EXPERIMENT - 1

AIM : Program to implement various array and math's operation.

PROGRAM :

```
def maxIN( arr ) :
   print( f"Maximum in array : {arr} is {max(arr)}")

def minIN( arr ) :
   print( f"Minimum in array : {arr} is {min(arr)}")

def sumOfElements( arr ) :
   print( f"Sum of elements : {arr} is {sum(arr)}" )

def avg( arr ) :
   print( f"Average of array : {arr} is {sum(arr)/len(arr)}")

def squareRoot( arr ) :
   print( f"Square root of {arr[-1]} is {arr[-1]**0.5}" )
   print( f"Round-Off value {round(arr[-1]**0.5)}" )

arr = list(map(int, input("Enter array:").split()))
if len(arr)>0 :
   maxIN(arr)
   minIN(arr)
   sumOfElements(arr)
   avg(arr)
   squareRoot(arr)
else :
   print("Empty array")
```

OUTPUT :

```
Enter array:4 8 7 1 9 10
Maximum in array : [4, 8, 7, 1, 9, 10] is 10
Minimum in array : [4, 8, 7, 1, 9, 10] is 1
Sum of elements : [4, 8, 7, 1, 9, 10] is 39
Average of array : [4, 8, 7, 1, 9, 10] is 6.5
Square root of 10 is 3.1622776601683795
Round-Off value 3
```

EXPERIMENT - 2

AIM : Program to implement various statistical operation.

PROGRAM :

```
import numpy as np

def findMean( arr ) :
   print( f"Mean is : {np.mean(arr)}")

def findMedian( arr ) :
   print( f"Median is : {np.median(arr)}")

def findMode( arr ) :
   unique_elements, counts = np.unique(arr, return_counts=True)
   mode_index = np.argmax(counts)
   mode = unique_elements[mode_index]
   print( f"Mode is : {mode}")

def findStandardDeviation( arr ) :
   print( f"Standard Deviation is : {np.std(arr)}")

arr = list(map(int, input("Enter array:").split()))
if len(arr)>0 :
   print( f"For array {arr}")
   findMean(arr)
   findMedian(arr)
   findMode(arr)
   findStandardDeviation(arr)
else :
   print("Empty array")
```

OUTPUT :

```
Enter array:1 8 9 8 9 2 4 0 1
For array [1, 8, 9, 8, 9, 2, 4, 0, 1]
Mean is : 4.666666666666667
Median is : 4.0
Mode is : 1
Standard Deviation is : 3.5901098714230026
```

EXPERIMENT - 3

AIM : Program to implement data handling operation over .csv .

PROGRAM :

a) Display Region and sales.

```
import pandas as pd

data = {
    "Name" : [ 'Willam', 'Willam', 'Emma', 'Emma', 'Anika', 'Anika'],
    "Region" : [ 'East', 'East', 'North', 'West', 'East', 'East'],
    "Sales" : [ 50000, 50000, 52000, 52000, 65000, 72000],
    "Expense" : [42000, 42000, 43000, 43000, 44000, 53000]
}
df = pd.DataFrame( data )
```

```
df[ ["Region", "Sales" ]]
```

	Region	Sales
0	East	50000
1	East	50000
2	North	52000
3	West	52000
4	East	65000
5	East	72000

b) Use loc and iloc to locate data of row 2 and column 3.

```
df.loc[1,"Sales"]
```

50000

```
df.iloc[1,2]
```

50000

c) Add new row in dataframe.

```
new_row = { "Name" : "ABC", "Region" : "South", "Sales" : 80000, "Expense" : 40000 }
df = pd.concat([df, pd.DataFrame([new_row])], ignore_index=True)
df
```

	Name	Region	Sales	Expense
0	Willam	East	50000	42000
1	Willam	East	50000	42000
2	Emma	North	52000	43000
3	Emma	West	52000	43000
4	Anika	East	65000	44000
5	Anika	East	72000	53000
6	ABC	South	80000	40000

d) Add new column in dataframe.

```
df['Gender'] = ['Male', 'Male', 'Female', 'Female', 'Female', 'Female', 'Male']
df
```

	Name	Region	Sales	Expense	Gender
0	Willam	East	50000	42000	Male
1	Willam	East	50000	42000	Male
2	Emma	North	52000	43000	Female
3	Emma	West	52000	43000	Female
4	Anika	East	65000	44000	Female
5	Anika	East	72000	53000	Female
6	ABC	South	80000	40000	Male

e) Change column name from "Name" to "Ename".

```
df.rename(columns={'Name': 'Ename'}, inplace=True)
df
```

	Ename	Region	Sales	Expense	Gender
0	Willam	East	50000	42000	Male
1	Willam	East	50000	42000	Male
2	Emma	North	52000	43000	Female
3	Emma	West	52000	43000	Female
4	Anika	East	65000	44000	Female
5	Anika	East	72000	53000	Female
6	ABC	South	80000	40000	Male

f) Use Groupby method.

```
df.groupby(["Gender"]).sum()
```

	Sales	Expense
Gender		
Female	241000	183000
Male	180000	124000

```
df.groupby(["Gender"]).mean()
```

	Sales	Expense
Gender		
Female	60250.0	45750.000000
Male	60000.0	41333.333333

```
df.groupby(["Gender"]).std()
```

	Sales	Expense
Gender		
Female	9945.685832	4856.267428
Male	17320.508076	1154.700538

EXPERIMENT - 4

AIM : a) Program to perform missing data handling.

PROGRAM :

```
import pandas as pd
import numpy as np

data = {'Name': ['Alice', 'Bob', np.nan, 'Charlie', 'David'],
        'Age': [25, 30, np.nan, 35, 40],
        'City': ['New York', 'San Francisco', 'Los Angeles', np.nan, 'Seattle']}

df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)

print("\nMissing Values:")
print(df.isnull())

df_dropped = df.dropna()

df_filled = df.fillna(value='Unknown')

numeric_columns = df.select_dtypes(include=np.number).columns
df_mean_filled = df.copy()
df_mean_filled[numeric_columns] =
df_mean_filled[numeric_columns].fillna(df_mean_filled[numeric_columns].mean())

print("\nDataFrame after dropping rows with missing values:")
print(df_dropped)

print("\nDataFrame after filling missing values with 'Unknown':")
print(df_filled)

print("\nDataFrame after filling missing values with the mean of numeric columns:")
print(df_mean_filled)
```

OUTPUT :

```
Original DataFrame:
      Name   Age           City
0    Alice  25.0       New York
1      Bob  30.0  San Francisco
2      NaN   NaN    Los Angeles
3  Charlie  35.0            NaN
4    David  40.0        Seattle

Missing Values:
    Name    Age   City
0  False  False  False
1  False  False  False
2   True   True  False
3  False  False   True
4  False  False  False

DataFrame after dropping rows with missing values:
    Name   Age           City
0  Alice  25.0       New York
1    Bob  30.0  San Francisco
4  David  40.0        Seattle

DataFrame after filling missing values with 'Unknown':
      Name      Age           City
0    Alice     25.0       New York
1      Bob     30.0  San Francisco
2  Unknown  Unknown    Los Angeles
3  Charlie     35.0        Unknown
4    David     40.0        Seattle

DataFrame after filling missing values with the mean of numeric columns:
      Name   Age           City
0    Alice  25.0       New York
1      Bob  30.0  San Francisco
2      NaN  32.5    Los Angeles
3  Charlie  35.0            NaN
4    David  40.0        Seattle
```

AIM : b) Program to perform min-max normalization.

PROGRAM :

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

data = {'Feature1': [10, 20, 30, 40, 50],
      'Feature2': [5, 15, 25, 35, 45]}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
scaler = MinMaxScaler()

normalized_data = scaler.fit_transform(df)
normalized_df = pd.DataFrame(normalized_data, columns=df.columns)

print("\nDataFrame after Min-Max normalization:")
print(normalized_df)
```

OUTPUT :

```
Original DataFrame:
   Feature1  Feature2
0        10         5
1        20        15
2        30        25
3        40        35
4        50        45

DataFrame after Min-Max normalization:
   Feature1  Feature2
0      0.00      0.00
1      0.25      0.25
2      0.50      0.50
3      0.75      0.75
4      1.00      1.00
```

EXPERIMENT - 5

AIM : Program to perform dimensionality reduction (PCA/SVM).

PROGRAM :

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_diabetes

data = load_diabetes()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
svm_classifier = SVC(kernel='linear', random_state=42)
svm_classifier.fit(X_train_std, y_train)
y_pred = svm_classifier.predict(X_test_std)
accuracy_no_reduction = accuracy_score(y_test, y_pred)
print(f"Accuracy without dimensionality reduction: {accuracy_no_reduction:.2f}")

pca = PCA(n_components=0.95, random_state=42)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)
svm_classifier_pca = SVC(kernel='linear', random_state=42)
svm_classifier_pca.fit(X_train_pca, y_train)
y_pred_pca = svm_classifier_pca.predict(X_test_pca)
accuracy_with_reduction = accuracy_score(y_test, y_pred_pca)
print(f"Accuracy with dimensionality reduction: {accuracy_with_reduction:.2f}")
```

OUTPUT :

```
Accuracy without dimensionality reduction: 0.01
Accuracy with dimensionality reduction: 0.00
```

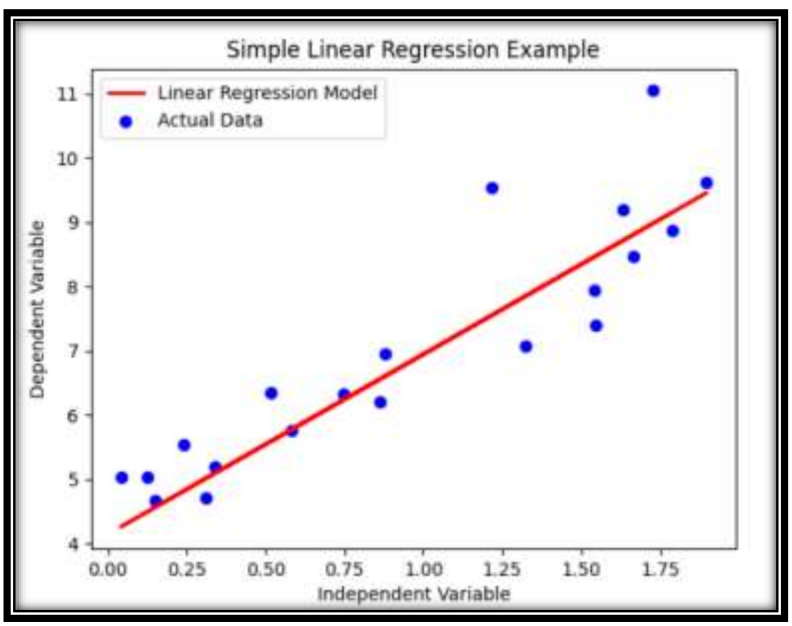
EXPERIMENT - 6

AIM : Program to implement simple Linear regression operation.

PROGRAM :

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
plt.scatter(X_test, y_test, color='blue', label='Actual Data')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Linear Regression Model')
plt.xlabel('Independent Variable')
plt.ylabel('Dependent Variable')
plt.title('Simple Linear Regression Example')
plt.legend()
plt.show()
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
```

OUTPUT :



EXPERIMENT - 7

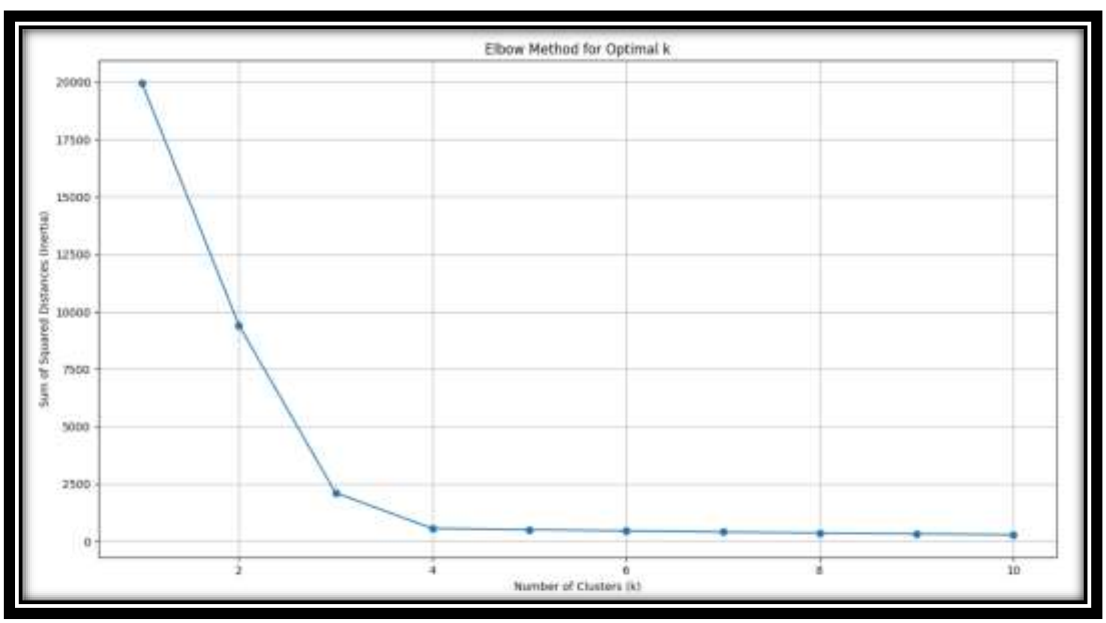
AIM : Program to perform K-Means clustering.

PROGRAM :

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
data, _ = make_blobs(n_samples=300, centers=4, random_state=42)
def calculate_inertia(data, k_range):
    inertia_values = []
    for k in k_range:
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(data)
        inertia_values.append(kmeans.inertia_)
    return inertia_values

k_values = range(1, 11)
inertia_values = calculate_inertia(data, k_values)
plt.figure(figsize=(8, 5))
plt.plot(k_values, inertia_values, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Sum of Squared Distances (Inertia)')
plt.grid(True)
plt.show()
```

OUTPUT :



EXPERIMENT - 8

AIM : Program to perform market basket analysis using Association rule.

PROGRAM :

```
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
import pandas as pd

# Sample transaction data (replace this with your dataset)
data = {'Transaction': [1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 4, 4],
      'Item': ['A', 'B', 'C', 'A', 'D', 'A', 'B', 'C', 'D', 'A', 'B', 'D']}
df = pd.DataFrame(data)

# Convert data to one-hot encoded format
basket = pd.crosstab(df['Transaction'], df['Item'], dropna=False)
basket = (basket > 0).astype(int)

# Apply Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(basket, min_support=0.2, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

# Display the results
print("Frequent Itemsets:")
print(frequent_itemsets)

print("\nAssociation Rules:")
print(rules)
```

OUTPUT :

Frequent Itemsets:

	support	itemsets
0	1.00	(A)
1	0.75	(B)
2	0.50	(C)
3	0.75	(D)
4	0.75	(B, A)
5	0.50	(A, C)
6	0.75	(A, D)
7	0.50	(B, C)
8	0.50	(B, D)
9	0.25	(C, D)
10	0.50	(B, A, C)
11	0.50	(B, A, D)
12	0.25	(A, C, D)
13	0.25	(B, C, D)
14	0.25	(B, A, C, D)

Association Rules:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(B)	(A)	0.75	1.00	0.75	1.00	1.000000	0.0000	inf	0.000000
1	(A)	(B)	1.00	0.75	0.75	0.75	1.000000	0.0000	1.0	0.000000
2	(C)	(A)	0.50	1.00	0.50	1.00	1.000000	0.0000	inf	0.000000
3	(A)	(D)	1.00	0.75	0.75	0.75	1.000000	0.0000	1.0	0.000000
4	(D)	(A)	0.75	1.00	0.75	1.00	1.000000	0.0000	inf	0.000000
5	(C)	(B)	0.50	0.75	0.50	1.00	1.333333	0.1250	inf	0.500000
6	(B, C)	(A)	0.50	1.00	0.50	1.00	1.000000	0.0000	inf	0.000000
7	(A, C)	(B)	0.50	0.75	0.50	1.00	1.333333	0.1250	inf	0.500000
8	(C)	(B, A)	0.50	0.75	0.50	1.00	1.333333	0.1250	inf	0.500000
9	(B, D)	(A)	0.50	1.00	0.50	1.00	1.000000	0.0000	inf	0.000000
10	(C, D)	(A)	0.25	1.00	0.25	1.00	1.000000	0.0000	inf	0.000000
11	(C, D)	(B)	0.25	0.75	0.25	1.00	1.333333	0.0625	inf	0.333333
12	(B, C, D)	(A)	0.25	1.00	0.25	1.00	1.000000	0.0000	inf	0.000000
13	(A, C, D)	(B)	0.25	0.75	0.25	1.00	1.333333	0.0625	inf	0.333333
14	(C, D)	(B, A)	0.25	0.75	0.25	1.00	1.333333	0.0625	inf	0.333333

EXPERIMENT - 9

AIM : Program to perform classification algorithms.

PROGRAM :

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
from mlxtend.plotting import plot_decision_regions  # Install mlxtend with: pip install
mlxtend

iris = datasets.load_iris()
X = iris.data[:, :2]  # Use only the first two features for visualization
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)

def train_evaluate_visualize_classifier(classifier, X_train, y_train, X_test, y_test, title):
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy:.2f}")
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    plt.figure(figsize=(8, 6))
    plot_decision_regions(X_train, y_train, clf=classifier, legend=2)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title(title)
    plt.show()

# Logistic Regression
logreg = LogisticRegression(random_state=42)
train_evaluate_visualize_classifier(logreg, X_train_std, y_train, X_test_std, y_test,
"Logistic Regression")
```

```
# k-Nearest Neighbors (KNN)
knn = KNeighborsClassifier(n_neighbors=5)
train_evaluate_visualize_classifier(knn, X_train_std, y_train, X_test_std, y_test, "k-Nearest Neighbors (KNN)")

# Naive Bayes
nb = GaussianNB()
train_evaluate_visualize_classifier(nb, X_train_std, y_train, X_test_std, y_test, "Naive Bayes")

# Support Vector Machine (SVM)
svm = SVC(kernel='linear', C=1.0, random_state=42)
train_evaluate_visualize_classifier(svm, X_train_std, y_train, X_test_std, y_test, "Support Vector Machine (SVM)")
```

OUTPUT :