Dewan VS Institute of Engineering Technology Meerut

Department of Artificial Intelligence

Academic Year 2025

(Session 2025-26)

DEWAN VS GROUP OF INSTITUTIONS INDIA

DATA ANALYTICS LAB

(6th Semester AI-DS)

TABLE OF CONTENTS

UNIVERSITY SYLLABUS

S. NO.	LIST OF EXPERIMENT	COURSE OUTCOME
1.	Write a program to implement following numerical operations in Python : Math functions such as MAX, MIN, AVG, SUM, SQRT, ROUND etc	CO1
2.	Write a program to implement following statistical operations in Python: Mean, Median, Mode and Standard deviation	CO1
3.	Write a program to perform import/export (.CSV, .XLS, .TXT) and other data handling operations using data frames in Python	CO1
4.	Write a program to perform following data pre-processing operations on various datasets in Python: a. Handling Missing data b. Min-Max normalization c. Lemmatization d. Stop word Removal etc	CO2
5.	To perform dimensionality reduction operation using PCA and compare the performance of SVM classifier with or without dimensionality reduction on Diabetes dataset.	CO2
6.	To perform dimensionality reduction using Principal Component Analysis (PCA) and compare the performance of the Support Vector Machine (SVM) classifier with and without dimensionality reduction on the Diabetes dataset.	C03
7.	To perform K-Means clustering on the Country-data dataset and evaluate the clustering results using data visualization and the elbow method.	C04
8.	Write Python script to perform market basket analysis using Association Rules (Apriori).	C04
9.	Write python script to perform classification algorithms such as Logistic Regression, KNN, Naïve Bayes, and Support Vector Machine and visualize the result on different datasets such as Iris, textual, image datasets etc.	C05
10.	Write a Python script to collect review data via web-scraping, APIs and data connectors from different social media sites and display the most important words using visualization tool such as word cloud.	C02

Course Outcomes (COs)

Upon the successful completion of the course, the students will be able to

CO1: Implement numerical and statistical analysis on various data sources.

CO2: Apply data preprocessing and dimensionality reduction methods on raw data.

CO3: Implement linear regression technique on numeric data for prediction.

CO4: Execute clustering and association rule mining algorithms on different datasets.

CO5: Implement and evaluate the performance of KNN algorithm on different datasets.

CO-PO mapping

LIST OF EXPERIMENT

S. No.	Practical's Name	Date	Sign
1.	Write a program to implement following numerical operations in Python: Math functions such as MAX, MIN, AVG, SUM, SQRT, ROUND etc.		
2.	Write a program to implement following statistical operations in Python: Mean, Median, Mode and Standard deviation		
3.	Write a program to perform import/export (.CSV, .XLS, .TXT) and other data handling operations using data frames in Python		
4.	Write a program to perform following data pre-processing operations on various datasets in Python: a. Handling Missing data b. Min-Max normalization c. Lemmatization d. Stop word Removal etc		
5.	Write a program to implement support vector machines and principal component analysis.		
6.	To perform dimensionality reduction using Principal Component Analysis (PCA) and compare the performance of the Support Vector Machine (SVM) classifier with and without dimensionality reduction on the Diabetes dataset.		
7.	To perform K-Means clustering on the Country-data dataset and evaluate the clustering results using data visualization and the elbow method.		
8.	Write Python script to perform market basket analysis using Association Rules (Apriori).		
9.	Write python script to perform classification algorithms such as Logistic Regression, KNN, Naïve Bayes, and Support Vector Machine and visualize the result on different datasets such as Iris, textual, image datasets etc.		
10.	Write a Python script to collect review data via web-scraping, APIs and data connectors from different social media sites and display the most important words using visualization tool such as word cloud.		

Experiment No. 1

Objective: To get the input from user and perform numerical operations (MAX, MIN, AVG, SUM, SQRT, ROUND) using in Python.

Description: This experiment aims to demonstrate how to accept numerical input from a user in Python and perform basic mathematical operations such as finding the maximum, minimum, average (mean), and sum of the input values. Additionally, it calculates the square root of each number using the math module and rounds both the original numbers and their square roots to the nearest whole number or to two decimal places. The input is taken as a comma-separated string, converted into a list of floating-point numbers, and processed using built-in Python functions like max(), min(), sum(), and round(). The "Numerical Operations Flowchart" in figure 1.1, illustrates the steps to input a number and perform various operations such as MAX, MIN, AVG, SUM, SQRT, and ROUND. It guides the user through decision points to choose the desired operation and display the result.

Apparatus Used:

Python Programming Language – For writing and executing the code.
Python IDE or Code Editor – Such as Google Colab, Jupyter Notebook.
Standard Input/output Devices – Keyboard for input, Monitor for output display.
Python Libraries –

- math module (for square root calculation)
- Built-in functions (max (), min (), sum (), round ())

Algorithm: To Perform Numerical Operations

1. Start
2. Prompt the user to enter a list of numbers separated by commas
3. Read the input from the user as a string
4. Split the input string using commas to extract individual number strings
5. Convert each extracted string into a float and store them in a list
6. Perform the following operations on the list of numbers:

a. Find the maximum number using max()
b. Find the minimum number using min()
c. Calculate the sum using sum()
d. Compute the average by dividing the sum by the total number of elements
e. Compute the square root of each number using math.sqrt()
f. Round each number and its square root using round()

7. Display all the computed results
8. End

Flowchart: -

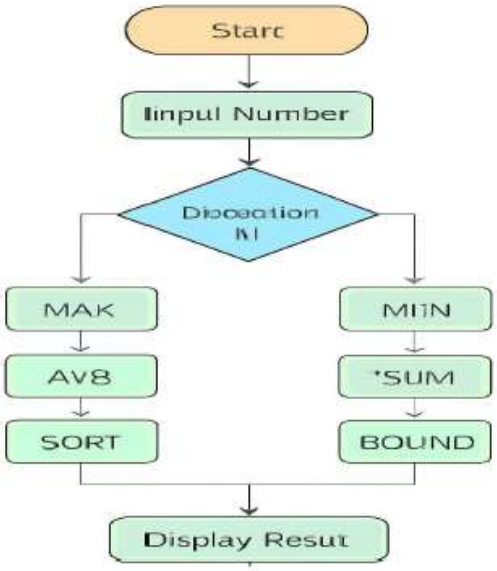

Figure 1.1: Numerical Operations Flowchart

Screenshot:

2/5/26, 11:21 AM Experiment01.ipynb - Colab

```
nums = input("Enter numbers separated by spaces: ")

numbers = list(map(float, nums.split()))

# Operations
maximum = max(numbers)
minimum = min(numbers)
total = sum(numbers)
average = total / len(numbers)
sorted_numbers = sorted(numbers)
rounded_numbers = [round(num) for num in numbers]

# Output results
print("\nResults:")
print("Numbers entered:", numbers)
print("Maximum:", maximum)
print("Minimum:", minimum)
print("Sum:", total)
print("Average:", average)
print("Sorted List:", sorted_numbers)
print("Rounded Numbers:", rounded_numbers)
```

Expected Output:

```
Enter numbers separated by spaces: 10.5 2.3 7.8 4.1

Results:
Numbers entered: [10.5, 2.3, 7.8, 4.1]
Maximum: 10.5
Minimum: 2.3
Sum: 24.7
Average: 6.175
Sorted List: [2.3, 4.1, 7.8, 10.5]
Rounded Numbers: [10, 2, 8, 4]
```

Program Code:

```
nums = input("Enter numbers separated by spaces: ")
numbers = list(map(float, nums.split()))
# Operations
maximum = max(numbers)
minimum = min(numbers)
total = sum(numbers)
average = total / len(numbers)
sorted_numbers = sorted(numbers)
rounded_numbers = [round(num) for num in numbers]
```

Output results

```
print("\nResults:")
print("Numbers entered:", numbers)
print("Maximum:", maximum)
print("Minimum:", minimum)
print("Sum:", total)
print("Average:", average)
print("Sorted List:", sorted_numbers)
print("Rounded Numbers:", rounded_numbers)
```

Experiment No. 2

Objective: Write a program in Python to implement following statistical operations: Mean, Median, Mode and Standard deviation.

Algorithm:
Step-By-Step Solution
Step 1
Import the required module statistics.
Step 2
Define the dataset.
Step 3
Calculate mean, mode, median, standard deviation, and variance using statistics functions.
Step 4
Print the results.

Program Code 1:

```
import statistics
data = [4, 8, 6, 5, 3, 2, 8, 9, 2, 5, 4, 4, 9, 23, 6]
mean = statistics.mean(data)
mode = statistics.mode(data)
median = statistics.median(data)
std_dev = statistics.stdev(data)
variance = statistics.variance(data)
print(f"Mean: {mean}")
print(f"Mode: {mode}")
print(f"Median: {median}")
print(f"Standard Deviation: {std_dev}")
print(f"Variance: {variance}")
```

Program Code 2:

```
import numpy as np
from scipy import stats
data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(f"Mean: {np.mean(data)}")
print(f"Median: {np.median(data)}")
print(f"Mode: {stats.mode(data, keepdims=True).mode[0]}")
print(f"Std Dev: {np.std(data):.4f}")
```

Post Experiment Questions

1) What are the Mean, Median, Variance and Mode of the data22, 7, 22, 1, 7, 18, 18, 16, 6, 6, 7.
2) Katie earned 84,92,84,75 and 70 on her first 5 tests. What is the minimum grade Katie needs to earn on the next test to have a mean of 84?

Experiment No. 3

Objective: To perform data import/export (.CSV, .XLS, .TXT) operations using data frames in Python.

Brief Description

About PANDAS - Pandas is an open-source Python library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Few features of Pandas

- Fast and efficient DataFrame object for data manipulation with integrated indexing
- Tools for reading and writing data between in-memory data structures and different formats: (.CSV, .XLS, .TXT)
- Columns can be inserted and deleted from data structures for size mutability
- Aggregating or transforming data with a powerful group by engine allowing split-apply-combine operations on data sets
- High performance merging and joining of data sets
- Hierarchical axis indexing provides an intuitive way of working with
- high-dimensional data in a lower-dimensional data structure

To install pandas

```
pip install pandas
```

To work with pandas in Python

```
import pandas as pd
```

To create dataframe from list

From 1-d list

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print df
```

From 2-d list

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print df
```

To create a dataframe from dictionary

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'], 'Age':[28,34,29,42]}
df= pd.DataFrame(data)
```

To create a DataFrame by passing a list of dictionaries and the row indices

```
import pandas as pd
data = [{'a': 1, 'b': 2}, {'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data, index=['first', 'second'])
print df
```

Select a single column by passing the column name
df['col_name']
Select multiple columns as a DataFrame by passing a list
df[['col_name1', 'col_name2']]
Adding a row to the dataframe using Series
df ['three']=pd. Series([10,20,30],index=['a', 'b', 'c'])
Select a single row by loc
df.loc['row_name']
Select multiple rows by loc
df.loc[['row_name1', 'row_name2']]
Select range of rows by loc
df.loc['row1':'row4']
Selecting a row and column from the dataFrame from location
df ['column_name']
df.loc[row_selection, column_selection]
Example: df.loc[['Dean', 'Cornelia'], ['age', 'state', 'score']]

Selecting rows and columns using iloc
df.iloc[3]#select single row
df.iloc[[5, 2, 4]]# select specific rows
df.iloc[3:5]# select rows in range
df.iloc[3:6, [1, 4]]# select multiple rows and columns
df.iloc[0, 2]# select single row and column
df.iloc[:, 5] # select column

To read data from csv file into pandas dataframes
data=pd.read_csv("path\\filename.csv")
df=pd.DataFrame(data)
To access first rows of created dataframes
df1.head(5)

Post Experiment Questions

1. Create the following data frame using dictionary and implement the operations on it:

A DATAFRAME STORES DATA IN A ROW-AND-COLUMN STRUCTURE

name	region	sales	expense
William	East	50000	42000
William	East	50000	42000
Emma	North	52000	43000
Emma	West	52000	43000
Anika	East	65000	44000
Anika	East	72000	53000

a. Display data of columns ‘region’ and ‘sales’.
b. Display data of row 2 and column 3 using loc and iloc function both.
c. Add one more row having values ‘ABC’,’South’,80000, 40000 in all fields and display the dataframe
d. Add one column ‘gender’ in the above dataframe and display the updated dataframe.
e. Rename column 1 by ename and display the updated dataframe.
f. Display the information of the dataframe and display the statistics of the given data using describe.
g. Use group by to create new dataframe based on gender and compute group wise sum, average and standard deviation of columns ‘sales’ and ‘expense’.
h. Delete the column ‘region’ from the dataframe and display the upadted dataframe.
i. Delete the last inserted row and display the updated dataframe.
j. Store the created dataframe into ‘emp.csv’ and then read the data from this csv into a new dataframe and display its contents.

Experiment No. 4

Objective: Write a program to perform following data pre-processing operations on various datasets in Python:

- Handling Missing data
- Min-Max normalization
- Lemmatization
- Stop word Removal etc

Brief Description

Input

Any dataset such as IRIS dataset retrieved from UCI repository

Operations to be performed on the retrieved dataset

- To read csv file into an input dataframe 'data' using the command

data = pd.read_csv("C:\\Users\\Lenovo\\Desktop\\iris.txt", header=None)

- To find missing/null values in given dataframe using different methods to handle missing values like replacing by mean,median,mode or constant velues
- Perform Min-Max and Standardization operation on the input data.

Min Max Normalization is used to rescale variables into the range [0, 1]

For example, for a dataset, we have min and max observable values as 30 and -10. We can then normalize any value, like 18.8, as follows:

y = (x - min) / (max - min)

y = (18.8 - (-10)) / (30 - (-10))

y = 28.8 / 40

y = 0.72

Standardization

Standardizing a dataset involves rescaling the distribution of values so that the mean of observed values is 0 and the standard deviation.

Subtracting the mean from the data is called centering, whereas dividing by the standard deviation is called scaling.

A value is standardized as follows:

y = (x – mean) / standard_deviation

where the mean is calculated as:

mean = sum(x) / count(x)

And the standard_deviation is calculated as:

standard_deviation = sqrt(sum((x – mean)^2) / count(x))

Example: We have a mean of 10.0 and a standard deviation of about 5.0. Using these values, we can standardize the first value of 20.7 as follows:

y = (x – mean) / standard_deviation

y = (20.7 – 10) / 5

y = (10.7) / 5

y = 2.14

Output

To print first five rows of the dataframe

To print rows and columns of the dataframe

To rename all columns with different names

To extract a class of data in a separate dataframe say ‘y’ and then drop it from the original data frame. Display the resultant dataframe data and y.

To print information of the input dataframe ‘data’

To print statistical information of the input dataframe ‘data’

Visualize the data using any one visualization tool of the matplotlib library in Python before and after applying data pre-processing operations on the data.

Hint: import matplotlib # Library used to visualize the data in Python

Experiment No. 5

Objective: Write a program to implement support vector machines and principal component analysis.

Brief Description: This example uses the built-in Iris dataset for demonstration. PCA is used to reduce the dimensionality of the data from 4 features to 2, and an SVM is then trained on the reduced data.

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

# 1. Load and Preprocess Data
# Load the Iris dataset from sklearn
iris = datasets.load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names

# Standardize the features (important for both PCA and SVM)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)
print("Original data shape:", X_train.shape)

# 2. Implement Principal Component Analysis (PCA)
# Initialize PCA to reduce dimensions to 2 principal components
pca = PCA(n_components=2)
```

```
# Fit and transform the training data
X_train_pca = pca.fit_transform(X_train)
# Transform the testing data
X_test_pca = pca.transform(X_test)
print("Data shape after PCA:", X_train_pca.shape)
print("Explained variance ratio by the two components:", pca.explained_variance_ratio_)

# 3. Implement Support Vector Machine (SVM)
# Initialize an SVM classifier with a radial basis function (RBF) kernel
svm = SVC(kernel='rbf', random_state=42)

# Train the SVM on the PCA-transformed training data
svm.fit(X_train_pca, y_train)

# Make predictions on the PCA-transformed testing data
y_pred = svm.predict(X_test_pca)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"\nSVM Accuracy on PCA-reduced test data: {accuracy:.2f}")

# 4. (Optional) Visualize the results of PCA and SVM decision boundary
def plot_decision_boundary(X, y, model, title):
    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])


    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

    # Plot also the training points
    for i, target_name in enumerate(target_names):
        plt.scatter(X[y == i, 0], X[y == i, 1],
                    label=target_name,
                    cmap=plt.cm.coolwarm, s=20, edgecolors='k')
```

```
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.title(title)
    plt.legend()
    plt.show()
plot_decision_boundary(X_test_pca, y_test, svm, 'SVM Decision Boundary after PCA')
```

Key Libraries Used:

NumPy for numerical operations.

Matplotlib for optional visualization.

Scikit-learn (sklearn) for the PCA, SVM, data splitting, scaling, and evaluation functions. This is the primary resource for machine learning in Python.

How to Run This Code:

1. Install necessary libraries if you don't have them:

pip install numpy matplotlib scikit-learn

2. Save the code above as a Python file (e.g., ml_demo.py).

3. Run the script from your terminal

python ml_demo.py

Experiment No. 6

Objective: To perform dimensionality reduction using **Principal Component Analysis (PCA)** and compare the performance of the **Support Vector Machine (SVM) classifier** with and without dimensionality reduction on the **Diabetes dataset**.

Input

Diabetes dataset collected from the **UCI Repository**.

Operations to be Performed

1. Extract the **class attribute** of the dataset into a separate dataframe named **'y'**, and then remove it from the original dataframe. Display the resultant dataframe **data** and **y**.
2. Apply **normalization (standard normalization)** on the input data.
3. Apply **PCA** to reduce the data into **n-components** (choose **n = 3**).
4. Visualize the **scatter plots of the n-components** after applying PCA using **Matplotlib**. (There will be **two scatter plots for n = 3**.)
5. Split the **original data** and the **data with n components** into **training and testing datasets**.
6. Build a model on both **original training data** and **reduced training data** using **Support Vector Machine (SVM)**.
7. Evaluate the model on **test data** and plot the **confusion matrix** with and without dimensionality reduction.
8. Experiment with different parameter settings of **SVM kernels**, **C**, and **gamma values** to obtain the **best parameter configuration** for the SVM model.
9. Visualize the **comparative results of accuracy and F-measure** using **Matplotlib**.

Output

- Display the **scatter plot after applying PCA**.
- Print the **confusion matrix of the evaluated model before and after applying PCA**.
- Display a **histogram to compare the model performance before and after applying PCA**.

Post Experiment Questions

1. What is the **need for dimensionality reduction** in classification?
2. Why is **standardization important before running the PCA algorithm**?

Python Code: PCA + SVM on Diabetes Dataset

```
pip install numpy pandas matplotlib scikit-learn seaborn
```

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_diabetes
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score
import seaborn as sns

# ----------------------------------------------------
# Step 1: Load Diabetes Dataset
# ----------------------------------------------------

diabetes = load_diabetes()

X = pd.DataFrame(diabetes.data)
y = pd.Series(diabetes.target)

# Convert regression target into classification
y = (y > y.mean()).astype(int)

print("Original Data Shape:", X.shape)

# ----------------------------------------------------
# Step 2: Standard Normalization
# ----------------------------------------------------

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ----------------------------------------------------
# Step 3: Apply PCA (n = 3 components)
# ----------------------------------------------------

pca = PCA(n_components=3)
X_pca = pca.fit_transform(X_scaled)

print("Reduced Data Shape:", X_pca.shape)

# ----------------------------------------------------
# Step 4: Scatter Plot after PCA
# ----------------------------------------------------

plt.figure()
```

```
plt.scatter(X_pca[:,0], X_pca[:,1], c=y)
plt.title("Scatter Plot of PCA Components (PC1 vs PC2)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")

plt.show()

# ------------------------------------------------
# Step 5: Train-Test Split
# ------------------------------------------------

# Original Data
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42)

# PCA Data
Xp_train, Xp_test, yp_train, yp_test = train_test_split(
    X_pca, y, test_size=0.3, random_state=42)

# ------------------------------------------------
# Step 6: Train SVM Model (Original Data)
# ------------------------------------------------

svm_original = SVC(kernel='rbf', C=1, gamma='scale')

svm_original.fit(X_train, y_train)

pred_original = svm_original.predict(X_test)

# ------------------------------------------------
# Step 7: Train SVM Model (PCA Data)
# ------------------------------------------------

svm_pca = SVC(kernel='rbf', C=1, gamma='scale')

svm_pca.fit(Xp_train, yp_train)

pred_pca = svm_pca.predict(Xp_test)

# ------------------------------------------------
# Step 8: Evaluation
# ------------------------------------------------

acc_original = accuracy_score(y_test, pred_original)
f1_original = f1_score(y_test, pred_original)

acc_pca = accuracy_score(yp_test, pred_pca)
f1_pca = f1_score(yp_test, pred_pca)

print("\nAccuracy without PCA:", acc_original)
```

```
print("F1 Score without PCA:", f1_original)

print("\nAccuracy with PCA:", acc_pca)
print("F1 Score with PCA:", f1_pca)

# ----------------------------------------------------
# Step 9: Confusion Matrix
# ----------------------------------------------------

cm_original = confusion_matrix(y_test, pred_original)
cm_pca = confusion_matrix(yp_test, pred_pca)

plt.figure()
sns.heatmap(cm_original, annot=True, fmt='d')
plt.title("Confusion Matrix (Without PCA)")
plt.show()

plt.figure()
sns.heatmap(cm_pca, annot=True, fmt='d')
plt.title("Confusion Matrix (With PCA)")
plt.show()

# ----------------------------------------------------
# Step 10: Histogram Comparison
# ----------------------------------------------------

labels = ['Accuracy', 'F1 Score']

without_pca = [acc_original, f1_original]
with_pca = [acc_pca, f1_pca]

x = np.arange(len(labels))

plt.figure()

plt.bar(x-0.2, without_pca, width=0.4, label='Without PCA')
plt.bar(x+0.2, with_pca, width=0.4, label='With PCA')

plt.xticks(x, labels)
plt.ylabel("Score")
plt.title("Comparison of Model Performance")
plt.legend()

plt.show()
```

Experiment No. 7

Objective: To perform K-Means clustering on the Country-data dataset and evaluate the clustering results using data visualization and the elbow method.

Theory

Clustering is an **unsupervised machine learning technique** used to group similar data points into clusters based on their characteristics. Unlike supervised learning, clustering does not use labeled data.

K-Means Clustering

K-Means is one of the most widely used clustering algorithms. It partitions the dataset into **K clusters**, where each data point belongs to the cluster with the nearest mean (centroid).

Working of K-Means Algorithm

1. Choose the number of clusters **K**.
2. Initialize **K centroids randomly**.
3. Assign each data point to the nearest centroid.
4. Recalculate centroids by taking the mean of all data points assigned to that cluster.
5. Repeat the process until cluster assignments do not change.
6.

Elbow Method

The **Elbow Method** helps determine the optimal number of clusters. It plots the **Within Cluster Sum of Squares (WCSS)** against the number of clusters. The point where the curve bends like an elbow indicates the optimal value of **K**.

Data Normalization

Before clustering, features are scaled using **Min-Max normalization** to ensure that all variables contribute equally to the clustering process.

Algorithm

1. Import required libraries such as **pandas, numpy, matplotlib, seaborn, sklearn**.
2. Load the **Country-data dataset**.
3. Display dataset information and statistical summary.
4. Remove the non-numeric column (country name).
5. Apply **Min-Max normalization** on the dataset.
6. Generate a **correlation heatmap** for the dataset.
7. Apply the **Elbow Method** to determine the optimal value of **K**.
8. Apply **K-Means clustering algorithm** using sklearn.
9. Predict cluster labels for the data points.
10. Visualize clustering results using scatter plots.

Python Code

```
# Import required libraries

import pandas as pd
```

```
import matplotlib.pyplot as plt

import seaborn as sns


from sklearn.preprocessing import MinMaxScaler

from sklearn.cluster import KMeans


# Load dataset

data = pd.read_csv("Country-data.csv")


# Display dataset information

print(data.info())


# Statistical summary

print(data.describe())


# Display first five rows

print(data.head())


# Remove country column

countries = data['country']

X = data.drop('country', axis=1)


# Min-Max Normalization

scaler = MinMaxScaler()
```

```
X_scaled = scaler.fit_transform(X)

# Convert back to DataFrame
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

# Heatmap
plt.figure()
sns.heatmap(X_scaled.corr(), annot=True)
plt.title("Correlation Heatmap")
plt.show()

# Elbow Method
wcss = []

for i in range(1,11):
    km = KMeans(n_clusters=i, random_state=42)
    km.fit(X_scaled)
    wcss.append(km.inertia_)

plt.figure()
plt.plot(range(1,11), wcss, marker='o')
plt.title("Elbow Method")
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
```

```
plt.show()


# Apply K-Means

k = 3

km = KMeans(n_clusters=k, random_state=42)


clusters = km.fit_predict(X_scaled)


# Add cluster label

data['Cluster'] = clusters


print(data[['country','Cluster']].head())


# Cluster Visualization

plt.figure()

plt.scatter(X_scaled.iloc[:,0], X_scaled.iloc[:,1], c=clusters)

plt.title("K-Means Clustering Result")

plt.xlabel(X_scaled.columns[0])

plt.ylabel(X_scaled.columns[1])

plt.show()
```

Post Experiment Questions

1. Name the most appropriate strategy for data cleaning before performing clustering analysis when the dataset has very few data points.

Answer:
Removing missing values carefully, applying normalization, and avoiding aggressive outlier removal are recommended.

2. What is the minimum number of variables/features required to perform clustering?

Answer:
At least **two features** are required to effectively perform clustering.

3. What is the termination condition for determining K-Means clustering?

Answer:
The algorithm terminates when:

- Cluster assignments no longer change, or
- Centroids stop moving significantly, or
- Maximum iterations are reached.

4. In which cases will K-Means clustering fail to give good results?

Answer:
K-Means may fail when:

- Clusters have **non-spherical shapes**
- Data contains **many outliers**
- Clusters have **unequal sizes or densities**
- The value of **K is chosen incorrectly**