RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM _ 602 105

AI19P71 - DATA VISUALIZATION USING PYTHON

Laboratory Record NoteBook

Name : . . . . . MEDHINI K . . . . . .

Year / Branch / Section : IV- AIML

Register No. : . . . .2116221501075 . . . . . .

College Roll No. : . . . . . .221501075 . . . . . .

Semester : 07

Academic Year : 2025-2026

RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

BONAFIDE CERTIFICATE

Name : . . . . . MEDHINI K . . . . . .

Academic Year : . . . . 2025-2026. . . . . Semester : 07 Branch : AIML

Register No. **2116221501075**

Certified that this is the bonafide record of work done by the above student in the

AI19P71 - DATA VISUALIZATION USING PYTHON LABORATORY

during the year 2025 - 2026.

Signature of Faculty in-charge

Submitted for the Practical Examination held on . . . . . .

Internal Examiner **External Examiner**

INDEX

Ex. No	Date	Name of the Experiment	Sign
1 (a) (b) (c)		Basic numpy array operations Aggregate functions Splitting and aggregation	
2 (a) (b)		Analyzing academic performance Analyzing healthcare service efficiency	
3 (a) (b)		Comprehensive student performance analysis using pandas Movie ratings aggregation and review analysis	
4		Visualizing employee productivity (line & bar chart)	
5		Visualizing employee productivity (box & scatter plots)	
6		Sales performance visualization using store names	
7		Visualizing text data using word cloud	
8		Implementation of linear regression	
9		Implementation of logistic regression	
10		Implementation of svm classification techniques	
11		Implementation of decision tree classification techniques	
12		Implementation of hierarchical clustering	
13		Implementation of clustering techniques k means	

Ex.No. 1 (a)	BASIC NUMPY ARRAY OPERATIONS
Date :	

AIM:

To understand the basics of NumPy arrays including creation, attributes, slicing and indexing.

ALGORITHM:

Step 1: Start.

Step 2: Initialize months and sales_revenue lists.

Step 3: Convert sales_revenue to NumPy array → revenue_array.

Step 4: Retrieve array attributes: shape, size, dtype.

Step 5: Slice revenue_array for January–June → first_half_revenue.

Step 6: Index May's revenue → revenue_array[4].

Step 7: Calculate total revenue → revenue_array.sum().

Step 8: Find index of max revenue → revenue_array.argmax(); get corresponding month.

Step 9: Display results (attributes, first half, May, total, highest month).

Step 10: Stop.

SOURCE CODE:

```
import numpy as np

months = ['January', 'February', 'March', 'April', 'May', 'June',' July', 'August', 'September', 'October', 'November', 'December']

sales_revenue = [12000, 15000, 17000, 13000, 16000, 18000, 20000, 19000, 17500, 16000, 15000, 16500]
sales_array = np.array(sales_revenue)

print("Shape:", sales_array.shape)

print("Size:", sales_array.size)

print("Data type:", sales_array.dtype)

print("First half of the year sales revenue:", sales_array[:6])

print("Sales revenue for May:", sales_array[4])
```

```
total_sales = np.sum(sales_array)
print("Total sales revenue for the year:", total_sales)
max_index = np.argmax(sales_array)  # gives index of max value
print("Month with highest sales revenue:", months[max_index])
```

OUTPUT:

```
Shape: (12,)
Size: 12
Data type: int64
First half of the year sales revenue: [12000 15000 17000 13000 16000 18000]
Sales revenue for May: 16000
Total sales revenue for the year: 195000
Month with highest sales revenue: July
```

RESULT:

Thus, the Python program to analyze the company's monthly sales revenue using NumPy by retrieving attributes, slicing data, indexing values, computing total revenue and identifying the highest sales month has been executed successfully.

Ex.No. 1 (b)	AGGREGATE FUNCTIONS
Date:	

AIM:

To use aggregate functions in NumPy for summarizing and analyzing data.

ALGORITHM:

Step 1: Start.

Step 2: Initialize the dictionary returns_data with returns for Portfolio A, B, and C.

Step 3: Convert each portfolio's return list into a NumPy array — A_array, B_array and C_array.

Step 4: Calculate the total return for each portfolio using the sum() function.

Step 5: Compute the mean monthly return for each portfolio using the mean() function.

Step 6: Identify the portfolio with the highest mean return using argmax().

Step 7: Combine all portfolio arrays and compute the overall average return using mean() on the combined data.

Step 8: Calculate the variance and standard deviation for each portfolio using var() and std() respectively to assess investment risk.

Step 9: Display all computed statistics.

Step 10: Stop.

SOURCE CODE:

```
import numpy as np
returns_data = {    'portfolio_A': [0.05, 0.02, 0.04, 0.03, 0.06, 0.01],
   'portfolio_B': [0.04, 0.03, 0.05, 0.02, 0.01, 0.04],
   'portfolio_C': [0.03, 0.01, 0.02, 0.03, 0.04, 0.05]}
portfolio_A = np.array(returns_data['portfolio_A'])
portfolio_B = np.array(returns_data['portfolio_B'])
portfolio_C = np.array(returns_data['portfolio_C'])
total_A = np.sum(portfolio_A)
total_B = np.sum(portfolio_B)
total_C = np.sum(portfolio_C)
```

```
mean_A = np.mean(portfolio_A)
mean_B = np.mean(portfolio_B)
mean_C = np.mean(portfolio_C)
mean_returns = {'portfolio_A': mean_A, 'portfolio_B': mean_B, 'portfolio_C': mean_C}
best_portfolio = max(mean_returns, key=mean_returns.get)
all_returns = np.concatenate([portfolio_A, portfolio_B, portfolio_C])
overall_avg = np.mean(all_returns)
var_A, std_A = np.var(portfolio_A), np.std(portfolio_A)
var_B, std_B = np.var(portfolio_B), np.std(portfolio_B)
var_C, std_C = np.var(portfolio_C), np.std(portfolio_C)
print("Total returns:", total_A, total_B, total_C)
print("Mean monthly returns:", mean_A, mean_B, mean_C)
print("Portfolio with highest mean return:", best_portfolio)
print("Overall average return across all portfolios:", overall_avg)
print("Variance:", var_A, var_B, var_C)
print("Standard deviation:", std_A, std_B, std_C)
```

OUTPUT:

```
Total returns: 0.21000000000000002 0.19000000000000003 0.18
Mean monthly returns: 0.035 0.03166666666666667 0.03
Portfolio with highest mean return: portfolio_A
Overall average return across all portfolios: 0.03222222222222223
Variance: 0.00029166666666666664 0.00018055555555555557 0.00016666666666666666
Standard deviation: 0.01707825127659933 0.01343709624716425 0.012909944487358056
```

RESULT:

Thus the Python program to analyze and summarize the investment portfolios' monthly returns using NumPy aggregate functions by computing total returns, mean returns, overall averages, variances and standard deviations has been executed successfully.

Ex.No. 1 (c)	SPLITTING AND AGGREGATION
Date:	

AIM:

To practice slicing, indexing, splitting, and aggregating Numpy arrays.

ALGORITHM:

Step 1: Start.

Step 2: Initialize the rainfall dataset:

rainfall_data = [120, 85, 90, 110, 95, 100, 105, 85, 115, 130, 125, 140]

Step 3: Convert the list into a NumPy array → rain_array.

Step 4: Slice rain_array[0:3] to obtain rainfall for the first quarter (Jan–Mar).

Step 5: Index rain_array[6] to get the rainfall for July.

Step 6: Split the array into four equal quarters using np.split(rain_array, 4).

Step 7: For each quarter:

(a) Print the rainfall data.

(b) Calculate total rainfall using sum().

(c) Identify the month with highest rainfall using max().

Step 8: Display all computed quarterly totals and highest rainfall values.

Step 9: Stop.

SOURCE CODE:

```
import numpy as np
rainfall_data = [120, 85, 90, 110, 95, 100, 105, 85, 115, 130, 125, 140]
rainfall_array = np.array(rainfall_data)
first_quarter = rainfall_array[:3]
print("First Quarter Rainfall Data:", first_quarter)
print("Rainfall for July:", rainfall_array[6])
quarters = np.split(rainfall_array, 4)
print("First Quarter Rainfall Data:", quarters[0])
```

```
print("Second Quarter Rainfall Data:", quarters[1])
print("Third Quarter Rainfall Data:", quarters[2])
print("Fourth Quarter Rainfall Data:", quarters[3])
for i, q in enumerate(quarters, start=1):
    total = np.sum(q)
    max_index = np.argmax(q) + 1  # +1 for month number in that quarter
    max_value = q[max_index - 1]
    print(f"Total Rainfall for Quarter {i}: {total}")
    print(f"Month with highest rainfall in Quarter {i}: Month {max_index} with {max_value}")
```

OUTPUT:

```
First Quarter Rainfall Data: [120  85  90]
Rainfall for July: 105
First Quarter Rainfall Data: [120  85  90]
Second Quarter Rainfall Data: [110  95 100]
Third Quarter Rainfall Data: [105  85 115]
Fourth Quarter Rainfall Data: [130 125 140]
Total Rainfall for Quarter 1: 295
Month with highest rainfall in Quarter 1: Month 1 with 120
Total Rainfall for Quarter 2: 305
Month with highest rainfall in Quarter 2: Month 1 with 110
Total Rainfall for Quarter 3: 305
Month with highest rainfall in Quarter 3: Month 3 with 115
Total Rainfall for Quarter 4: 395
Month with highest rainfall in Quarter 4: Month 3 with 140
```

RESULT:

Thus, the Python program to analyze monthly rainfall data using NumPy by performing slicing, indexing, splitting and aggregation to determine quarterly totals and the month with the highest rainfall has been executed successfully.

Ex.No. 2 (a)	ANALYZING ACADEMIC PERFORMANCE
Date :	

AIM:

To assess student performance trends across subjects while managing missing values, duplicates, and normalization using NumPy operations.

ALGORITHM:

Step 1: Start.

Step 2: Initialize the dataset with roll numbers and subject scores.

Step 3: Convert the list into a NumPy array.

Step 4: Identify missing values and duplicate records.

Step 5: Replace missing values with the median of each subject.

Step 6: Remove duplicate records, keeping the first occurrence.

Step 7: Apply Min–Max normalization to subject scores (excluding roll numbers).

Step 8: Calculate mean, median, and standard deviation for each subject.

Step 9: Identify the subject with the highest variability.

Step 10: Stop.

SOURCE CODE:

```
import numpy as np
import pandas as pd

student_data = [(101, 45, 78, None),(102, 65, 56, 77),
                (103, 95, 85, 92), (102, 65, 56, 77),
                (104, 45, None, 88), (101, 45, 78, None)]
student_df = pd.DataFrame(student_data, columns=['Roll_Number', 'Mathematics', 'Science',
'English'])
missing_values = student_df.isnull().sum()
duplicate_entries = student_df.duplicated(subset=['Roll_Number'], keep=False)
print("Missing values per subject:\n", missing_values)
```

```
print("\nDuplicate entries based on Roll Number:\n", student_df[duplicate_entries])

for col in ['Mathematics', 'Science', 'English']:
    student_df[col].fillna(student_df[col].median(),  inplace=True)
student_df_cleaned = student_df.drop_duplicates(subset=['Roll_Number'], keep='first')
print("\nCleaned DataFrame:\n", student_df_cleaned)
scores_df = student_df_cleaned[['Mathematics', 'Science', 'English']].astype(float)
student_df_normalized = (scores_df - scores_df.min()) / (scores_df.max() - scores_df.min())
normalized_mean = student_df_normalized.mean()
normalized_median = student_df_normalized.median()
normalized_std = student_df_normalized.std()
print("\nNormalized Mean:\n", normalized_mean)
print("\nNormalized Median:\n", normalized_median)
print("\nNormalized Standard Deviation:\n", normalized_std)
subject_highest_variability = normalized_std.idxmax()
print(f"\nSubject with the highest variability: {subject_highest_variability}")
```

OUTPUT:

```
Missing values per subject:
 Roll_Number    0
Mathematics     0
Science         1
English         2
dtype: int64

Duplicate entries based on Roll Number:
    Roll_Number  Mathematics  Science  English
0          101           45     78.0      NaN
1          102           65     56.0     77.0
3          102           65     56.0     77.0
5          101           45     78.0      NaN

Cleaned DataFrame:
    Roll_Number  Mathematics  Science  English
0          101           45     78.0     82.5
1          102           65     56.0     77.0
2          103           95     85.0     92.0
4          104           45     78.0     88.0

Normalized Mean:
 Mathematics    0.35000
Science         0.62931
English         0.52500
dtype: float64

Normalized Median:
 Mathematics    0.200000
Science         0.758621
English         0.550000
dtype: float64

Normalized Standard Deviation:
 Mathematics    0.472582
Science         0.434697
English         0.435784
dtype: float64

Subject with the highest variability: Mathematics
```

RESULT:

Thus, the Python program to clean, normalize, and analyze student academic performance data using NumPy has been executed successfully.

Ex.No. 2 (b)	ANALYZING HEALTHCARE SERVICE EFFICIENCY
Date :	

AIM:

To evaluate departmental performance by handling missing data, duplicates, and performing normalization and statistical analysis using NumPy.

ALGORITHM:

Step 1: Start.

Step 2: Initialize the dataset with department IDs and performance metrics.

Step 3: Convert the list into a NumPy array.

Step 4: Identify missing values and duplicate department records.

Step 5: Replace missing values with the median of each metric.

Step 6: Remove duplicate records, keeping the first occurrence of each department.

Step 7: Apply Min–Max normalization to performance scores (excluding IDs).

Step 8: Calculate mean, median, and standard deviation for each metric.

Step 9: Identify the metric with the highest variability in scores.

Step 10: Stop.

SOURCE CODE:

```
import numpy as np

data = [
    (701, 90, 95, None),
    (702, 88, 90, 85),
    (703, 92, None, 80),
    (702, 88, 90, 85),
    (704, 85, None, 78),
    (701, 90, 95, None)
]
```

```
arr = np.array(data, dtype=object)
print("Original Data:\n", arr)
missing_counts = np.sum(arr[:, 1:] == None, axis=0)
print("\nMissing values per metric (Satisfaction, Success Rate, Wait Time):", missing_counts)
_, unique_idx = np.unique(arr[:, 0], return_index=True)
duplicates = np.setdiff1d(np.arange(len(arr)), unique_idx)
print("\nDuplicate records:\n", arr[duplicates])
for col in range(1, arr.shape[1]):
    values = arr[:, col]
    median_val = np.median([v for v in values if v is not None])
    arr[:, col] = [median_val if v is None else v for v in values]
arr = arr[np.sort(unique_idx)]
print("\nCleaned Data (missing handled, duplicates removed):\n", arr)
scores = arr[:, 1:].astype(float)
min_vals = scores.min(axis=0)
max_vals = scores.max(axis=0)
normalized_scores = (scores - min_vals) / (max_vals - min_vals)
print("\nNormalized Scores:\n", normalized_scores)
means = normalized_scores.mean(axis=0)
medians = np.median(normalized_scores, axis=0)
std_devs = normalized_scores.std(axis=0)
metrics = ["Patient Satisfaction", "Treatment Success Rate", "Wait Time"]
max_var_metric = metrics[np.argmax(std_devs)]
print("\nStatistical Summary (Normalized Metrics)")
print("-------------------------------------------------")
print("Metric                        | Mean   | Median | Std Dev")
print("-------------------------------------------------")
for i, metric in enumerate(metrics):
    print(f"{metric:<30} | {means[i]:.3f} | {medians[i]:.3f} | {std_devs[i]:.3f}")
```

```
print("--------------------------------------------------")
print(f"Metric with highest variability: {max_var_metric}")
```

OUTPUT:

```
Original Data:
 [[701 90 95 None]
 [702 88 90 85]
 [703 92 None 80]
 [702 88 90 85]
 [704 85 None 78]
 [701 90 95 None]]

Missing values per metric (Satisfaction, Success Rate, Wait Time): [0 2 2]

Duplicate records:
 [[702 88 90 85]
 [701 90 95 None]]

Cleaned Data (missing handled, duplicates removed):
 [[701 90 95 np.float64(82.5)]
 [702 88 90 85]
 [703 92 np.float64(92.5) 80]
 [704 85 np.float64(92.5) 78]]

Normalized Scores:
 [[0.71428571 1.         0.64285714]
 [0.42857143 0.         1.        ]
 [1.         0.5        0.28571429]
 [0.         0.5        0.        ]]

Statistical Summary (Normalized Metrics)
--------------------------------------------------
Metric                          | Mean   | Median | Std Dev
--------------------------------------------------
Patient Satisfaction           | 0.536 | 0.571 | 0.369
Treatment Success Rate         | 0.500 | 0.500 | 0.354
Wait Time                      | 0.482 | 0.464 | 0.376
--------------------------------------------------
Metric with highest variability: Wait Time
```

RESULT:

Thus, the Python program to analyze healthcare service efficiency using NumPy by cleaning data, normalizing metrics and performing statistical analysis to evaluate departmental performance has been executed successfully.

Ex.No. 3 (a)	COMPREHENSIVE STUDENT PERFORMANCE ANALYSIS USING PANDAS
Date:	

AIM:

To analyze student performance data using Pandas by handling missing marks, duplicate entries, inconsistent formatting, and by performing data transformation and statistical analysis.

ALGORITHM:

Step 1: Start.

Step 2: Load CSV into a DataFrame: df = pd.read_csv('student_scores.csv').

Step 3: Inspect data: df.info(), df.describe(), and df.isna().sum() to count missing values.

Step 4: Detect duplicates by name: df[df.duplicated(subset=['Name'], keep=False)].

Step 5: Remove duplicates, keeping first occurrence: df = df.drop_duplicates(subset=['Name'], keep='first').

Step 6: Replace missing marks with subject medians:

Step 7: Assign unique StudentID starting at 1: df.insert(0, 'StudentID', range(1, len(df)+1)).

Step 8: Add TotalMarks and AverageMarks:

Step 9: Min–Max normalize subject scores into new columns

Step 10: Compute normalized statistics and performance metric

SOURCE CODE:

```
import pandas as pd

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

print("Original Data:\n", df)

print("\nDataset Info:")

print(df.info())

print("\nDataset Description:\n", df.describe())

print("\nMissing Values per Subject:\n", df.isna().sum())

duplicates = df[df.duplicated(subset="Name")]

print("\nDuplicate Rows:\n", duplicates)
```

```
df = df.drop_duplicates(subset="Name", keep="first").copy()
for subject in ["Math", "Physics", "Chemistry"]:
    median_val = df[subject].median()
    df[subject] = df[subject].fillna(median_val)
df.insert(0, "StudentID", range(1, len(df) + 1))

print("\nCleaned Data:\n", df)
df["TotalMarks"] = df[["Math", "Physics", "Chemistry"]].sum(axis=1)
df["AverageMarks"] = df["TotalMarks"] / 3
for subject in ["Math", "Physics", "Chemistry"]:
    min_val = df[subject].min()
    max_val = df[subject].max()
    df[f"{subject}_Norm"] = (df[subject] - min_val) / (max_val - min_val)
print("\nData with Normalized Scores:\n", df)
stats = df[["Math_Norm", "Physics_Norm", "Chemistry_Norm"]].agg(["mean", "median",
"std"])
print("\nNormalized Score Statistics:\n", stats)
most_variable_subject = stats.loc["std"].idxmax()
print("\nSubject with highest variability:", most_variable_subject)
top_students = df.nlargest(3, "AverageMarks")[["StudentID", "Name", "AverageMarks"]]
print("\nTop 3 Students:\n", top_students)
pivot_avg = df[["Math", "Physics", "Chemistry"]].mean().to_frame(name="Average Score") #
Excluded 'Name' column
print("\nSubject-wise Average Performance:\n", pivot_avg)
```

OUTPUT:

```
Original Data:
      Name  Math  Physics  Chemistry
0    Alice  78.0     85.0        NaN
1      Bob  82.0      NaN       88.0
2  Charlie  75.0     79.0       91.0
3    Alice  78.0     85.0        NaN
4    David   NaN     92.0       87.0
5      Eve  85.0     84.0       88.0
6      Bob  82.0      NaN       88.0

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Name       7 non-null      object
 1   Math       6 non-null      float64
 2   Physics    5 non-null      float64
 3   Chemistry  5 non-null      float64
dtypes: float64(3), object(1)
memory usage: 356.0+ bytes
None

Dataset Description:
            Math    Physics  Chemistry
count   6.00000   5.000000   5.000000
mean   80.00000  85.000000  88.400000
std     3.63318   4.636809   1.516575
min    75.00000  79.000000  87.000000
25%    78.00000  84.000000  88.000000
50%    80.00000  85.000000  88.000000
75%    82.00000  85.000000  88.000000
max    85.00000  92.000000  91.000000

Missing Values per Subject:
 Name         0
Math         1
Physics      2
Chemistry    2
dtype: int64

Duplicate Rows:
     Name  Math  Physics  Chemistry
3  Alice  78.0     85.0        NaN
6    Bob  82.0      NaN       88.0
```

```
Cleaned Data:
   StudentID     Name  Math  Physics  Chemistry
0          1    Alice  78.0     85.0       88.0
1          2      Bob  82.0     84.5       88.0
2          3  Charlie  75.0     79.0       91.0
4          4    David  88.0     92.0       87.0
5          5      Eve  85.0     84.0       88.0

Data with Normalized Scores:
   StudentID     Name  Math  Physics  Chemistry  TotalMarks  AverageMarks  \
0          1    Alice  78.0     85.0       88.0       251.0     83.666667
1          2      Bob  82.0     84.5       88.0       254.5     84.833333
2          3  Charlie  75.0     79.0       91.0       245.0     81.666667
4          4    David  88.0     92.0       87.0       259.0     86.333333
5          5      Eve  85.0     84.0       88.0       257.0     85.666667

   Math_Norm  Physics_Norm  Cheistry_Norm
0        0.3      0.461538           0.25
1        0.7      0.423077           0.25
2        0.0      0.000000           1.00
4        0.5      1.000000           0.00
5        1.0      0.384615           0.25

Normalized Score Statistics:
        Math_Norm  Physics_Norm  Chemistry_Norm
mean     0.500000      0.453846        0.350000
median   0.500000      0.423077        0.250000
std      0.308709      0.357092        0.379144

Subject with highest variability: Math_Norm

Top 3 Students:
   StudentID   Name  AverageMarks
4          4  David     86.333333
5          5    Eve     85.666667
1          2    Bob     84.833333

Subject-wise Average Performance:
           Average Score
Math                88.0
Physics             84.9
Chemistry           88.4
```

RESULT:

Thus, the Python program to clean, normalize and analyze student assessment scores using Pandas addressing missing marks, duplicates, formatting and producing key statistical and performance insights has been executed successfully.

Ex.No. 3 (b)	MOVIE RATINGS AGGREGATION AND REVIEW ANALYSIS
Date :	

AIM:

To clean, aggregate and analyze movie ratings and reviews using Pandas addressing data duplication, missing values and text inconsistencies.

ALGORITHM:

Step 1: Start.

Step 2: Load the dataset movie_reviews.csv into a Pandas DataFrame.

Step 3: Remove duplicate entries based on UserID and MovieTitle using drop_duplicates().

Step 4: Replace missing ratings with the average rating per movie using groupby().transform('mean').

Step 5: Clean review text by stripping whitespace and converting to lowercase using vectorized string methods (.str.strip(), .str.lower()).

Step 6: Group data by MovieTitle and compute mean, median, and count of ratings.

Step 7: Identify the top-rated movie using the highest mean rating.

Step 8: Count occurrences of the word “amazing” in all reviews using .str.contains('amazing').

Step 9: Filter reviews where rating > 4.0 and review text contains “plot”.

Step 10: Display the cleaned dataset, aggregated statistics, and filtered insights.

Step 11: Stop

SOURCE CODE:

```
import pandas as pd

df = pd.read_csv("/content/movie_reviews.csv")
df = df.drop_duplicates(subset=['UserID', 'MovieTitle'], keep='first')
df['Rating'] = df.groupby('MovieTitle')['Rating'].transform(lambda x: x.fillna(x.mean()))
df['ReviewText'] = df['ReviewText'].str.strip().str.lower()
agg_stats = df.groupby('MovieTitle')['Rating'].agg(['mean', 'median', 'count']).reset_index()
```

```
print("Movie Ratings Aggregation:\n", agg_stats)
top_movie = agg_stats.loc[agg_stats['mean'].idxmax(), 'MovieTitle']
print(f"\nTop-rated movie: {top_movie}")
# Count occurrences of the word "amazing" across all reviews
amazing_count = df['ReviewText'].str.count(r'\bamazing\b').sum()
print(f"\nTotal 'amazing' word count in reviews: {amazing_count}")
filtered_reviews = df[(df['Rating'] > 4.0) & (df['ReviewText'].str.contains(r'\bplot\b'))]
print("\nFiltered reviews (rating > 4.0 and contains 'plot'):\n", filtered_reviews)
```

OUTPUT:

```
Movie Ratings Aggregation:
      MovieTitle      mean  median  count
0     Inception  4.233333     4.2      3
1  Interstellar  5.000000     5.0      2

Top-rated movie: Interstellar

Total 'amazing' word count in reviews: 1

Filtered reviews (rating > 4.0 and contains 'plot'):
   UserID MovieTitle  Rating              ReviewText
5   U005  Inception     4.2  good plot but confusing
```

RESULT:

Thus, the Python program to analyze movie ratings and reviews using Pandas by removing duplicates, handling missing ratings, cleaning text data, performing aggregation and extracting keyword-based insights has been executed successfully.

Ex.No. 4	VISUALIZING EMPLOYEE PRODUCTIVITY(LINE & BAR CHART)
Date :	

AIM:

To visualize employee productivity trends across quarters and compare average productivity between departments using line and bar charts in Pandas and Matplotlib.

ALGORITHM:

Step 1: Start.

Step 2: Initialize or load the dataset containing employee productivity scores for each quarter.

Step 3: Convert the data into a Pandas DataFrame.

Step 4: Identify and remove duplicate employee records using .drop_duplicates().

Step 5: Handle missing values by replacing them with the mean or median of the respective quarter using .fillna().

Step 6: Plot a Line Chart to visualize productivity trends across quarters for all employees or departments.

Step 7: Group data by department and calculate the average productivity per department across all quarters.

Step 8: Plot a Bar Chart to compare average productivity among departments.

Step 9: Display and interpret the visualizations.

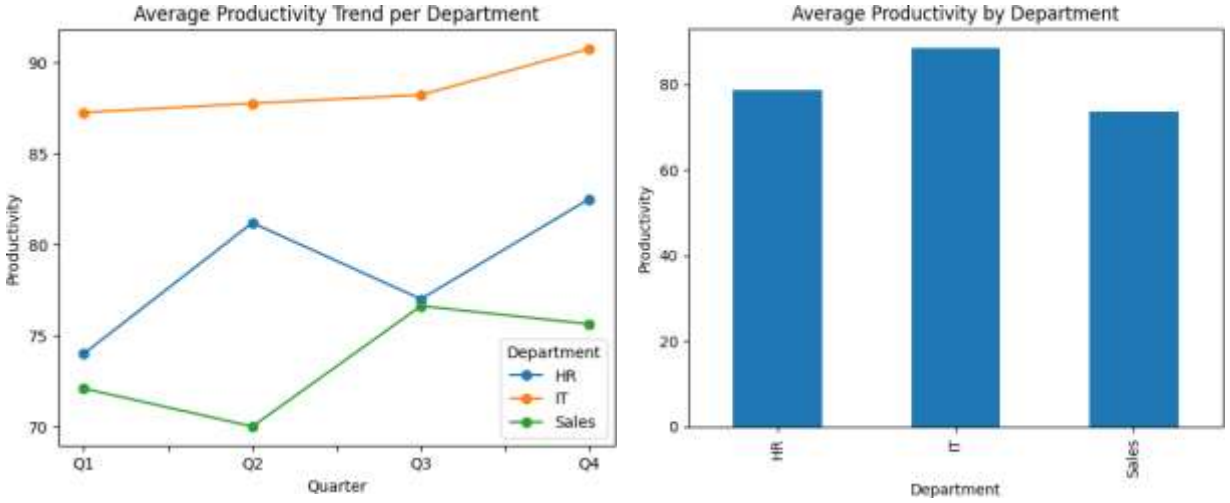
Step 10: Stop.

SOURCE CODE:

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

data = {
   'Emp_ID':[201,202,203,204,205,206,202,207,208,209],
   'Department':['IT','HR','IT','Sales','HR','IT','HR','Sales','Sales','IT'],
   'Q1':[80,70,90,None,78,88,70,65,72,91],
   'Q2':[85,None,95,65,82,84,None,70,75,87],
```

```
    'Q3':[None,75,92,70,79,90,75,None,78,89],
    'Q4':[90,80,94,75,85,86,80,68,None,93]}
df = pd.DataFrame(data)
df = df.drop_duplicates()
df = df.fillna(df.mean(numeric_only=True))
#Line Chart
df.groupby('Department')[['Q1','Q2','Q3','Q4']].mean().T.plot(marker='o')
plt.title('Average Productivity Trend per Department')
plt.xlabel('Quarter')
plt.ylabel('Productivity')
plt.show()
#Bar Chart
df.groupby('Department')[['Q1','Q2','Q3','Q4']].mean().mean(axis=1).plot(kind='bar')
plt.title('Average Productivity by Department')
plt.ylabel('Productivity')
plt.show()
```

OUTPUT:



RESULT:

Thus, the Python program to visualize employee productivity trends and departmental averages using line and bar charts after cleaning and imputing missing data has been executed successfully.

Ex.No. 5	VISUALIZING EMPLOYEE PRODUCTIVITY (BOX & SCATTER PLOTS)
Date :	

AIM:

To analyze the distribution and relationships of employee productivity scores across quarters using box and scatter plots in Pandas and Matplotlib.

ALGORITHM:

Step 1: Start.

Step 2: Load the cleaned employee productivity dataset from the previous exercise.

Step 3: Verify the data integrity (no duplicates or missing values).

Step 4: Plot a Box Plot to analyze the distribution and variability of productivity scores across the four quarters (Q1–Q4).

Step 5: Plot a Scatter Plot to visualize the relationship between productivity in two quarters.

Step 6: Label all axes, add titles, and apply color differentiation for clarity.

Step 7: Interpret visual patterns to assess consistency and correlations in performance.

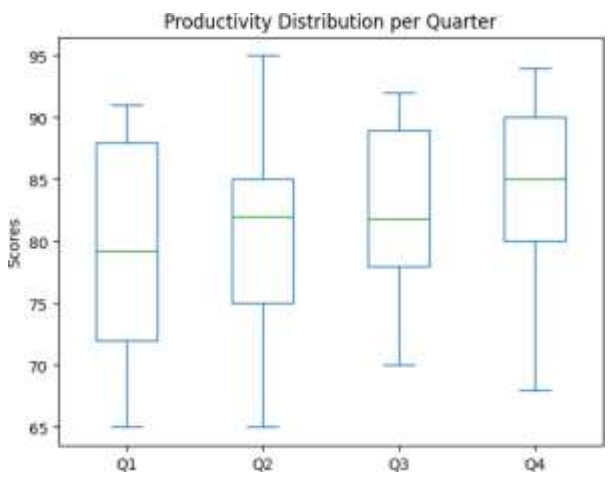
Step 8: Stop.

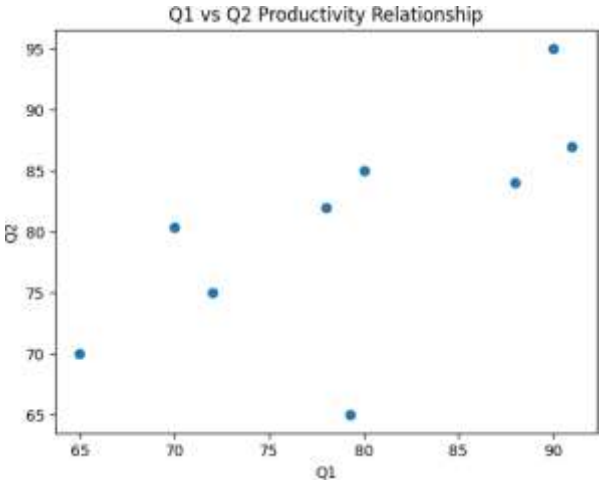
SOURCE CODE:

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

data = {
   'Emp_ID':[201,202,203,204,205,206,202,207,208,209],
   'Department':['IT','HR','IT','Sales','HR','IT','HR','Sales','Sales','IT'],
   'Q1':[80,70,90,None,78,88,70,65,72,91],
   'Q2':[85,None,95,65,82,84,None,70,75,87],
   'Q3':[None,75,92,70,79,90,75,None,78,89],
   'Q4':[90,80,94,75,85,86,80,68,None,93]
}
df = pd.DataFrame(data)
df = df.drop_duplicates()
df = df.fillna(df.mean(numeric_only=True))
```

```
#Box plot
df[['Q1','Q2','Q3','Q4']].plot(kind='box')
plt.title('Productivity Distribution per Quarter')
plt.ylabel('Scores')
plt.show()
# Scatter Plot
plt.scatter(df['Q1'], df['Q2'])
plt.title('Q1 vs Q2 Productivity Relationship')
plt.xlabel('Q1')
plt.ylabel('Q2')
plt.show()
```

OUTPUT:





RESULT:

Thus, the Python program to analyze employee productivity distribution and inter-quarter relationships using box and scatter plots providing insights into performance variability and consistency has been executed successfully.

Ex.No. 6	SALES PERFORMANCE VISUALIZATION USING STORE NAMES
Date :	

AIM:

To visualize and analyze monthly sales trends across different retail stores by cleaning data, handling missing values and generating graphical insights using area charts, heatmaps and correlograms.

ALGORITHM:

Step 1: Start.

Step 2: Import the necessary libraries pandas, numpy, matplotlib, and seaborn.

Step 3: Initialize or load the dataset containing monthly sales for each store.

Step 4: Identify and remove duplicate records using .drop_duplicates().

Step 5: Handle missing values by replacing them with the mean or median of respective months using .fillna().

Step 6: Plot an Area Chart to visualize sales trends of all stores over the months (Jan–Apr).

Step 7: Compute the correlation matrix among monthly sales using .corr().

Step 8: Visualize the correlation matrix using a Heatmap to highlight strong and weak relationships between months.

Step 9: Create a Correlogram using seaborn.pairplot() to explore pairwise relationships among monthly sales figures.

Step 10: Analyze and interpret the visualizations to identify patterns and inter-month dependencies.

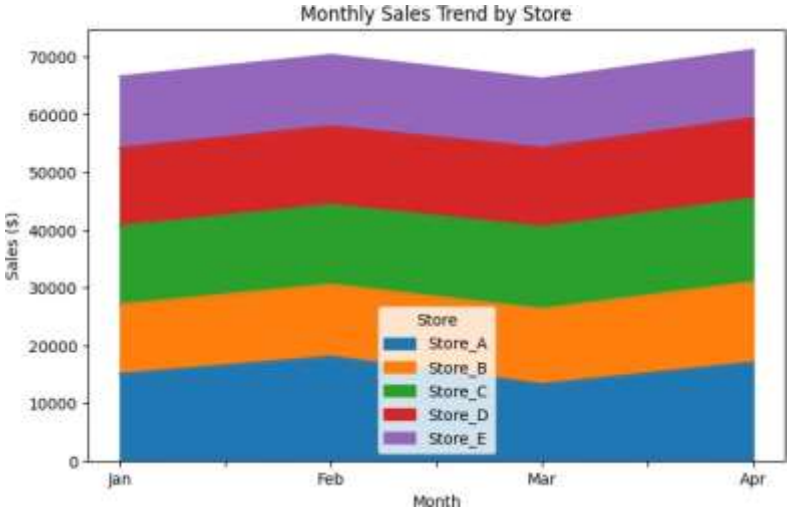
Step 11: Stop.

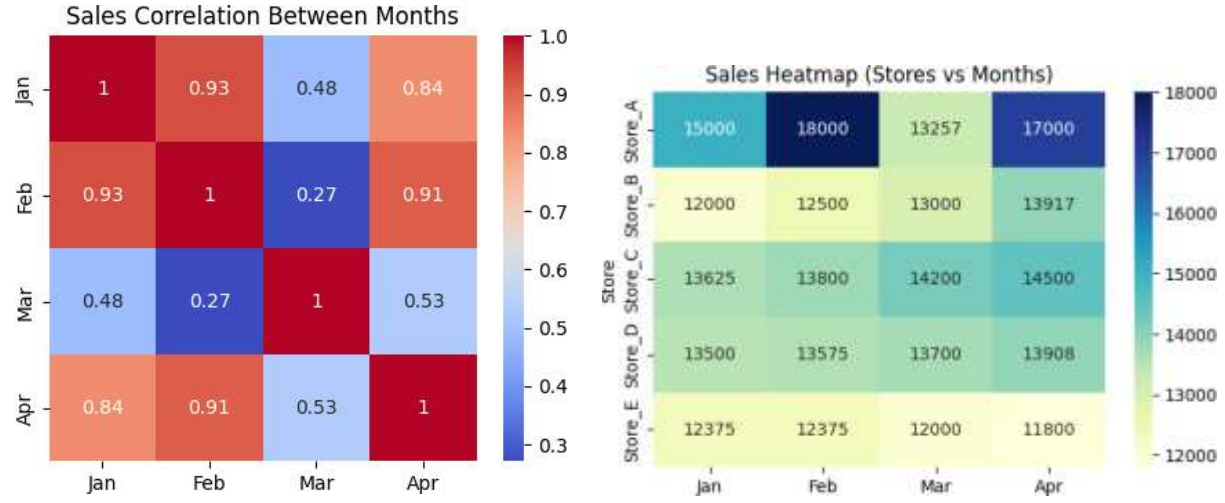
SOURCE CODE:

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

```
data = {
    'Store': ['Store_A','Store_B','Store_C','Store_D','Store_E',
              'Store_B','Store_A','Store_C','Store_D','Store_E'],
    'Jan':   [15000,12000,14000,13500,None,12000,15000,None,13500,11500],
    'Feb':   [18000,12500,13800,None,11000,12500,18000,13800,13400,None],
    'Mar':   [None,13000,14200,13700,12000,13000,None,14200,13700,12000],
    'Apr': [17000,None,14500,13900,11800,None,17000,14500,None,11800]
}
df = pd.DataFrame(data)
df = df.drop_duplicates()
df = df.fillna(df.mean(numeric_only=True))
df_group = df.groupby('Store')[['Jan','Feb','Mar','Apr']].mean()
df_group.T.plot.area(figsize=(8,5))
plt.title('Monthly Sales Trend by Store')
plt.xlabel('Month'); plt.ylabel('Sales ($)')
plt.show()
plt.figure(figsize=(6,4))
sns.heatmap(df_group, annot=True, fmt=".0f", cmap="YlGnBu")
plt.title('Sales Heatmap (Stores vs Months)')
plt.show()
plt.figure(figsize=(5,4))
sns.heatmap(df_group.corr(), annot=True, cmap="coolwarm")
plt.title('Sales Correlation Between Months')
plt.show()
```

OUTPUT:





RESULT:

Thus, the Python program to visualize and analyze monthly store sales performance using Pandas, Matplotlib and Seaborn through area charts, heatmaps and correlograms has been executed successfully.

Ex.No. 7	VISUALIZING TEXT DATA USING WORD CLOUD
Date :	

AIM:

To analyze and visualize audience sentiment from a synthetic dataset of movie reviews by generating word clouds for positive and negative sentiments, providing insights into commonly used expressions.

ALGORITHM:

Step 1: Start.

Step 2: Import the necessary libraries pandas, matplotlib and wordcloud.

Step 3: Load the movie review dataset into a Pandas DataFrame.

Step 4: Separate the dataset into positive and negative reviews based on the sentiment label (1 = Positive, 0 = Negative).

Step 5: Preprocess the text data by converting all text to lowercase also removing punctuation, numbers and stopwords.

Step 6: Generate a Word Cloud for All reviews combined, only positive reviews and only negative reviews.

Step 7: Visualize and display the word clouds using matplotlib.pyplot.

Step 8: Interpret the frequently used words in positive and negative reviews to understand sentiment trends.

Step 9: Stop.

SOURCE CODE:

```
import pandas as pd
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import string
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
data = {
```

```
    'Review': [
        "The movie was absolutely fantastic with brilliant acting",
        "A dull storyline made the film boring and predictable",
        "Loved the visuals and the music score",
        "The film was a complete waste of time",
        "An excellent performance by the lead actor",
        "Poor script and weak direction ruined the experience",
        "A fun, heartwarming movie for the entire family",
        "Disappointing and far too slow in the middle",
        "Great balance of humor and emotion throughout",
        "The dialogue felt forced and unnatural",
        "A refreshing story with strong character development",
        "The ending was confusing and unsatisfying",
        "Brilliant direction and flawless screenplay",
        "Weak plot twists and badly written characters",
        "The cinematography was stunning and breathtaking",
        "An overhyped movie with little substance",
        "Perfect blend of action, drama, and comedy",
        "The pacing was uneven and dragged in parts",
        "A powerful message delivered beautifully",
        "Forgettable acting and lack of originality",
        "Heartfelt performances made the film inspiring",
        "Too many clichés ruined the experience",
        "An emotional rollercoaster that touched the audience",
        "The sound design was poor and distracting",
        "Wonderful direction and meaningful storytelling",
        "The background score was irritating and loud",
        "Beautifully crafted movie with a touching theme",
        "Predictable plot that offered no surprises"
    ], 'Label': [1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]}
df = pd.DataFrame(data)
stop_words = set(stopwords.words('english'))
def clean_text(text):
    text = text.lower()
    text = ''.join([c for c in text if c not in string.punctuation])
    words = [w for w in text.split() if w not in stop_words]
    return ' '.join(words)
```

```
df['Cleaned'] = df['Review'].apply(clean_text)
positive_reviews = ' '.join(df[df['Label']==1]['Cleaned'])
negative_reviews = ' '.join(df[df['Label']==0]['Cleaned'])
all_reviews = ' '.join(df['Cleaned'])
plt.figure(figsize=(15,8))
plt.subplot(1,3,1)
wc_all = WordCloud(width=400, height=300, background_color='white',
colormap='viridis').generate(all_reviews)
plt.imshow(wc_all); plt.axis('off'); plt.title('All Reviews', fontsize=14)
plt.subplot(1,3,2)
wc_pos = WordCloud(width=400, height=300, background_color='white',
colormap='Greens').generate(positive_reviews)
plt.imshow(wc_pos); plt.axis('off'); plt.title('Positive Reviews', fontsize=14)
plt.subplot(1,3,3)
wc_neg = WordCloud(width=400, height=300, background_color='white',
colormap='Reds').generate(negative_reviews)
plt.imshow(wc_neg); plt.axis('off'); plt.title('Negative Reviews', fontsize=14)
plt.tight_layout()
plt.show()
```

OUTPUT :

RESULT:

Thus, the Python program to analyze and visualize movie review sentiments using word clouds by cleaning text data and displaying frequently used positive and negative expressions has been executed successfully.

Ex.No. 8	IMPLEMENTATION OF LINEAR REGRESSION
Date :	

AIM:

To analyze the relationship between TV advertising budget and sales revenue using both the Mathematical Least Squares Method and the Scikit-learn Linear Regression Model and to evaluate the model's performance using R^2.

ALGORITHM:

Step 1: Start.

Step 2: Load the dataset containing TV advertising budgets and corresponding sales values.

Step 3: Compute the mean values of TV and Sales.

Step 4: Calculate the slope (b_1) and intercept (b_0) using the Least Squares formulas, and form the regression equation $\hat{y} = b_0 + b_1 x$.

Step 5: Predict sales and calculate the R^2 value to measure model accuracy.

Step 6: Implement the Scikit-learn LinearRegression model and fit it to the same data.

Step 7: Retrieve and compare slope, intercept, and R^2 score from both methods for validation.

Step 8: Visualize the relationship using a scatter plot with the regression line.

Step 9: Stop.

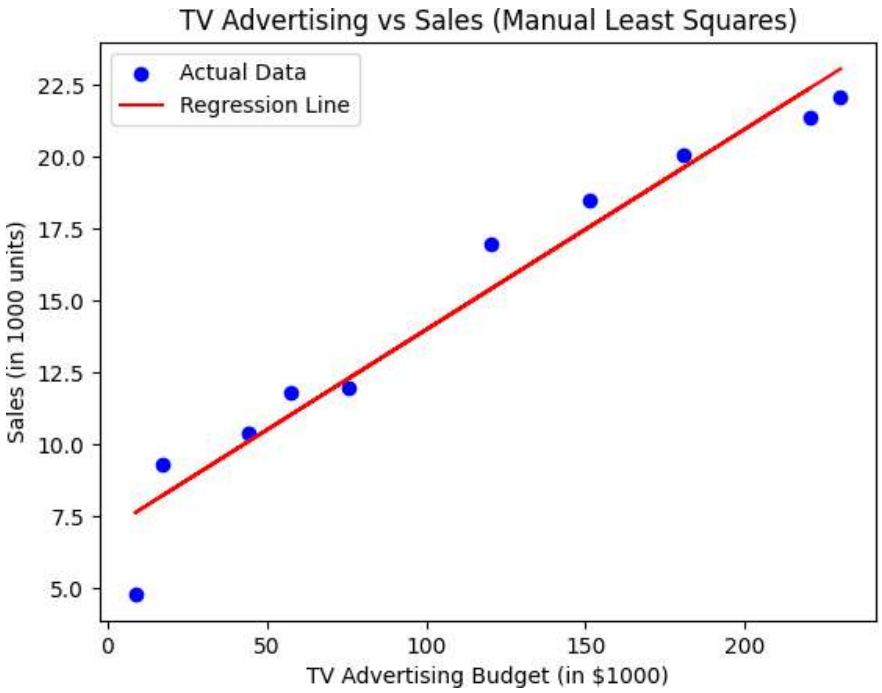
SOURCE CODE:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score


data = { 'TV': [230.1, 44.5, 17.2, 151.5, 180.8, 8.7, 57.5, 120.2, 220.3, 75.5],
   'Sales': [22.1, 10.4, 9.3, 18.5, 20.1, 4.8, 11.8, 17.0, 21.4, 12.0]}
df = pd.DataFrame(data)
x_mean = np.mean(df['TV'])
y_mean = np.mean(df['Sales'])
```

```
b1 = np.sum((df['TV'] - x_mean) * (df['Sales'] - y_mean)) / np.sum((df['TV'] - x_mean)**2)
b0 = y_mean - b1 * x_mean
print("----- Mathematical Regression -----")
print(f"Slope (b1): {b1:.4f}")
print(f"Intercept (b0): {b0:.4f}")
df['Pred_Sales_Math'] = b0 + b1 * df['TV']
ss_total = np.sum((df['Sales'] - y_mean)**2)
ss_res = np.sum((df['Sales'] - df['Pred_Sales_Math'])**2)
r2_math = 1 - (ss_res / ss_total)
print(f"R² (Mathematical): {r2_math:.4f}")
plt.scatter(df['TV'], df['Sales'], color='blue', label='Actual Data')
plt.plot(df['TV'], df['Pred_Sales_Math'], color='red', label='Regression Line')
plt.title('TV Advertising vs Sales (Manual Least Squares)')
plt.xlabel('TV Advertising Budget (in $1000)')
plt.ylabel('Sales (in 1000 units)')
plt.legend()
plt.show()
X = df[['TV']]
y = df['Sales']
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)
print("\n----- Scikit-learn Regression -----")
print(f"Slope (Coefficient): {model.coef_[0]:.4f}")
print(f"Intercept: {model.intercept_:.4f}")
print(f"R² (Scikit-learn): {model.score(X, y):.4f}")
comparison = pd.DataFrame({
    'Method': ['Mathematical', 'Scikit-learn'],
    'Slope': [b1, model.coef_[0]],
    'Intercept': [b0, model.intercept_],
    'R²': [r2_math, model.score(X, y)]
```

```
})
print("\nComparison Summary:")
print(comparison.round(4))
```

OUTPUT:




```
----- Mathematical Regression -----
Slope (b1): 0.0697
Intercept (b0): 7.0241
R² (Mathematical): 0.9502
```

```
----- Scikit-learn Regression -----
Slope (Coefficient): 0.0697
Intercept: 7.0241
R² (Scikit-learn): 0.9502

Comparison Summary:
         Method   Slope  Intercept      R²
0  Mathematical  0.0697     7.0241  0.9502
1  Scikit-learn  0.0697     7.0241  0.9502
```

RESULT:

Thus, the Python program to analyze the relationship between TV advertising and product sales using both the Mathematical Least Squares Method and the Scikit-learn Linear Regression Model has been executed successfully.

Ex.No. 9	IMPLEMENTATION OF LOGISTIC REGRESSION
Date :	

AIM:

To build & evaluate Logistic Regression model that predicts the likelihood of diabetes in patients using key medical features such as glucose level, BMI, blood pressure, insulin level and age.

ALGORITHM:

Step 1: Start.

Step 2: Load the Diabetes Dataset and inspect its structure using Pandas.

Step 3: Preprocess the data by handling missing or invalid values and ensuring all features are numerical.

Step 4: Separate the dataset into features (X) and target (y) (Outcome column).

Step 5: Split the data into training and testing sets using train_test_split().

Step 6: Initialize and train a LogisticRegression model using the training data.

Step 7: Predict diabetes outcomes for the test set and evaluate model performance using Accuracy, F1-Score, and Confusion Matrix.

Step 8: Visualize the confusion matrix and interpret the model's predictive capability.

Step 9: Stop.

SOURCE CODE:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("/content/diabetes (1).csv")   # Ensure your dataset file is in the same folder
```

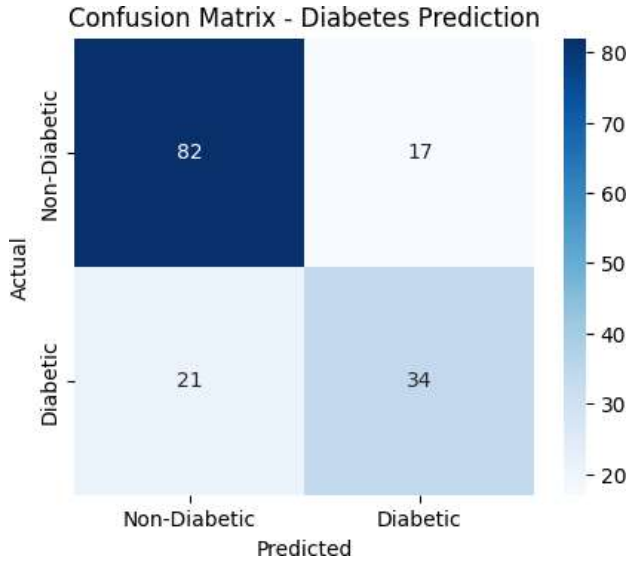
```
print("Dataset Preview:")
print(df.head(), "\n")
print("Dataset Information:")
print(df.info(), "\n")
print("Missing Values in Each Column:")
print(df.isnull().sum(), "\n")
cols_with_invalid_zeros = ['Glucose', 'BloodPressure', 'Insulin', 'BMI']
df[cols_with_invalid_zeros] = df[cols_with_invalid_zeros].replace(0, np.nan)
df = df.fillna(df.mean(numeric_only=True))
X = df[['Glucose', 'BMI', 'BloodPressure', 'Age', 'Insulin']]
y = df['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print("----- Model Evaluation -----")
print(f"Accuracy Score: {accuracy:.4f}")
print(f"F1-Score: {f1:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Non-Diabetic', 'Diabetic'],
            yticklabels=['Non-Diabetic', 'Diabetic'])
plt.title('Confusion Matrix - Diabetes Prediction')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
new_patient = pd.DataFrame([{
    'Glucose': 145,
```

```
    'BMI': 32.5,
    'BloodPressure': 82,
    'Age': 45,
    'Insulin': 130
}])
prediction = model.predict(new_patient)
print(f"\nPredicted Outcome for New Patient: {'Diabetic' if prediction[0]==1 else 'Non-Diabetic'}")
```

OUTPUT:

```
Dataset Preview:
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1

Dataset Information:
<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
None

Missing Values in Each Column:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
----- Model Evaluation -----
Accuracy Score: 0.7532
F1-Score: 0.6415

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.83      0.81        99
           1       0.67      0.62      0.64        55

    accuracy                           0.75       154
   macro avg       0.73      0.72      0.73       154
weighted avg       0.75      0.75      0.75       154
```



RESULT:

Thus, the Python program to predict diabetes occurrence using the Logistic Regression model has been executed successfully.

Ex.No. 10	IMPLEMENTATION OF SVM CLASSIFICATION TECHNIQUES
Date :	

AIM:

To implement and evaluate a Support Vector Machine (SVM) classifier for early detection of breast cancer using the Breast Cancer Wisconsin Dataset.

ALGORITHM:

Step 1: Start.

Step 2: Load the Breast Cancer Wisconsin Dataset from scikit-learn.

Step 3: Inspect the dataset to understand features and target labels (benign or malignant).

Step 4: Split the dataset into training and testing sets using train_test_split().

Step 5: Initialize an SVM classifier (e.g., SVC(kernel='linear')).

Step 6: Train the model on the training dataset using .fit(X_train, y_train).

Step 7: Predict cancer outcomes on the test set and compute the accuracy score using .score() or accuracy_score().

Step 8: Display the classification accuracy and interpret the model's effectiveness.

Step 9: Stop.

SOURCE CODE:

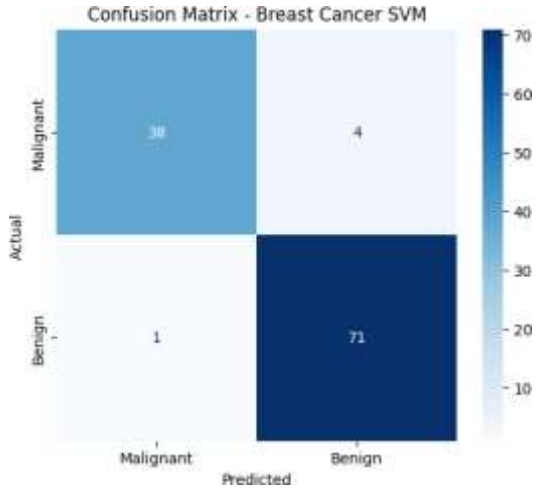
```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)
```

```
print("Dataset Preview:")
print(X.head())
print("\nTarget Distribution:")
print(y.value_counts())
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42,stratify=y)
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nSVM Classifier Accuracy: {accuracy:.4f}\n")
print("Classification Report:\n", classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Malignant','Benign'],
        yticklabels=['Malignant','Benign'])
plt.title('Confusion Matrix - Breast Cancer SVM')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

OUTPUT:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.90      0.94        42
           1       0.95      0.99      0.97        72

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

```
Dataset Preview:
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst radius  worst texture  worst perimeter  \
0                 0.07871  ...         25.38          17.33           184.60
1                 0.05667  ...         24.99          23.41           158.80
2                 0.05999  ...         23.57          25.53           152.50
3                 0.09744  ...         14.91          26.50            98.87
4                 0.05883  ...         22.54          16.67           152.20

   worst area  worst smoothness  worst compactness  worst concavity  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869
4      1575.0            0.1374             0.2050           0.4000

   worst concave points  worst symmetry  worst fractal dimension
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678

[5 rows x 30 columns]

Target Distribution:
1    357
0    212
Name: count, dtype: int64

SVM Classifier Accuracy: 0.9561
```



RESULT:

Thus, the Python program to classify breast cancer cases using the Support Vector Machine (SVM) algorithm has been executed successfully.

Ex.No. 11	IMPLEMENTATION OF DECISION TREE CLASSIFICATION TECHNIQUES
Date :	

AIM:

To build and evaluate a Decision Tree classifier using the Breast Cancer Wisconsin dataset that classifies tumors as benign or malignant and to visualize the trained tree for interpretability.

ALGORITHM:

Step 1: Start.

Step 2: Load the Breast Cancer Wisconsin dataset (from sklearn.datasets) and inspect features and target labels.

Step 3: Split the data into training and testing sets using train_test_split().

Step 4: Initialize a DecisionTreeClassifier (optionally set max_depth, criterion, and random_state).

Step 5: Train the classifier with .fit(X_train, y_train).

Step 6: Predict labels for the test set and compute accuracy using accuracy_score() (or .score()).

Step 7: Visualize the trained decision tree using plot_tree() or export_graphviz() (with feature names and class names), and display the plotted tree diagram.

Step 8: Display accuracy, interpret key splits in the plotted tree.

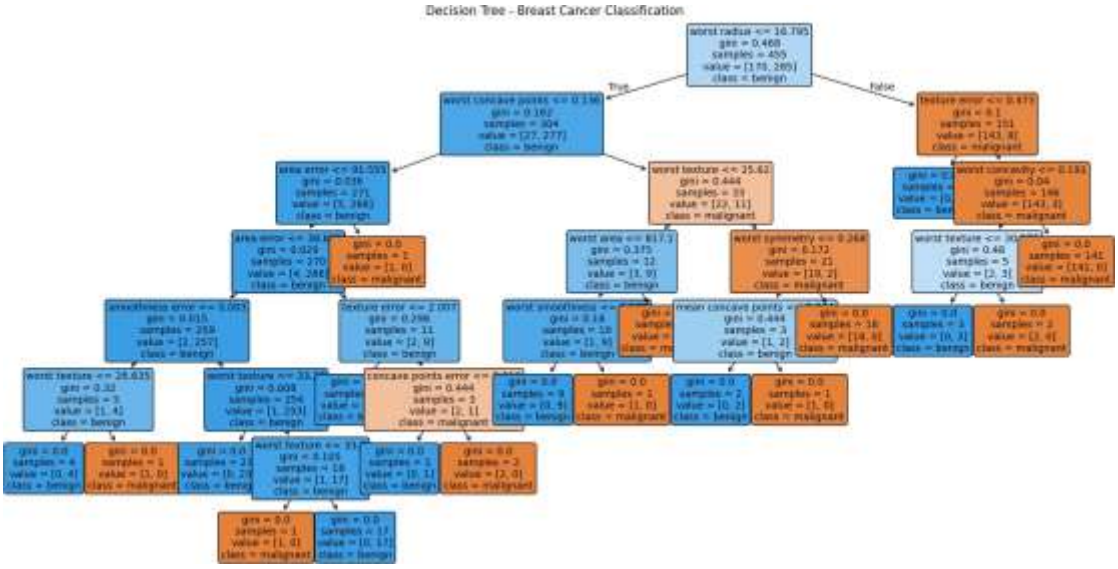
Step 9: Stop.

SOURCE CODE:

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

data = load_breast_cancer()
```

```
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)  # 0 = malignant, 1 = benign
print("Dataset Preview:")
print(X.head())
print("\nTarget Distribution:")
print(y.value_counts())
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,random_state=42, stratify=y)
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
y_pred = dt_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nDecision Tree Accuracy: {accuracy:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
plt.figure(figsize=(20,10))
plot_tree( dt_model,
    feature_names=data.feature_names,
    class_names=data.target_names,
    filled=True,
    rounded=True,
    fontsize=10
)
plt.title("Decision Tree - Breast Cancer Classification")
plt.show()
```

OUTPUT;




```
Dataset Preview:
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst radius  worst texture  worst perimeter  \
0                 0.07871  ...         25.38          17.33           184.60
1                 0.05667  ...         24.99          23.41           158.80
2                 0.05999  ...         23.57          25.53           152.50
3                 0.09744  ...         14.91          26.50            98.87
4                 0.05883  ...         22.54          16.67           152.20

   worst area  worst smoothness  worst compactness  worst concavity  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869
4      1575.0            0.1374             0.2050           0.4000

   worst concave points  worst symmetry  worst fractal dimension
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678

[5 rows x 30 columns]

Target Distribution:
1    357
0    212
Name: count, dtype: int64

Decision Tree Accuracy: 0.9123
```

RESULT:

Thus, the Python program to classify tumors using a Decision Tree classifier has been executed successfully.

Ex.No. 12	IMPLEMENTATION OF HIERARCHICAL CLUSTERING
Date :	

AIM:

To perform Hierarchical Clustering on customer data to identify distinct customer segments based on annual income and spending score, aiding in targeted marketing strategies.

ALGORITHM:

Step 1: Start.

Step 2: Import required libraries – numpy, pandas, matplotlib.pyplot, scipy.cluster.hierarchy, and sklearn.cluster.

Step 3: Load the customer dataset into a Pandas DataFrame and inspect the data.

Step 4: Select relevant features — Annual Income and Spending Score — and store them as variable X.

Step 5: Construct a dendrogram using scipy.cluster.hierarchy.linkage(X, method='ward') and plot it to determine the optimal number of clusters.

Step 6: Apply Agglomerative Clustering with the chosen number of clusters (e.g., 5) using AgglomerativeClustering(n_clusters=5, linkage='ward').

Step 7: Fit the model on X, obtain cluster labels, and assign them to the dataset.

Step 8: Visualize the resulting clusters on a scatter plot, differentiating each cluster by color and labeling axes for clarity.

Step 9: Stop.

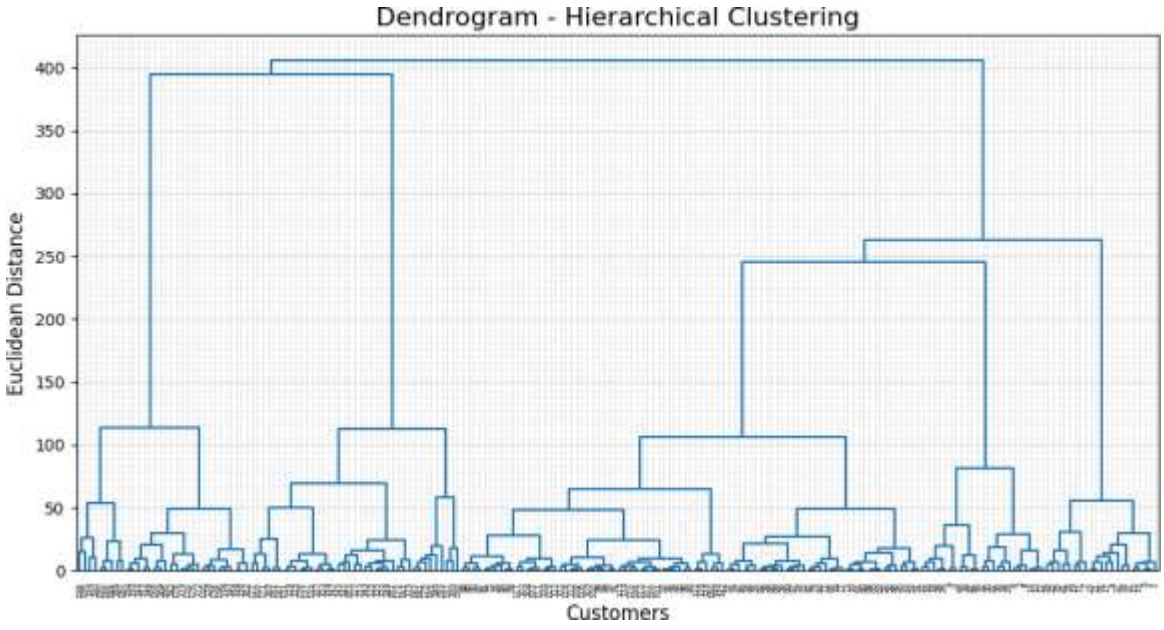
SOURCE CODE:

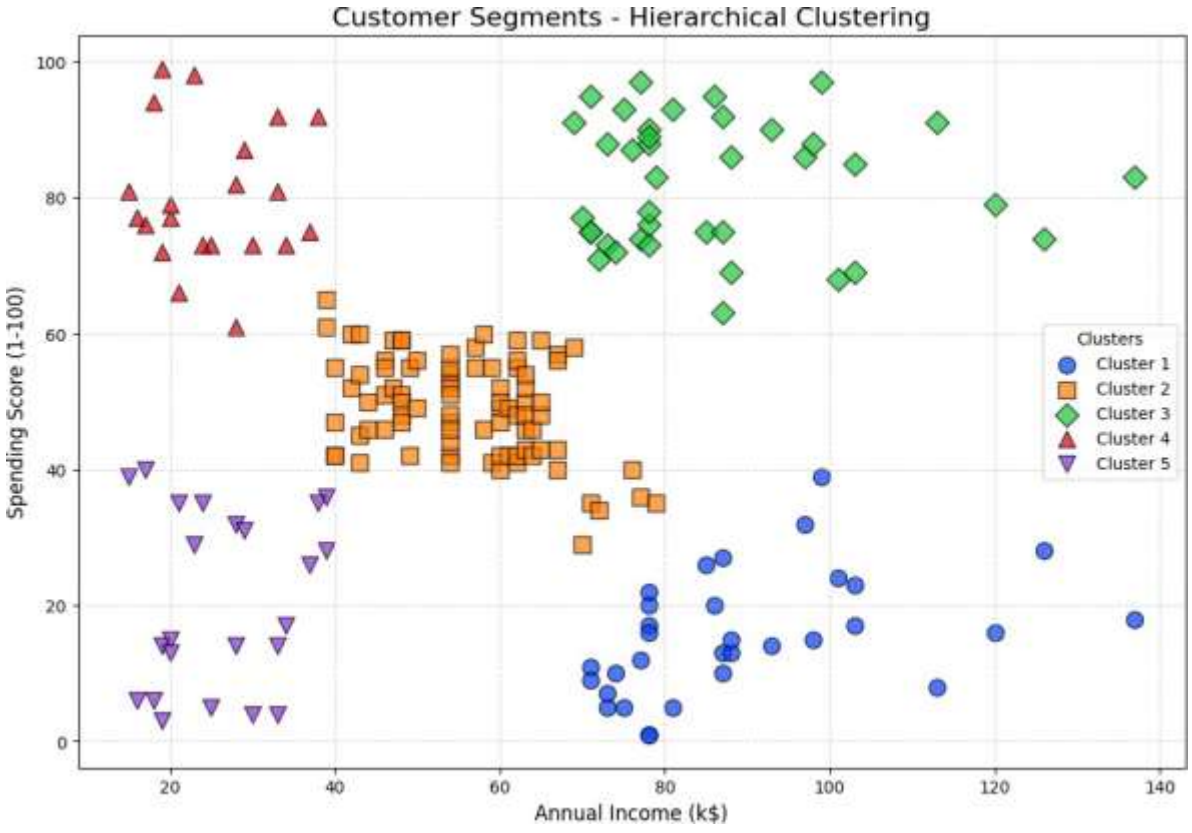
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import linkage, dendrogram
from sklearn.cluster import AgglomerativeClustering
import seaborn as sns
df = pd.read_csv("/content/Mall_Customers.csv")
print("Dataset Preview:")
```

```
n_clusters = 5
hc = AgglomerativeClustering(n_clusters=n_clusters, metric='euclidean', linkage='ward')
y_hc = hc.fit_predict(X)
plt.figure(figsize=(12, 8))
print(df.head())
X = df[['Annual Income (k$)', 'Spending Score (1-100)']].values
linked = linkage(X, method='ward')
plt.figure(figsize=(12, 6))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=False,
color_threshold=0)
plt.title('Dendrogram - Hierarchical Clustering', fontsize=16)
plt.xlabel('Customers', fontsize=12)
plt.ylabel('Euclidean Distance', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
palette = sns.color_palette("bright", n_clusters)
markers = ['o', 's', 'D', '^', 'v']
for i in range(n_clusters):
    plt.scatter(X[y_hc == i, 0], X[y_hc == i, 1],
                s=100, c=[palette[i]], marker=markers[i],
                edgecolor='k', alpha=0.7, label=f'Cluster {i+1}')

plt.title('Customer Segments - Hierarchical Clustering', fontsize=16)
plt.xlabel('Annual Income (k$)', fontsize=12)
plt.ylabel('Spending Score (1-100)', fontsize=12)
plt.legend(title='Clusters', fontsize=10)
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
```

OUTPUT:

```
Dataset Preview:
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
```





RESULT:

Thus, the Python program for performing Hierarchical Clustering on the retail store's customer dataset has been executed successfully.

Ex.No. 13	IMPLEMENTATION OF CLUSTERING TECHNIQUES K MEANS
Date :	

AIM:

To apply the K-Means Clustering algorithm to segment retail mall customers based on their annual income and spending score, enabling data-driven marketing and promotional strategies.

ALGORITHM:

Step 1: Start.

Step 2: Import necessary libraries numpy, pandas matplotlib.pyplot and sklearn.cluster.KMeans.

Step 3: Load the dataset into a Pandas DataFrame and display the first few records for verification.

Step 4: Select the features Annual Income and Spending Score and store them in variable X.

Step 5: Use the Elbow Method by plotting the within-cluster sum of squares (WCSS) against various K values to determine the optimal number of clusters.

Step 6: Initialize and apply the K-Means algorithm with the optimal K value (e.g., K=5), then fit and predict cluster labels for all customers.

Step 7: Visualize the resulting clusters using a scatter plot with different colors for each cluster and mark centroids with distinct markers.

Step 8: Interpret and label clusters to derive business insights.

Step 9: Stop.

SOURCE CODE:

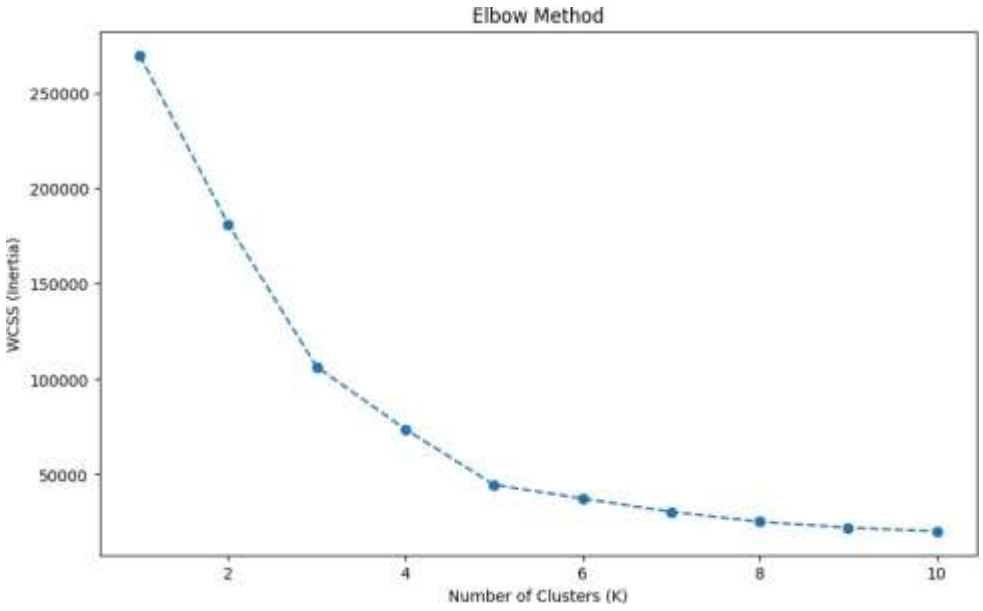
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
df = pd.read_csv('/content/Mall_Customers.csv')
X = df[['Annual Income (k$)', 'Spending Score (1-100)']].values
print("## 1. First Few Rows of the Dataset")
print(df.head())
```

```
print("\n" + "="*50 + "\n")
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42, n_init=10)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
print("## 2. Generating Elbow Method Plot...")
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS (Inertia)')
plt.show()
optimal_k = 5
kmeans = KMeans(n_clusters=optimal_k, init='k-means++', random_state=42, n_init=10)
y_kmeans = kmeans.fit_predict(X)
print("3. Cluster Assignments for Each Data Point")
df['Cluster'] = y_kmeans
print(df.head(10))
print("\n" + "="*50 + "\n")
plt.figure(figsize=(12, 8))
cluster_map = {
    0: {'label': 'Standard', 'color': 'green'},
    1: {'label': 'Target', 'color': 'red'},
    2: {'label': 'Sensible', 'color': 'orange'},
    3: {'label': 'Careless', 'color': 'purple'},
    4: {'label': 'Careful', 'color': 'blue'}
}
for i in range(optimal_k):
        plt.scatter(X[y_kmeans == i, 0], X[y_kmeans == i, 1], s=100, c=cluster_map[i]['color'],
label=cluster_map[i]['label'])
```

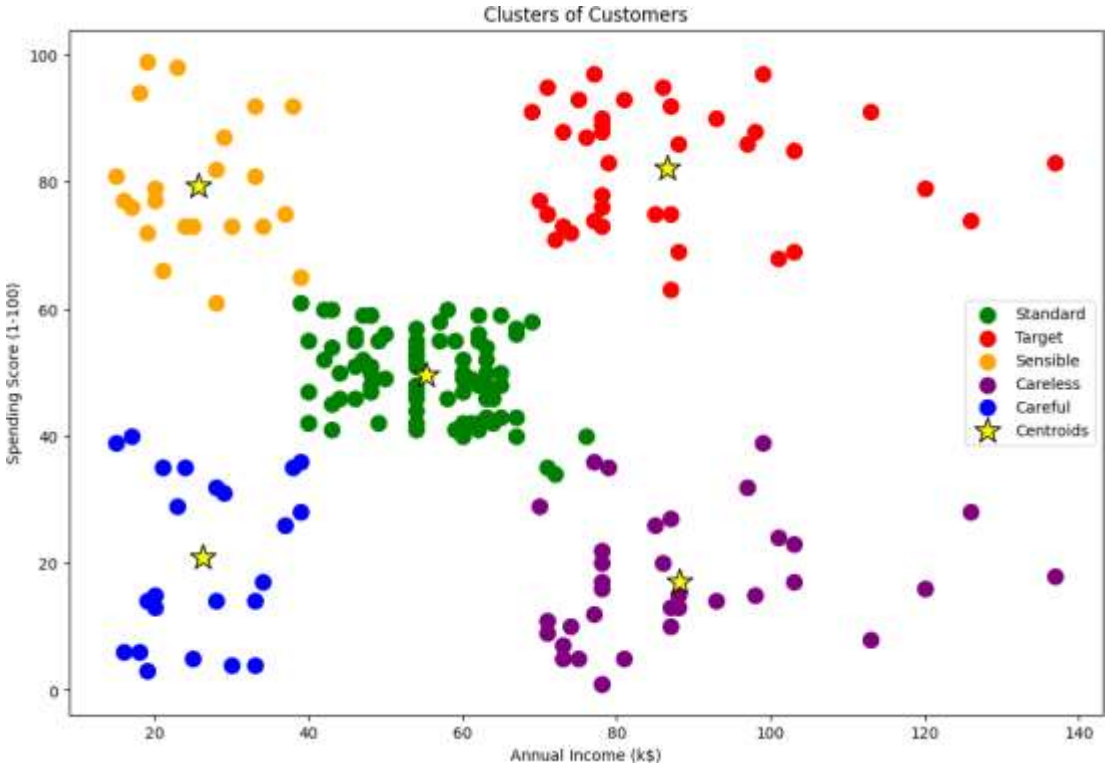
```
#centroid
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='yellow', marker='*', label='Centroids', edgecolors='black')
plt.title('Clusters of Customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

OUTPUT:

```
## 1. First Few Rows of the Dataset
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
```



```
## 3. Cluster Assignments for Each Data Point
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)  \
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
5           6  Female   22                  17                      76
6           7  Female   35                  18                       6
7           8  Female   23                  18                      94
8           9    Male   64                  19                       3
9          10  Female   30                  19                      72

   Cluster
0        4
1        2
2        4
3        2
4        4
5        2
6        4
7        2
8        4
9        2
```



RESULT:

Thus, the Python program for K-Means Clustering has been executed successfully.