Department of computer science

Foundation of Data Science-BCA Ⅲ

List of Experiments:

1. Perform Arithmetic Operations on NumPy Arrays
2. Filter Rows Based on Condition in Pandas
3. Frequency Count using Value Counts in Pandas
4. Write a python program for
   a) Conditional statements
   b) Loops
   c) Functions
5. Data Loading, Storage
6. Data Cleaning and Preparation.
7. Data Manipulation with Pandas.
8. Plotting and Visualization.
9. Advanced Numpy.
10. Matplotlib
11. Building and optimizing pipelines in scikit-learn.
12. BOX PLOT: Write a Python program using Matplotlib to create a boxplot that shows the distribution of marks for students in three subjects: Math, Science, and English. Add appropriate labels, a title, and display the plot.

Program 1: Perform Arithmetic Operations on NumPy Arrays

Code:
```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
print("Sum:", a + b)
print("Product:", a * b)
```

Output:

Sum: [5 7 9]

Product: [4 10 18]

Program 2: Filter Rows Based on Condition in Pandas

```
import pandas as pd
df = pd.DataFrame({'Name': ['A', 'B', 'C'], 'Marks': [45, 80, 65]})
print("Marks > 60:\n", df[df['Marks'] > 60])
```

Output:

```
Marks > 60:
  Name  Marks
1    B     80
2    C     65
```

Program 3: Frequency Count using Value Counts in Pandas

Code:

```
import pandas as pd
df = pd.DataFrame({'Fruit': ['Apple', 'Banana', 'Apple', 'Orange', 'Banana']})
print(df['Fruit'].value_counts())
```

Output:

```
Fruit
Apple      2
Banana    2
Orange    1
Name: count, dtype: int64
```

Program 4: Write a python program for

a) Conditional statements

b) Loops

c) Functions

Code:

Conditional statements:

```
num = int(input("Enter a number: "))
if num > 0:
   print("The number is Positive")
elif num < 0:
   print("The number is Negative")
else:
   print("The number is Zero")
```

Output:

```
Enter a number: -7
The number is Negative
```

Loops:

```
for i in range(1, 11):
   print(i)
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Functions:

```
def square(num):
   return num * num

n = int(input("Enter a number: "))
print("Square of", n, "is", square(n))
```

Output:

```
Enter a number: 5
Square of 5 is 25
```

Program 5: Data Loading, Storage

Code:

```
import pandas as pd
from io import StringIO
data = {'Name': ['Ali', 'Sara', 'John'], 'Marks': [85, 90, 78]}
df = pd.DataFrame(data)
# Simulate CSV file using StringIO
csv_buffer = StringIO()
df.to_csv(csv_buffer, index=False)
csv_buffer.seek(0)
# Load data from the simulated CSV
```

```
df_loaded = pd.read_csv(csv_buffer)
print("Original Data:")
print(df)
print("\nLoaded Data from CSV:")
print(df_loaded)
```

Output:

```
Original Data:
   Name  Marks
0   Ali     85
1  Sara     90
2  John     78

Loaded Data from CSV:
   Name  Marks
0   Ali     85
1  Sara     90
2  John     78
```

Program 6: Data Cleaning and Preparation.

Code:

```
import pandas as pd
data = {
    'Name': ['Ali', 'Sara', None, 'John', 'Ali'],
    'Marks': [85, 90, None, 78, 150],
    'City': ['Delhi', 'Mumbai', 'Delhi', None, 'Delhi']
}
df = pd.DataFrame(data)
print("Original Data:")
print(df)
# Remove duplicates
df = df.drop_duplicates()
# Handle missing values
df = df.fillna({'Name': 'Unknown', 'Marks': df['Marks'].mean(), 'City':
'Unknown'})
# Rename columns
```

```
df = df.rename(columns={'Name': 'Student_Name', 'Marks': 'Score', 'City':
'Location'})
# Remove outliers (Marks > 100)
df = df[df['Score'] <= 100]
# Convert names to title case
df['Student_Name'] = df['Student_Name'].str.title()
print("\nCleaned & Prepared Data:")
print(df)
```

Output:

```
Original Data:
   Name  Marks    City
0   Ali   85.0   Delhi
1  Sara   90.0  Mumbai
2  None    NaN   Delhi
3  John   78.0    None
4   Ali  150.0   Delhi

Cleaned & Prepared Data:
  Student_Name  Score Location
0         Ali   85.0   Delhi
1        Sara   90.0  Mumbai
3        John   78.0 Unknown
2     Unknown   84.3   Delhi
```

Program 7: Data Manipulation with Pandas.

Code:

```
import pandas as pd

data = {
   'Name': ['Ali', 'Sara', 'John', 'Emma'],
   'Marks': [85, 90, 78, 92],
   'City': ['Delhi', 'Mumbai', 'Delhi', 'Chennai']
}
df = pd.DataFrame(data)
```

```
print("Original Data:")
print(df)
# Select specific columns
print("\nSelect Name and Marks:")
print(df[['Name', 'Marks']])
# Filter rows where Marks > 80
print("\nStudents with Marks > 80:")
print(df[df['Marks'] > 80])
# Sort by Marks descending
print("\nSorted by Marks (Descending):")
print(df.sort_values(by='Marks', ascending=False))
# Add a new column (Grade)
df['Grade'] = ['B', 'A', 'C', 'A']
print("\nAfter Adding Grade Column:")
print(df)
# Group by City and calculate average Marks
print("\nAverage Marks by City:")
print(df.groupby('City')['Marks'].mean())
```

Output:

```
Original Data:
   Name  Marks    City
0   Ali     85   Delhi
1  Sara     90  Mumbai
2  John     78   Delhi
3  Emma     92 Chennai

Select Name and Marks:
   Name  Marks
0   Ali     85
1  Sara     90
2  John     78
3  Emma     92

Students with Marks > 80:
```

```
   Name  Marks    City
0    Ali     85   Delhi
1   Sara     90  Mumbai
3   Emma     92 Chennai
```

Sorted by Marks (Descending):

```
   Name  Marks    City
3   Emma     92 Chennai
1   Sara     90  Mumbai
0    Ali     85   Delhi
2   John     78   Delhi
```

After Adding Grade Column:

```
   Name  Marks    City Grade
0    Ali     85   Delhi     B
1   Sara     90  Mumbai     A
2   John     78   Delhi     C
3   Emma     92 Chennai     A
```

Average Marks by City:

```
City
Chennai    92.0
Delhi      81.5
Mumbai     90.0
Name: Marks, dtype: float64
```

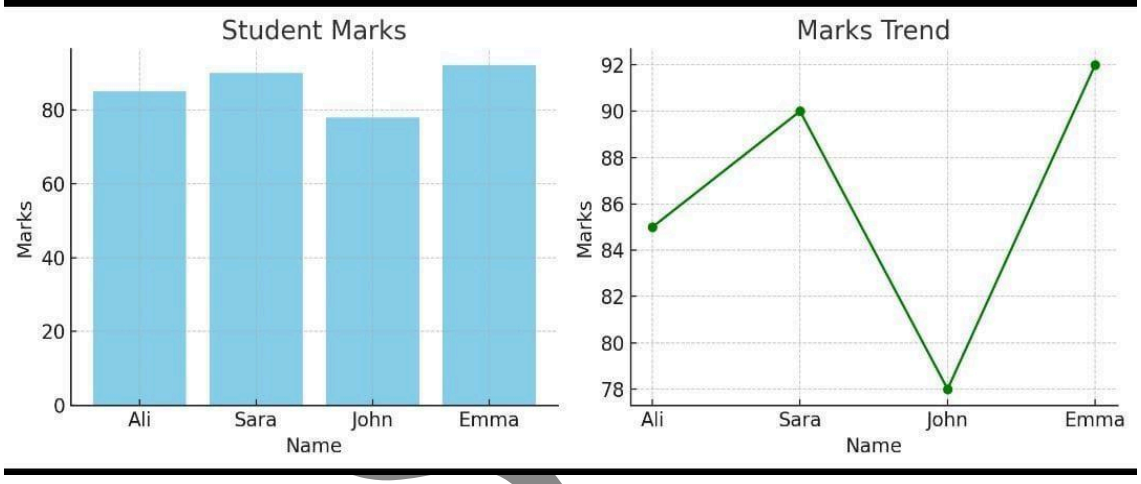
Program 8: Plotting and Visualization.

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
data = {'Name': ['Ali', 'Sara', 'John', 'Emma'],
        'Marks': [85, 90, 78, 92]}
df = pd.DataFrame(data)
# Bar Plot
plt.figure(figsize=(6,4))
```

```
plt.bar(df['Name'], df['Marks'], color='skyblue')
plt.title('Student Marks')
plt.xlabel('Name')
plt.ylabel('Marks')
plt.show()
# Line Plot
plt.figure(figsize=(6,4))
plt.plot(df['Name'], df['Marks'], marker='o', color='green')
plt.title('Marks Trend')
plt.xlabel('Name')
plt.ylabel('Marks')
plt.show()
```

Output:



Program 9: Advanced Numpy.

Code:

```
import numpy as np
# 1. Create a 2D array
data = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])
print("Original Array:\n", data)
# 2. Broadcasting (Add 5 to every element)
broadcast_result = data + 5
print("\nAfter Broadcasting (Add 5):\n", broadcast_result)
# 3. Boolean Indexing (Find values > 50)
```

```
greater_than_50 = data[data > 50]
print("\nValues greater than 50:\n", greater_than_50)
# 4. Fancy Indexing (Select specific rows and columns)
fancy_result = data[[0, 2], [1, 2]]  # (row0,col1) and (row2,col2)
print("\nFancy Indexing Result:\n", fancy_result)
# 5. Vectorized Operation (Square of all elements)
squared = np.square(data)
print("\nSquared Elements:\n", squared)
# 6. Aggregate Functions (mean, sum, std)
print("\nMean:", np.mean(data))
print("Sum:", np.sum(data))
print("Standard Deviation:", np.std(data))
# 7. Reshape (Convert 3x3 to 1x9)
reshaped = data.reshape(1, 9)
print("\nReshaped Array (1x9):\n", reshaped)
```

Output:

```
Original Array:
 [[10 20 30]
 [40 50 60]
 [70 80 90]]
After Broadcasting (Add 5):
 [[15 25 35]
 [45 55 65]
 [75 85 95]]
Values greater than 50:
 [60 70 80 90]
Fancy Indexing Result:
 [20 90]
Squared Elements:
 [[ 100  400  900]
 [1600 2500 3600]
 [4900 6400 8100]]
Mean: 50.0
Sum: 450
```

Standard Deviation: 25.81988897471611
Reshaped Array (1x9):
[[10 20 30 40 50 60 70 80 90]]

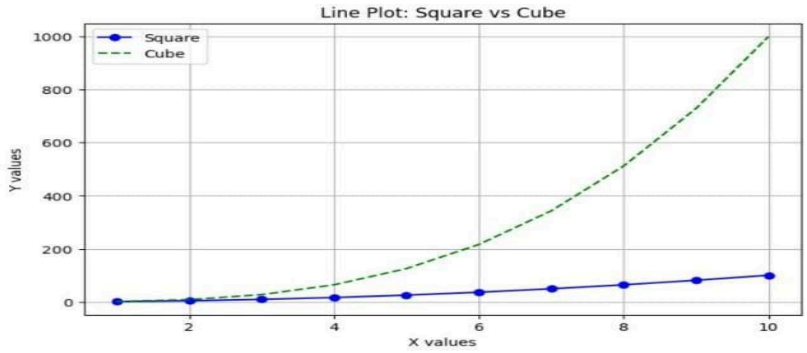
Program 10: Matplotlib

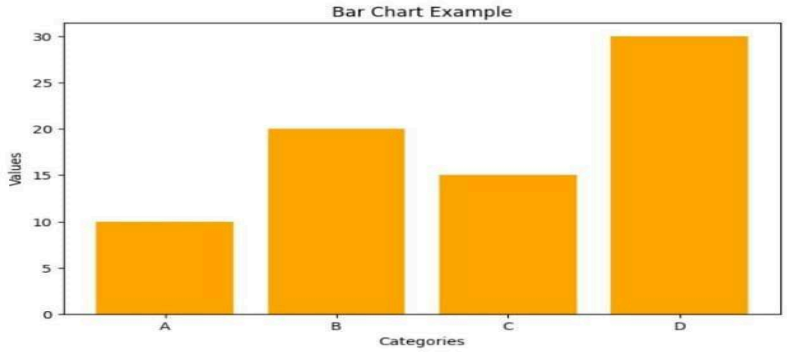
Code:

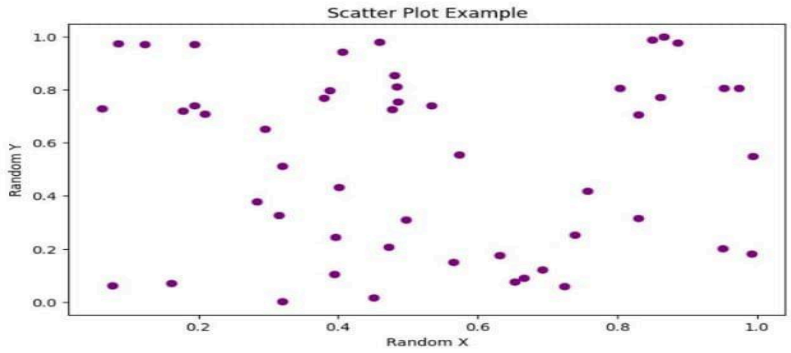
```
import matplotlib.pyplot as plt
import numpy as np
# Sample Data
x = np.arange(1, 11)
y1 = x ** 2
y2 = x ** 3
# 1. Line Plot
plt.figure(figsize=(8, 5))
plt.plot(x, y1, label='Square', color='blue', marker='o')
plt.plot(x, y2, label='Cube', color='green', linestyle='--')
plt.title("Line Plot: Square vs Cube")
plt.xlabel("X values")
plt.ylabel("Y values")
plt.legend()
plt.grid(True)
plt.show()
# 2. Bar Chart
plt.figure(figsize=(8, 5))
categories = ['A', 'B', 'C', 'D']
values = [10, 20, 15, 30]
plt.bar(categories, values, color='orange')
plt.title("Bar Chart Example")
plt.xlabel("Categories")
plt.ylabel("Values")
plt.show()
# 3. Scatter Plot
plt.figure(figsize=(8, 5))
x_scatter = np.random.rand(50)
y_scatter = np.random.rand(50)
```

```
plt.scatter(x_scatter, y_scatter, color='purple')
plt.title("Scatter Plot Example")
plt.xlabel("Random X")
plt.ylabel("Random Y")
plt.show()
```

Output:







Program 11: Building and optimizing pipelines in scikit-learn.

Code:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.pipeline import Pipeline

# 1. Load dataset
iris = load_iris()
X, y = iris.data, iris.target

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

# 3. Create a pipeline: Scaling + Logistic Regression
pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Feature scaling
    ('model', LogisticRegression(max_iter=200))  # Model
])

# 4. Define hyperparameter grid for optimization
param_grid = {
    'model__C': [0.01, 0.1, 1, 10],      # Regularization strength
    'model__solver': ['lbfgs', 'liblinear']  # Solver options
}

# 5. Use GridSearchCV for optimization
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# 6. Print best parameters and scores
print("Best Parameters:", grid_search.best_params_)
print("Best CV Score:", grid_search.best_score_)

# 7. Test set accuracy
test_accuracy = grid_search.score(X_test, y_test)
print("Test Accuracy:", test_accuracy)
```

Output:

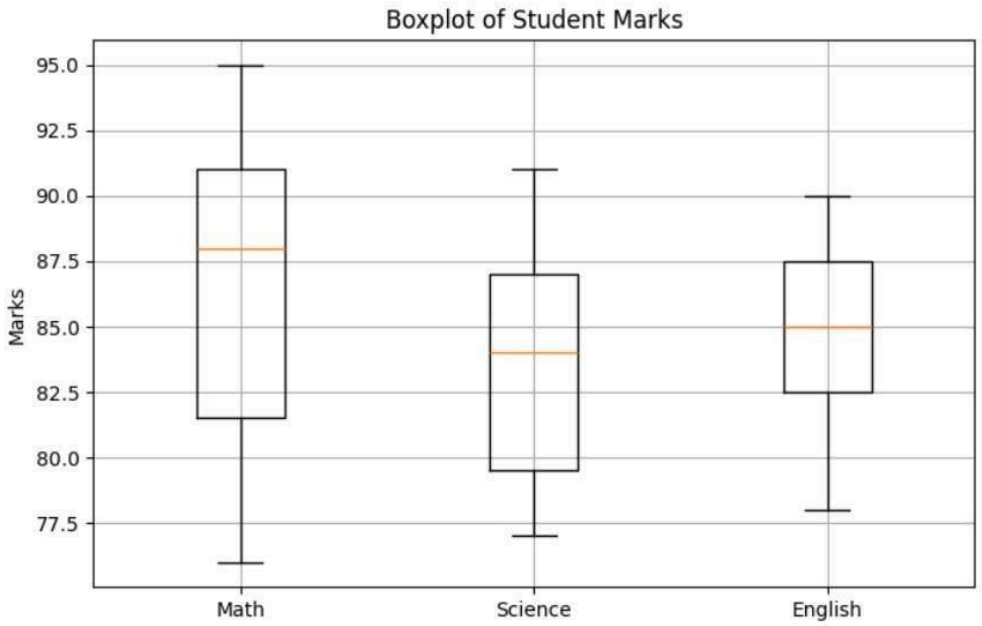
Best Parameters: {'model_C': 1, 'model_solver': 'lbfgs'}
Best CV Score: 0.975
Test Accuracy: 0.978

Program 12: BOX PLOT, Write a Python program using Matplotlib to create a boxplot that shows the distribution of marks for students in three subjects: Math, Science, and English. Add appropriate labels, a title, and display the plot.

Code:

```
import matplotlib.pyplot as plt
# Sample data: Marks of students in three subjects
math = [85, 90, 78, 92, 88, 76, 95]
science = [80, 85, 79, 91, 84, 77, 89]
english = [78, 82, 85, 88, 90, 83, 87]
# Combine all subjects
data = [math, science, english]
# Create boxplot
plt.figure(figsize=(8, 5))
plt.boxplot(data, labels=['Math', 'Science', 'English'])
plt.title("Boxplot of Student Marks")
plt.ylabel("Marks")
plt.grid(True)
plt.show()
```

Output:


Boxplot of Student Marks
Marks
95.0
92.5
90.0
87.5
85.0
82.5
80.0
77.5
Math
Science
English