ylabel()` labels the y-axis as "Steps walked".
5. **Display the Plot** — Uses `plt.show()` to render the graph on the screen.

Program:

```
import matplotlib.pyplot as plt

days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]

steps_walked = [8934, 14902, 3409, 25672, 12300, 2023, 6890]

plt.plot(days, steps_walked, "x-r")

plt.title("Step count for the week")

plt.xlabel("Days of the week")

plt.ylabel("Steps walked")

plt.show()
```

Output



Step count for the week
25000
20000
15000
10000
5000
Steps walked
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Days of the week

Ex. No: 4 **Date:** 20-01-25	**Exploring plot() Function**

Aim: To plot a Comparative Graph

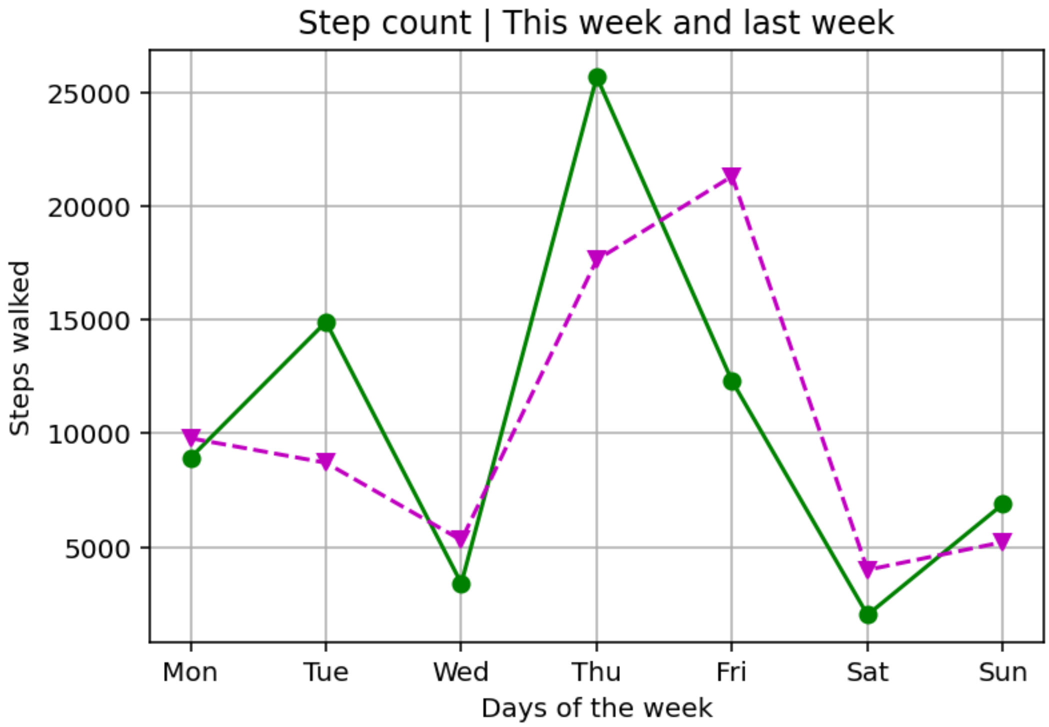
Procedure:

1. **Import Library** — Imports the `matplotlib.pyplot` library as `plt` for plotting.
2. **Define Data** — Creates three lists:
   - `days` — Represents the days of the week.
   - `steps_walked` — Represents the number of steps walked this week.
   - `steps_last_week` — Represents the number of steps walked last week.
3. **Plot This Week's Data** — Uses `plt.plot()` to plot `days` against `steps_walked` with the format `"o-g"` (green line with 'o' markers).
4. **Plot Last Week's Data** — Uses `plt.plot()` to plot `days` against `steps_last_week` with the format `"v--m"` (magenta dashed line with 'v' markers).
5. **Add Title and Labels** —
   - `plt.title()` sets the chart title.
   - `plt.xlabel()` labels the x-axis as "Days of the week".
   - `plt.ylabel()` labels the y-axis as "Steps walked".
6. **Add Grid** — Uses `plt.grid(True)` to display a grid for better readability.
7. **Display the Plot** — Uses `plt.show()` to render the graph on the screen.

Code:

```
import matplotlib.pyplot as plt

days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]

steps_walked = [8934, 14902, 3409, 25672, 12300, 2023, 6890]

steps_last_week = [9788, 8710, 5308, 17630, 21309, 4002, 5223]

plt.plot(days, steps_walked, "o-g")

plt.plot(days, steps_last_week, "v--m")

plt.title("Step count | This week and last week")

plt.xlabel("Days of the week")

plt.ylabel("Steps walked")

plt.grid(True)

plt.show()
```

Output:



Step count | This week and last week
Steps walked
25000
20000
15000
10000
5000
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Days of the week

:

Ex No: 5 **Date:** 3-02-25	**Exploring Statistical Functions**

Aim: To calculate and display statistical measures (mean, median, mode, and standard deviation) for a given dataset using the statistics module.

Procedure:

1. **Import Library** — Imports the `statistics` module for statistical calculations.
2. **Define Data** — Creates a list named `data` containing sample numeric values.
3. **Calculate Mean** — Uses `statistics.mean()` to compute the average of the dataset.
4. **Calculate Median** — Uses `statistics.median()` to compute the middle value of the sorted dataset.
5. **Calculate Mode** — Uses `statistics.mode()` to identify the most frequently occurring value in the dataset.
6. **Calculate Standard Deviation** — Uses `statistics.stdev()` to compute the sample standard deviation.
7. **Display Results** — Prints the dataset along with the calculated mean, median, mode, and standard deviation.

Program :

```
import statistics

# Sample data list

data = [1, 2, 2, 3, 4, 5, 5, 5, 6]

# Calculate statistical measures

mean_value = statistics.mean(data)

median_value = statistics.median(data)

mode_value = statistics.mode(data)  # Returns the single most common value

std_deviation = statistics.stdev(data)  # Sample standard deviation

# Print the results

print("Data: ", data)

print("Mean: ", mean_value)

print("Median: ", median_value)

print("Mode: ", mode_value)

print("Standard Deviation: ", std_deviation)
```

Output:

```
%runfile C:/Users/admin/untitled0.py --wdir
Data:  [1, 2, 2, 3, 4, 5, 5, 5, 6]
Mean:  3.6666666666666665
Median:  4
Mode:  5
Standard Deviation:  1.7320508075688772
```

Ex No: 6

Date : 10-02-25

Data Pre-Processing

Aim: To Demonstrate various data pre-processing techniques for a given dataset such as Reshaping the data, Filtering the data.

Procedure:

1. **Import Library** — Imports the `statistics` module for statistical calculations.
2. **Define Data** — Creates a list named `data` containing sample numeric values.
3. **Calculate Mean** — Uses `statistics.mean()` to compute the average of the dataset.
4. **Calculate Median** — Uses `statistics.median()` to compute the middle value of the sorted dataset.
5. **Calculate Mode** — Uses `statistics.mode()` to identify the most frequently occurring value in the dataset.
6. **Calculate Standard Deviation** — Uses `statistics.stdev()` to compute the sample standard deviation.
7. **Display Results** — Prints the dataset along with the calculated mean, median, mode, and standard deviation.

Code:

```
import pandas as pd

import numpy as np

# Sample dataset creation

data = {

    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],

    'Age': [25, 30, 35, 40, 45],

    'Salary': [50000, 60000, 70000, 80000, 90000],

    'Department': ['HR', 'IT', 'Finance', 'IT', 'HR']

}

# Creating a DataFrame

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)
```

```
# 1. Reshaping Data (Melt)

df_melted = pd.melt(df, id_vars=['Name'], value_vars=['Age', 'Salary'],
var_name='Attribute', value_name='Value')

print("\nMelted DataFrame:\n", df_melted)


# 2. Reshaping Data (Pivot)

df_pivot = df_melted.pivot_table(index='Name', columns='Attribute',
values='Value').reset_index()

print("\nPivoted DataFrame:\n", df_pivot)


# 3. Filtering Data - Rows where Age > 30

filtered_df = df[df['Age'] > 30]

print("\nFiltered Data (Age > 30):\n", filtered_df)


# 4. Filtering Data - Employees in 'IT' Department

it_department_df = df[df['Department'] == 'IT']

print("\nFiltered Data (IT Department):\n", it_department_df)
```

Output:

```
Original DataFrame:
      Name  Age  Salary Department
0    Alice   25   50000         HR
1      Bob   30   60000         IT
2  Charlie   35   70000    Finance
3    David   40   80000         IT
4      Eva   45   90000         HR


Melted DataFrame:
      Name Attribute  Value
0    Alice       Age     25
```

```
1      Bob       Age     30
2  Charlie       Age     35
3    David       Age     40
4      Eva       Age     45
5    Alice    Salary  50000
6      Bob    Salary  60000
7  Charlie    Salary  70000
8    David    Salary  80000
9      Eva    Salary  90000

Pivoted DataFrame:
 Attribute     Name   Age   Salary
0            Alice  25.0  50000.0
1              Bob  30.0  60000.0
2          Charlie  35.0  70000.0
3            David  40.0  80000.0
4              Eva  45.0  90000.0

Filtered Data (Age > 30):
      Name  Age  Salary Department
2  Charlie   35   70000    Finance
3    David   40   80000         IT
4      Eva   45   90000         HR

Filtered Data (IT Department):
    Name  Age  Salary Department
1    Bob   30   60000         IT
3  David   40   80000         IT
```

Ex No : 7 **Date :** 18-02-25	**Data Preprocessing**

Aim: To demonstrate merging, handling missing value and feature Normalization

Procedure:

1. **Import Libraries**
   - Import `pandas` for data manipulation.
   - Import `numpy` for handling missing values like `NaN`.
2. **Create Sample Datasets**
   - Define `data1` with columns: `ID`, `Name`, `Age`, and `Salary`.
   - Define `data2` with columns: `ID`, `Department`, and `Experience`.
3. **Create DataFrames**
   - Use `pd.DataFrame()` to convert the dictionaries (`data1` and `data2`) into DataFrames `df1` and `df2`.
4. **Display Original DataFrames**
   - Print the original DataFrames to observe their content.
5. **Merge DataFrames**
   - Use `pd.merge()` with `how='outer'` to perform an outer join, ensuring all rows from both DataFrames are included.
6. **Handle Missing Values**
   - Use `.fillna()` to replace:
     - Missing values in `Age` with the mean age.
     - Missing values in `Salary` with the mean salary.
     - Missing values in `Department` with `"Unknown"`.
     - Missing values in `Experience` with `0`.
7. **Feature Normalization (Min-Max Normalization)**
   - Compute normalized values for the `Salary` column
   - Assign the normalized values to a new column named `Salary_Normalized`.
8. **Display Results**
   - Print the merged DataFrame, the DataFrame after handling missing values, and the final DataFrame with normalized salary values.

Code:

```
import pandas as pd

import numpy as np


# Sample dataset creation

data1 = {
```

```
    'ID': [1, 2, 3, 4, 5],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Age': [25, 30, 35, np.nan, 45],
    'Salary': [50000, 60000, np.nan, 80000, 90000]
}

data2 = {
    'ID': [3, 4, 5, 6, 7],
    'Department': ['Finance', 'IT', 'HR', 'Marketing', 'Sales'],
    'Experience': [5, 7, 10, 2, 3]
}

# Creating DataFrames
df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)
print("Original DataFrame 1:\n", df1)
print("\nOriginal DataFrame 2:\n", df2)

# 1. Merging Data
df_merged = pd.merge(df1, df2, on='ID', how='outer')
print("\nMerged DataFrame:\n", df_merged)

# 2. Handling Missing Values
df_merged.fillna({
    'Age': df_merged['Age'].mean(),
    'Salary': df_merged['Salary'].mean(),
    'Department': 'Unknown',
    'Experience': 0
}, inplace=True)
print("\nDataFrame after Handling Missing Values:\n", df_merged)
```

```python
# 3. Feature Normalization - Min-Max Normalization
df_merged['Salary_Normalized'] = (df_merged['Salary'] - df_merged['Salary'].min()) / (df_merged['Salary'].max() - df_merged['Salary'].min())
print("\nDataFrame after Min-Max Normalization:\n", df_merged)
```

Output:

```
Original DataFrame 1:
   ID     Name   Age   Salary
0   1    Alice  25.0  50000.0
1   2      Bob  30.0  60000.0
2   3  Charlie  35.0      NaN
3   4    David   NaN  80000.0
4   5      Eva  45.0  90000.0

Original DataFrame 2:
   ID Department  Experience
0   3    Finance           5
1   4         IT           7
2   5         HR          10
3   6  Marketing           2
4   7      Sales           3

Merged DataFrame:
   ID     Name   Age   Salary Department  Experience
0   1    Alice  25.0  50000.0        NaN         NaN
1   2      Bob  30.0  60000.0        NaN         NaN
2   3  Charlie  35.0      NaN    Finance         5.0
3   4    David   NaN  80000.0         IT         7.0
4   5      Eva  45.0  90000.0         HR        10.0
5   6      NaN   NaN      NaN  Marketing         2.0
```

```
6   7     NaN   NaN     NaN     Sales        3.0

DataFrame after Handling Missing Values:
   ID     Name    Age   Salary Department  Experience
0   1    Alice  25.00  50000.0    Unknown        0.0
1   2      Bob  30.00  60000.0    Unknown        0.0
2   3  Charlie  35.00  70000.0    Finance        5.0
3   4    David  33.75  80000.0         IT        7.0
4   5      Eva  45.00  90000.0         HR       10.0
5   6      NaN  33.75  70000.0  Marketing        2.0
6   7      NaN  33.75  70000.0      Sales        3.0

DataFrame after Min-Max Normalization:
   ID     Name    Age   Salary Department  Experience  Salary_Normalized
0   1    Alice  25.00  50000.0    Unknown        0.0               0.00
1   2      Bob  30.00  60000.0    Unknown        0.0               0.25
2   3  Charlie  35.00  70000.0    Finance        5.0               0.50
3   4    David  33.75  80000.0         IT        7.0               0.75
4   5      Eva  45.00  90000.0         HR       10.0               1.00
5   6      NaN  33.75  70000.0  Marketing        2.0               0.50
6   7      NaN  33.75  70000.0      Sales        3.0               0.50
```

Ex No: 8 **Date:** 25-02-25	**Naïve Bayes Classifier**

Aim: To implement the Naïve Bayes Classifier on the isis dataset and check for Accuracy

Procedure:

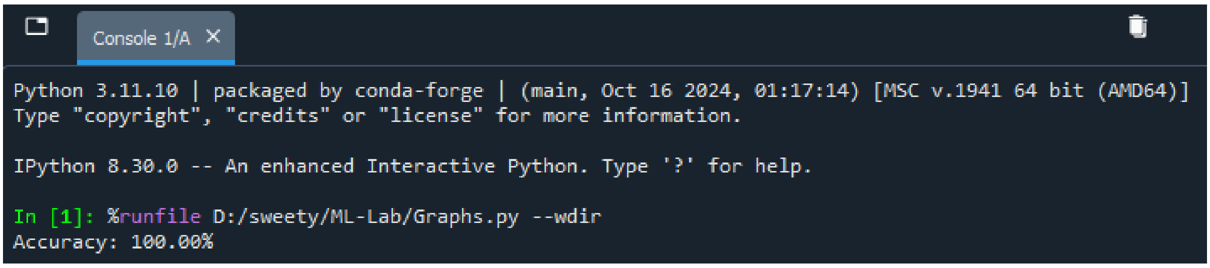
1. **Import Libraries**
   - Import the following from `sklearn`:
     - `load_iris` for loading the Iris dataset.
     - `train_test_split` for splitting the data into training and testing sets.
     - `GaussianNB` from `sklearn.naive_bayes` for the Naive Bayes classifier.
     - `accuracy_score` for evaluating the model's performance.
2. **Load the Dataset**
   - Use `load_iris()` to load the Iris dataset into the variable `iris`.
   - Extract the feature data (`iris.data`) into `X` and the target labels (`iris.target`) into `y`.
3. **Split the Dataset**
   - Use `train_test_split()` to divide the dataset:
     - `X_train` and `y_train` for training.
     - `X_test` and `y_test` for testing.
   - Specify `test_size=0.2` to allocate 20% of the data for testing.
   - Use `random_state=42` to ensure reproducibility.
4. **Initialize the Classifier**
   - Create an instance of `GaussianNB()` and assign it to `nb_classifier`.
5. **Train the Classifier**
   - Use `.fit()` to train the model using the training data (`X_train`, `y_train`).
6. **Make Predictions**
   - Use `.predict()` to generate predictions on the test set (`X_test`).
7. **Evaluate the Model**
   - Calculate the model's accuracy using `accuracy_score()`.
   - Print the accuracy value as a percentage with two decimal points for clarity.

Program:

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score

# Load the dataset

iris = load_iris()
```

```
X, y = iris.data, iris.target

# Split the dataset into training and testing sets

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

# Initialize the Naive Bayes classifier

nb_classifier = GaussianNB()

# Train the classifier

nb_classifier.fit(X_train, y_train)

# Make predictions

y_pred = nb_classifier.predict(X_test)

# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy * 100:.2f}%")
```

OUTPUT:



```
Console 1/A

Python 3.11.10 | packaged by conda-forge | (main, Oct 16 2024, 01:17:14) [MSC v.1941 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.30.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %runfile D:/sweety/ML-Lab/Graphs.py --wdir
Accuracy: 100.00%
```

Ex. No: 9

Date: 4-03-25

Linear Regression

Aim: To implement Linear Regression to predict house price using the California Housing Dataset

Procedure:

1. **Import Required Libraries**
   - Use libraries like pandas, numpy, matplotlib, seaborn, and sklearn for data manipulation, visualization, and model building.
2. **Load the Dataset**
   - Use fetch_california_housing() to load the California housing data.
   - Convert the dataset into a DataFrame and add the target variable (Price).
3. **Explore the Dataset**
   - Display the first few rows to understand the data structure.
   - Identify key features like MedInc, HouseAge, AveRooms, etc.
4. **Define Features (X) and Target (y)**
   - Separate the independent variables (features) and the dependent variable (Price).
5. **Split Data into Training and Testing Sets**
   - Use train_test_split() to divide the data (e.g., 80% training, 20% testing).
   - Set random_state for reproducibility.
6. **Initialize and Train the Linear Regression Model**
   - Create a LinearRegression() model instance.
   - Fit the model using the training data.
7. **Predict Housing Prices**
   - Use the trained model to predict house prices on the test data.
8. **Evaluate the Model**
   - Calculate evaluation metrics:

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R^2 Score

9. **Visualize Actual vs Predicted Prices**
   - Use a scatterplot to compare predicted prices against actual values.
   - Add a red dashed line (y = x) to indicate perfect predictions.

10. **Interpret Results**

- A strong alignment with the red dashed line suggests good model performance.
- Significant scatter may indicate model limitations or noisy data.

Program:

```
import matplotlib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.datasets import fetch_california_housing  # Newer housing dataset

# Load California Housing Dataset
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
df["Price"] = data.target  # Target variable
# Display dataset info
print(df.head())
# Define features (X) and target (y)
X = df.drop(columns=["Price"])
```

```
y = df["Price"]
# Split data into training & testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Initialize and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Predict on test data
y_pred = model.predict(X_test)
# Model Evaluation
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R² Score: {r2:.2f}")
# Plot actual vs predicted prices
# Plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test.to_numpy(), y=y_pred, alpha=0.6)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',
linestyle="--")
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Housing Prices")
plt.show()
```

Output:

Actual vs Predicted Housing Prices
Predicted Prices
Actual Prices
12
10
8
6
4
2
0
0
1
2
3
4
5

Ex. No: 10 **Date:** 11-03-25	**Logistic Regression**

Aim : To predict breast cancer using Logistic Regression

Procedure:

1. **Import Required Libraries**
   - Import libraries for data manipulation (pandas, numpy), visualization (matplotlib, seaborn), and model building (sklearn).
2. **Load the Dataset**
   - Load the Breast Cancer dataset using load_breast_cancer().
   - Convert the dataset into a DataFrame and add the target variable.
3. **Feature Selection**
   - Separate the dataset into:
     - Features (X): Independent variables (all columns except the target).
     - Target (y): Dependent variable indicating cancer presence.
4. **Data Visualization**
   - Use a scatterplot to visualize the relationship between mean radius and mean texture, with different colors representing the target classes (malignant/benign).
5. **Split the Data into Training and Testing Sets**
   - Use train_test_split() to split the data into training (70%) and testing (30%) sets.
   - Set random_state for reproducibility.
6. **Initialize and Train the Logistic Regression Model**
   - Create a LogisticRegression model with increased max_iter to ensure convergence.
   - Fit the model using the training data.
7. **Make Predictions**
   - Use the trained model to predict outcomes on the test data.

8. **Evaluate the Model**
    - Calculate the Accuracy Score to measure the model's overall performance.
    - Display the Classification Report to examine precision, recall, and F1-score for each class.
9. **Visualize the Confusion Matrix**
    - Use a heatmap to display the confusion matrix, showing true positives, true negatives, false positives, and false negatives.
10. **Interpret Results**

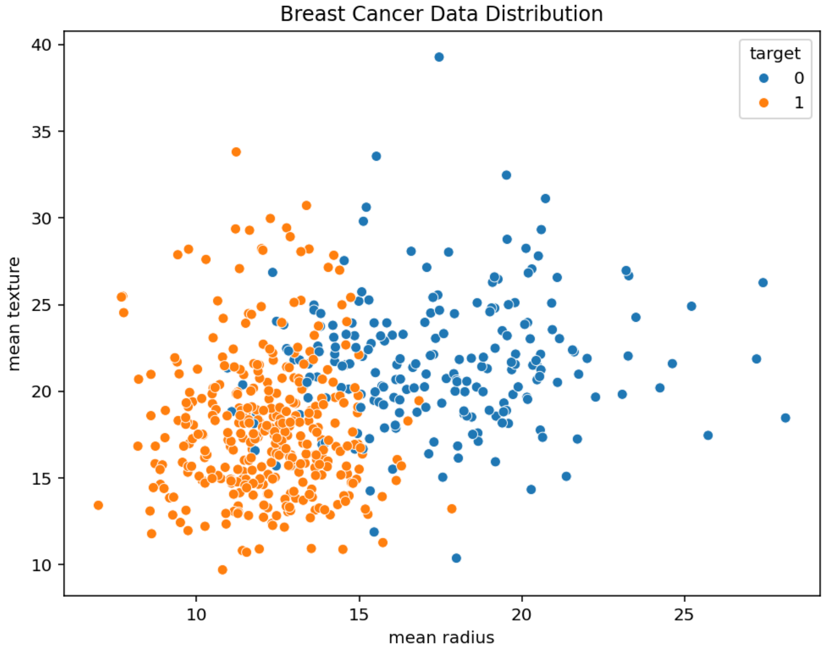
- A higher accuracy score and well-balanced precision/recall values indicate a robust model.
- The confusion matrix helps identify common misclassifications and potential improvements**.**

Program :

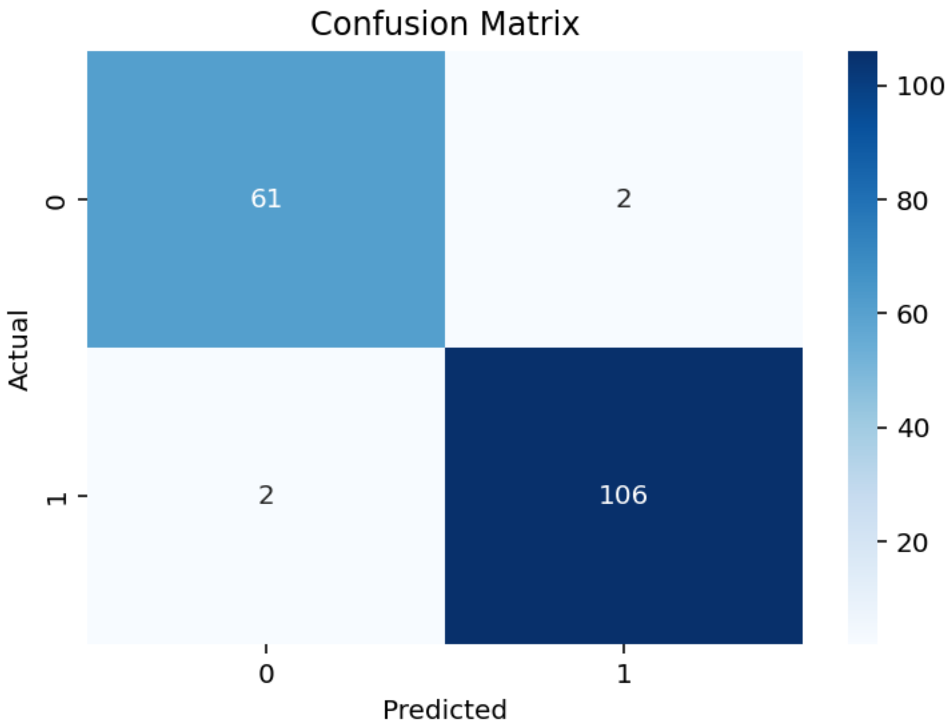
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the Breast Cancer dataset
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
# Feature selection
X = df.drop('target', axis=1)
y = df['target']
# Data visualization
plt.figure(figsize=(8, 6))
```

```
sns.scatterplot(x='mean radius', y='mean texture', hue='target', data=df)

plt.title("Breast Cancer Data Distribution")

plt.show()

# Split the data into training and testing sets

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

# Initialize and train the Logistic Regression model

model = LogisticRegression(max_iter=5000)

model.fit(X_train, y_train)

# Predictions

y_pred = model.predict(X_test)

# Evaluate the model

print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")

print("Classification Report:")

print(classification_report(y_test, y_pred))


# Confusion Matrix Visualization

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

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()
```

Output:



Breast Cancer Data Distribution
target
0
1
mean texture
mean radius
40
35
30
25
20
15
10
10
15
20
25



Confusion Matrix
61
2
2
106
Actual
Predicted
0
1
0
1
100
80
60
40
20

Ex. No: 11

Date: 14-03-25

Decision Tree

Aim: To use decision tree for predicting risk of heart disease using the Cleveland heart disease dataset

Procedure:

1. **Import Required Libraries**
   - Use pandas and numpy for data manipulation.
   - Use matplotlib for visualization.
   - Import train_test_split, DecisionTreeClassifier, plot_tree, and performance metrics from sklearn.
2. **Load the Heart Disease Dataset**
   - Load the dataset from the provided URL.
   - Assign appropriate column names based on the dataset's structure.
3. **Data Cleaning**
   - Replace missing values represented by '?' with NaN.
   - Drop rows with missing values to ensure data quality.
4. **Target Conversion for Binary Classification**
   - The original target values indicate different stages of heart disease.
   - Convert the target column into binary form:
     - 1 for the presence of heart disease.
     - 0 for the absence of heart disease.
5. **Feature Selection**
   - Split the dataset into:
     - Features (X): All columns except the target.
     - Target (y): The binary target variable.
6. **Data Splitting**
   - Split the dataset into training (70%) and testing (30%) sets.
   - Use random_state to ensure reproducibility.
7. **Initialize and Train the Decision Tree Model**

- Instantiate a DecisionTreeClassifier.
- Train the model using the training data.

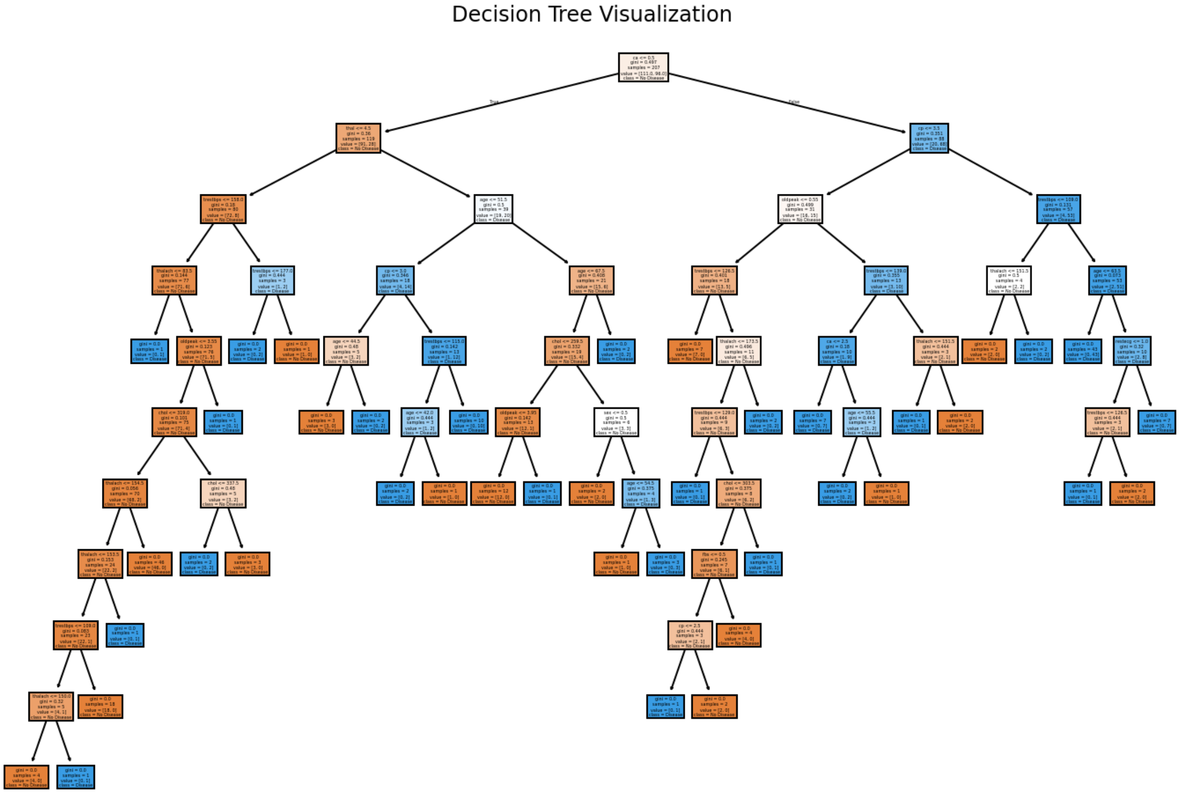
8. **Make Predictions**
    - **Use the trained model to predict outcomes on the test set.**
9. **Evaluate the Model**
    - Calculate the Accuracy Score to assess the overall performance.
    - Display the Classification Report to review precision, recall, and F1-score for each class.
10. **Visualize the Decision Tree**
    - Use plot_tree() to visualize the decision tree structure.
    - Display feature names and class labels for better interpretability.
11. **Interpret Results**
    - A high accuracy score with balanced precision/recall values indicates a well-performing model.
    - The decision tree visualization helps understand the model's decision logic.

Program :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report


# Load the Heart Disease dataset
df = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data", header=None)
df.columns = ["age", "sex", "cp", "trestbps", "chol", "fbs", "restecg", "thalach", "exang", "oldpeak", "slope", "ca", "thal", "target"]


# Clean the dataset by replacing '?' with NaN and dropping rows with missing values
df.replace("?", np.nan, inplace=True)
```

```
df.dropna(inplace=True)

# Convert target into binary classification (presence of heart disease: 1, absence: 0)
df['target'] = (df['target'] > 0).astype(int)

# Feature selection
X = df.drop('target', axis=1)
y = df['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Initialize and train the Decision Tree model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate the model
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Visualizing the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(model, feature_names=X.columns, class_names=['No Disease', 'Disease'],
filled=True)
plt.title("Decision Tree Visualization")
plt.show()
```

Output:




```
In [5]: %runfile D:/sweety/ML-Lab/untitled1.py --wdir
Accuracy: 0.69
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.71      0.71        49
           1       0.66      0.66      0.66        41

    accuracy                           0.69        90
   macro avg       0.69      0.69      0.69        90
weighted avg       0.69      0.69      0.69        90
```

Ex. No: 12 **Date:** 18-03-25	**KNN Algorithm**

Aim :

To implement a K-Nearest Neighbors (KNN) classifier using the Iris dataset for predicting flower species based on feature measurements.

Procedure:

1. **Import Libraries:**
   Import essential libraries such as numpy, pandas, matplotlib, seaborn, and scikit-learn modules for data manipulation, visualization, and model building.
2. **Load the Dataset:**
   Load the built-in Iris dataset using load_iris() from sklearn.datasets. The dataset contains features such as sepal length, sepal width, petal length, and petal width, along with the target class representing three flower species.
3. **Create a DataFrame:**
   Convert the dataset into a pandas DataFrame and add the target column to the DataFrame for easier handling.
4. **Feature Selection:**
   Define X (features) by excluding the target column, and define y (target) as the target column.
5. **Split the Data:**
   Split the data into training and testing sets using train_test_split() with a 70:30 ratio to train and evaluate the model effectively.
6. **Initialize and Train the Model:**
   Create a KNeighborsClassifier model with n_neighbors=5 and fit it using the training data.
7. **Make Predictions:**
   Use the trained model to predict labels for the test data.
8. **Evaluate the Model:**
   - Calculate the accuracy score to assess the model's performance.
   - Display the classification report showing precision, recall, and F1-score for each class.
9. **Visualize the Confusion Matrix:**
   - Use Seaborn's heatmap() to plot the confusion matrix.

- Label the axes as Actual and Predicted for clarity.

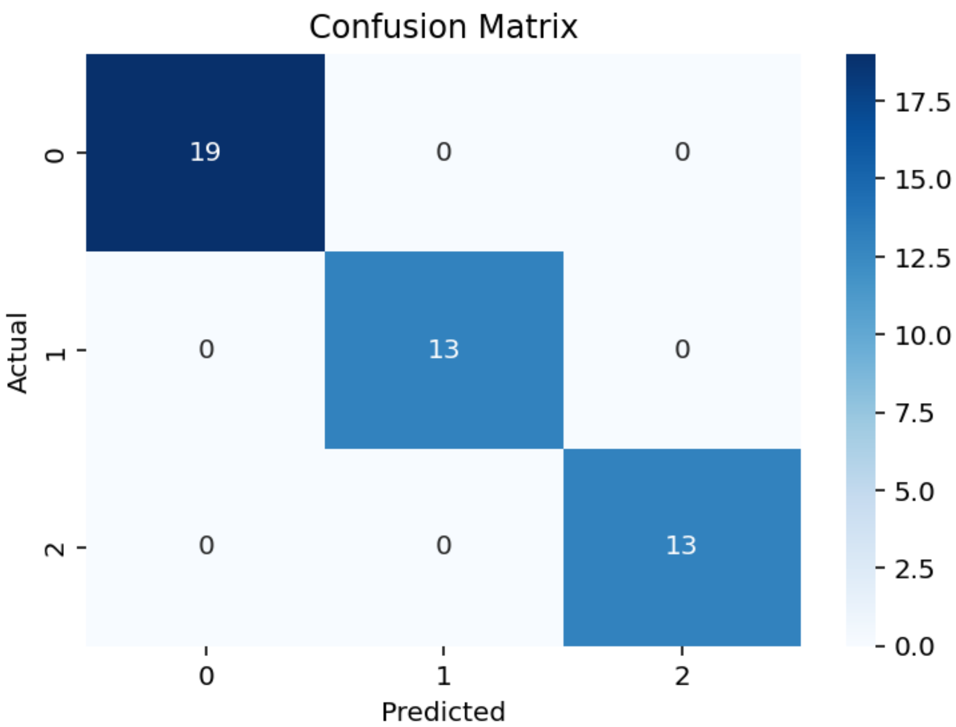
10. **Conclusion:**
    The confusion matrix helps visualize the model's performance by showing correctly and incorrectly predicted labels, aiding in better understanding and further model tuning.

Program:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.datasets import load_iris

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

import seaborn as sns


# Load the built-in Iris dataset

data = load_iris()

df = pd.DataFrame(data.data, columns=data.feature_names)

df['target'] = data.target


# Feature selection

X = df.drop('target', axis=1)

y = df['target']


# Split the data into training and testing sets

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


# Initialize and train the KNN model

model = KNeighborsClassifier(n_neighbors=5)

model.fit(X_train, y_train)
```

```
# Predictions
y_pred = model.predict(X_test)

# Evaluate the model
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion Matrix Visualization
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='Blues', fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Output :



```
In [7]: %runfile D:/sweety/ML-Lab/untitled2.py --wdir
Accuracy: 1.00
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
```

Ex. No: 13 **Date:** 20-03-25	**K- Mean Clustering**

Aim : To implement K-Means clustering on the built-in Wine dataset to group data points based on their feature similarities.

Procedure:

1. **Import Required Libraries**
   - Use pandas and numpy for data manipulation.
   - Use matplotlib for visualization.
   - Import KMeans from sklearn for clustering and StandardScaler for data standardization.
2. **Load the Wine Dataset**
   - Load the built-in Wine dataset using load_wine() from sklearn.
   - Convert the dataset into a DataFrame and assign appropriate feature names.
3. **Data Standardization**
   - Since K-Means is sensitive to feature scales, standardize the dataset using StandardScaler().
   - Fit and transform the data to ensure all features have a mean of zero and a standard deviation of one.
4. **Apply K-Means Clustering**
   - Initialize a KMeans model with n_clusters=3 (since the Wine dataset has three classes).
   - Use random_state=42 for reproducibility.
   - Set n_init=10 to ensure stable clustering results by performing multiple initializations.
5. **Assign Cluster Labels**
   - Use fit_predict() to train the K-Means model and assign each data point to a cluster.
   - Add the predicted cluster labels as a new column in the DataFrame.
6. **Visualize the Clusters**
   - Create a scatter plot of the first two features to visualize data distribution.
   - Color each point based on its assigned cluster.

- Plot the cluster centroids as red 'X' markers to show the cluster centers.

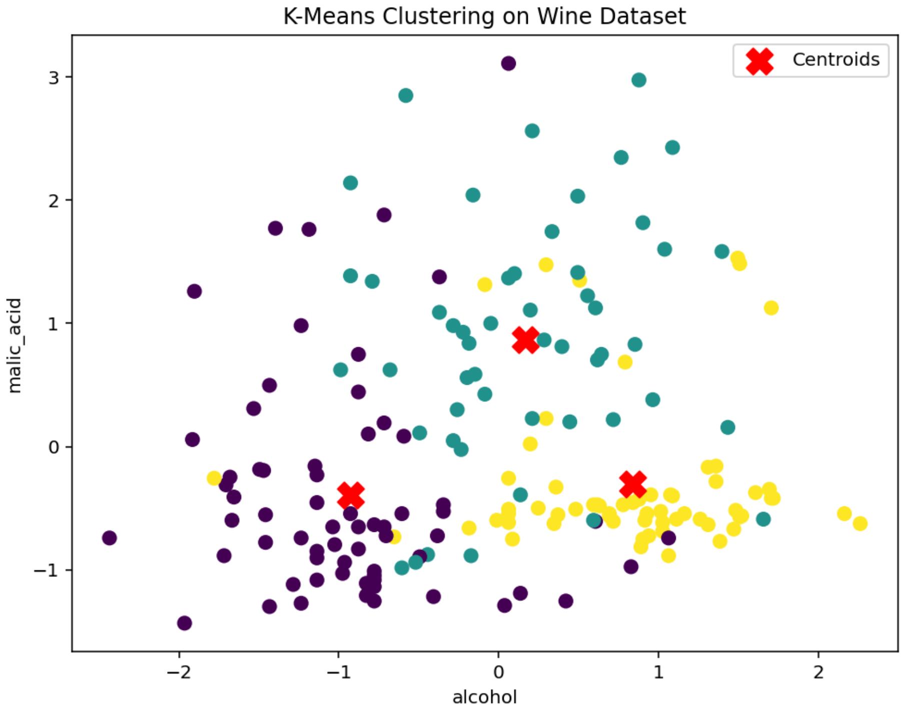
7. **Add Descriptive Details**
    - Add appropriate labels for the X-axis, Y-axis, and plot title.
    - Include a legend to distinguish the centroids from the data points.
8. **Interpret Results**
    - Observe the clustering pattern to evaluate how well the data points are grouped.
    - Note that since K-Means is an unsupervised method, the clusters may not directly match the original class labels but can reveal meaningful groupings.

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler

# Load the built-in Wine dataset
data = load_wine()
df = pd.DataFrame(data.data, columns=data.feature_names)

# Standardizing the data for better clustering performance
scaler = StandardScaler()
X = scaler.fit_transform(df)

# Applying K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df['cluster'] = kmeans.fit_predict(X)

# Visualizing the clusters
```

```
plt.figure(figsize=(8, 6))

plt.scatter(X[:, 0], X[:, 1], c=df['cluster'], cmap='viridis', s=50)

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=200, c='red', marker='X', label='Centroids')

plt.title('K-Means Clustering on Wine Dataset')

plt.xlabel(data.feature_names[0])

plt.ylabel(data.feature_names[1])

plt.legend()

plt.show()
```

Output:



Ex No: 12 **Date :** 21 -03-24	**Spam Mail Detection**

Aim: To develop a spam email detection system using the Naive Bayes algorithm with visualizations for data distribution and model evaluation.

Procedure:

1. **Import Libraries:** Import necessary libraries such as pandas, seaborn, matplotlib, and scikit-learn for data handling, visualization, and model building.
2. **Prepare the Dataset:** Create a sample dataset with email messages labeled as 'spam' or 'ham'.
3. **Encode Labels:** Convert the 'ham' and 'spam' labels into binary values (0 for ham, 1 for spam).
4. **Visualize Data Distribution:** Use a bar plot to visualize the proportion of spam vs ham messages.
5. **Text Vectorization:** Use CountVectorizer to transform email messages into numerical features.
6. **Split the Data:** Divide the dataset into training and testing sets for model evaluation.
7. **Train the Model:** Fit a **Multinomial Naive Bayes** classifier on the training data.
8. **Make Predictions:** Predict labels for the test data.

Program:

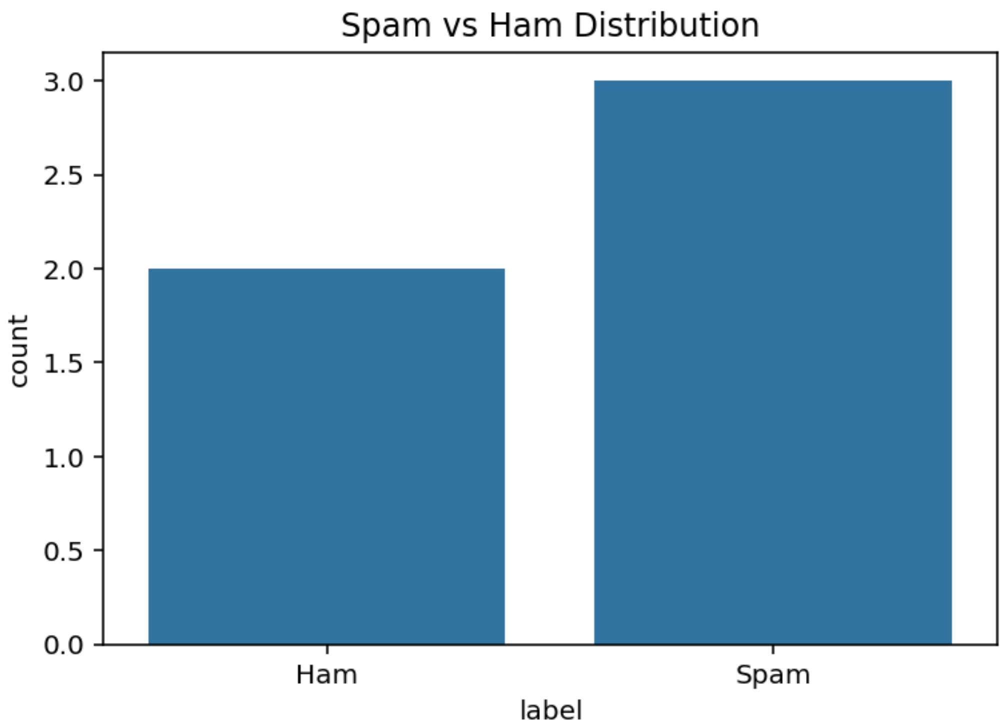
```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, confusion_matrix


# Sample email data (simplified dataset)

data = {

    'message': [
```

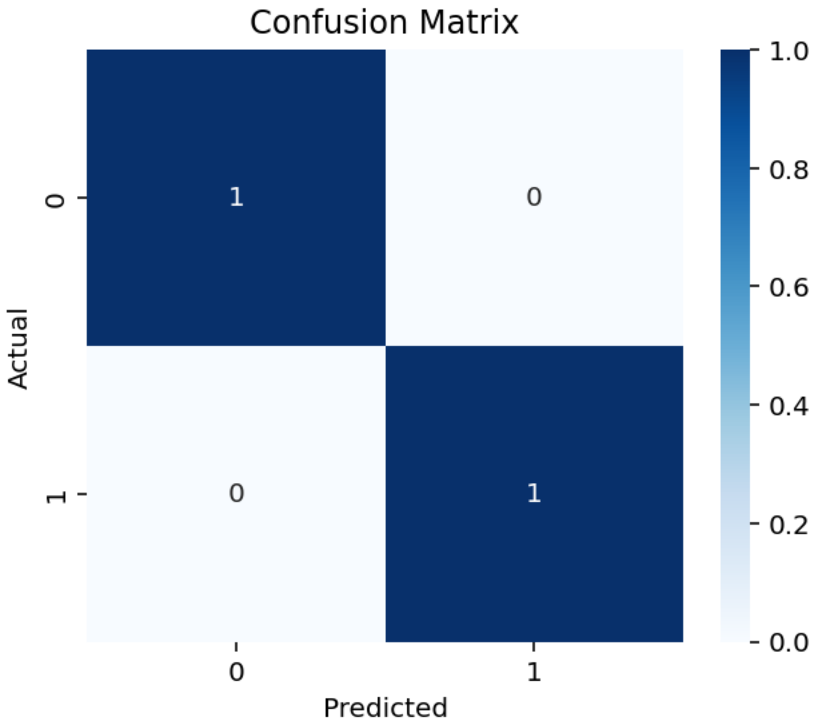
```
        "Win a brand new car! Click here now.",
        "Meeting at 3 PM. Don't forget the documents.",
        "You've won a $1000 gift card! Claim your prize.",
        "Hey, let's catch up this weekend.",
        "Urgent! Your account has been compromised. Act now!"
    ],
    'label': ['spam', 'ham', 'spam', 'ham', 'spam']
}

# Create a DataFrame
df = pd.DataFrame(data)

# Encode labels (ham = 0, spam = 1)
df['label'] = df['label'].map({'ham': 0, 'spam': 1})

# Data Visualization - Bar Plot
plt.figure(figsize=(6, 4))
sns.countplot(x='label', data=df)
plt.xticks([0, 1], ['Ham', 'Spam'])
plt.title("Spam vs Ham Distribution")
plt.show()

# Text vectorization
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['message'])
y = df['label']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

```python
# Train the Naive Bayes model
model = MultinomialNB()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Display accuracy
print(f"Accuracy: {accuracy_score(y_test, y_pred) * 100:.2f}%")

# Confusion Matrix Visualization
plt.figure(figsize=(5, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='Blues', fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Example prediction
sample_message = ["Congratulations! You've won a free vacation!"]
sample_vector = vectorizer.transform(sample_message)
prediction = model.predict(sample_vector)
print(f"Prediction for sample message: {'Spam' if prediction[0] == 1 else 'Ham'}")
```

Output:

```
Accuracy: 100.00%
Prediction for sample message: Spam
```


Spam vs Ham Distribution
3.0
2.5
2.0
1.5
1.0
0.5
0.0
count
Ham
Spam
label


Confusion Matrix
1
0
0
1
Actual
0
1
0
1
Predicted
1.0
0.8
0.6
0.4
0.2
0.0