Experiment No.03 : Program using Numpy Fancy Indexing .

Aim: To demonstrate the use of NumPy fancy indexing by selecting random points from a dataset.

Code:

```
import numpy as np
# Create an array of 2D points (e.g., 10 points in 2D space)
points = np.array([[1, 2],
                   [3, 4],
                   [5, 6],
                   [7, 8],
                   [9, 10],
                   [11, 12],
                   [15, 16],
                   [17, 18],
                   [19, 20]])
# Number of random points to select
num_samples = 4
# Randomly select 4 unique indices from the array
random_indices = np.random.choice(points.shape[0], size=num_samples,
replace=False)
# Use fancy indexing to select the random points
selected_points = points[random_indices]
#output
print("All Points:\n", points)
print("Random Indices Selected:", random_indices)
print("Selected Random Points:\n", selected_points)
```

Output:

```
All Points:
 [[ 1  2]
 [ 3  4]
 [ 5  6]
 [ 7  8]
 [ 9 10]
 [11 12]
 [13 14]
 [15 16]
 [17 18]
 [19 20]]
Random Indices Selected: [2 1 3 8]
Selected Random Points:
 [[ 5  6]
 [ 3  4]
 [ 7  8]
 [17 18]]
```

Experiment No.06: Program using Pandas to Combining Datasets: Join.

Aim: To demonstrate how to combine datasets using join operations in Pandas, including inner join, left join, right join, and outer join, by merging two DataFrames based on a common key.

Code:

```
import pandas as pd

# Create two DataFrames
df1 = pd.DataFrame({
    'ID': [1, 2, 3],
    'Name': ['Alice', 'Bob', 'Charlie']
})

df2 = pd.DataFrame({
    'Age': [25, 30, 22]
}, index=[1, 2, 3])

# Set 'ID' as index for df1 to match df2's index
df1.set_index('ID', inplace=True)

# Join the two DataFrames
result = df1.join(df2)

# Display the result
print("Joined DataFrame:")
print(result)
```

Output:

```
Joined DataFrame:

       Name  Age

ID

1     Alice   25

2       Bob   30

3   Charlie   22
```

Experiment No.07: Program using Pandas on Pivot Tables

Aim: To create and use pivot tables in Pandas for summarizing and analyzing data efficiently.

Code:

```
import pandas as pd

# Sample sales data
data = {
    'Date': ['2024-01-01', '2024-01-01', '2024-01-02', '2024-01-02',
             '2024-01-03', '2024-01-03', '2024-01-04', '2024-01-04'],
    'Product': ['Apple', 'Banana', 'Apple', 'Banana',
                'Apple', 'Banana', 'Apple', 'Banana'],
    'Sales': [100, 150, 120, 130, 140, 160, 110, 180],
    'Region': ['East', 'East', 'West', 'West', 'East', 'East', 'West',
'West']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display original data
print("Original Data:")
print(df)

# Create Pivot Table
pivot = pd.pivot_table(df,
                       index='Date',        # Rows
                       columns='Product',   # Columns
                       values='Sales',      # Values to aggregate
                       aggfunc='sum',       # Aggregation function
                       fill_value=0)        # Fill missing values with 0

# Display Pivot Table
print("\nPivot Table:")
print(pivot)
```

output:

```
Original Data:

         Date Product  Sales Region

0  2024-01-01   Apple    100   East

1  2024-01-01  Banana    150   East

2  2024-01-02   Apple    120   West

3  2024-01-02  Banana    130   West
```

```
4  2024-01-03   Apple    140   East
5  2024-01-03  Banana    160   East
6  2024-01-04   Apple    110   West
7  2024-01-04  Banana    180   West

Pivot Table:
Product     Apple  Banana
Date
2024-01-01    100     150
2024-01-02    120     130
2024-01-03    140     160
2024-01-04    110     180
```

Experiment No.08: Program using Pandas to Vectorized String Operations.

Aim: To perform efficient and fast string manipulations on Pandas Series using vectorized string functions.

Code:

```
import pandas as pd
# Sample data
data = {
    'Name': ['alice smith', 'BOB JOHNSON', 'Charlie Brown', 'david lee'],
    'Email': ['alice@example.com', 'BOB@EXAMPLE.COM', 'charlie@sample.net',
'david123@work.org']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display original data
print("Original Data:")
print(df)

# Apply vectorized string operations
df['Name_TitleCase'] = df['Name'].str.title()        # Capitalize each word
df['Email_Lower'] = df['Email'].str.lower()          # Convert to lowercase
df['Domain'] = df['Email'].str.split('@').str[1]     # Extract domain from
email
df['Name_Length'] = df['Name'].str.len()             # Length of name string
df['Has_Number'] = df['Email'].str.contains(r'\d')   # Check if email has a
digit

# Display modified DataFrame
print("\nAfter String Operations:")
print(df)
```

output:

```
Original Data:

            Name              Email

0    alice smith   alice@example.com

1    BOB JOHNSON     BOB@EXAMPLE.COM

2  Charlie Brown  charlie@sample.net

3      david lee   david123@work.org
```

```
After String Operations:
            Name                Email  ... Name_Length Has_Number
0    alice smith    alice@example.com  ...          11      False
1    BOB JOHNSON      BOB@EXAMPLE.COM  ...          11      False
2  Charlie Brown   charlie@sample.net  ...          13      False
3      david lee    david123@work.org  ...           9       True

[4 rows x 7 columns]
```

Experiment No.09: Program using Pandas to Work with Time Series.

Aim: To work with time series data using Pandas by parsing dates, resampling, and visualizing temporal trends.

Code:

```
import pandas as pd

import matplotlib.pyplot as plt

# Load Seattle bike data (assuming CSV is available)

# Example CSV URL (if needed):
https://data.seattle.gov/Transportation/Fremont-Bridge-Hourly-Bicycle-
Counts-by-Month-o/65db-xm6k

# We'll simulate a local CSV file here

# Make sure your CSV has a Date/Time column

# Load dataset

df = pd.read_csv("FremontBridge.csv", parse_dates=['Date'],
index_col='Date')

# Display first few rows

print("Original Data:")

print(df.head())

# Rename columns for clarity (optional)

df.columns = ['West', 'East']

# Add a total column

df['Total'] = df['West'] + df['East']

# Resample to daily data

daily_counts = df.resample('D').sum()

# Plotting

plt.figure(figsize=(12, 6))

plt.plot(daily_counts.index, daily_counts['Total'], label='Total Bicycles')

plt.title('Daily Bicycle Counts - Seattle (Fremont Bridge)')

plt.xlabel('Date')

plt.ylabel('Bicycle Count')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Output:

```
Traceback (most recent call last):
  File "c:\data practical\vcounts.py", line 10, in <module>
    df = pd.read_csv("FremontBridge.csv", parse_dates=['Date'],
index_col='Date')

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Ankush\AppData\Roaming\Python\Python312\site-
packages\pandas\io\parsers\readers.py", line 1026, in read_csv
    return _read(filepath_or_buffer, kwds)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Ankush\AppData\Roaming\Python\Python312\site-
packages\pandas\io\parsers\readers.py", line 620, in _read
    parser = TextFileReader(filepath_or_buffer, **kwds)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Ankush\AppData\Roaming\Python\Python312\site-
packages\pandas\io\parsers\readers.py", line 1620, in __init__
    self._engine = self._make_engine(f, self.engine)
                   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Ankush\AppData\Roaming\Python\Python312\site-
packages\pandas\io\parsers\readers.py", line 1880, in _make_engine
    self.handles = get_handle(
                   ^^^^^^^^^^^
File "C:\Users\Ankush\AppData\Roaming\Python\Python312\site-
packages\pandas\io\common.py", line 873, in get_handle
    handle = open(
       ^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'FremontBridge.csv'
```

Experiment No.10:Write a NumPy program to swap rows and columns of a given array in reverse order.

Aim: To write a NumPy program that reverses the order of rows and columns in a given array using slicing techniques.

Code:

```
import numpy as np

# Create a sample 2D array
arr = np.array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

print("Original Array:")
print(arr)

# Reverse rows and columns using slicing
reversed_arr = arr[::-1, ::-1]

print("\nArray after Swapping Rows and Columns in Reverse Order:")
print(reversed_arr
```

output:

```
Original Array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Array after Swapping Rows and Columns in Reverse Order:
[[9 8 7]
 [6 5 4]
 [3 2 1]]
```

Experiment No.01: Program on Numpy Aggregations: Min, Max, and etc.

Aim: To understand and implement NumPy aggregation functions such as minimum, maximum, mean, median, and standard deviation for efficient numerical data analysis.

Code:

```
import numpy as np

# Heights of Prime Ministers of India in centimeters (example data)

pm_heights_cm = np.array([170, 165, 168, 172, 174, 167, 169, 173, 175, 171])

# Aggregation operations

min_height = np.min(pm_heights_cm)

max_height = np.max(pm_heights_cm)

mean_height = np.mean(pm_heights_cm)

median_height = np.median(pm_heights_cm)

std_dev = np.std(pm_heights_cm)

# Display results

print("Prime Ministers' Heights (cm):", pm_heights_cm)

print(f"Minimum Height: {min_height} cm")

print(f"Maximum Height: {max_height} cm")

print(f"Average Height: {mean_height:.2f} cm")

print(f"Median Height: {median_height} cm")

print(f"Standard Deviation: {std_dev:.2f} cm")
```

Output:

```
Minimum Height: 165 cm

Maximum Height: 175 cm

Average Height: 170.4 cm

Median Height: 170.5 cm

Standard Deviation: 3.20 cm
```

Experiment No.2: Program using Numpy Comparisons, Masks, and Boolean Logic example: Counting Rainy Days.

Aim: To use NumPy comparison operators, Boolean masks, and logical operations for analyzing data, such as identifying and counting rainy days based on rainfall thresholds.

Code:

```
import numpy as np

# Simulated daily rainfall data for a 30-day month (in millimeters)
rainfall_mm = np.array([0.0, 5.2, 0.0, 0.0, 12.4, 3.3, 0.0,
                        0.0, 0.0, 7.8, 0.0, 14.5, 0.0, 0.0,
                        0.0, 2.1, 0.0, 0.0, 20.3, 0.0, 1.1,
                        0.0, 0.0, 0.0, 5.6, 0.0, 0.0, 0.0,
                        0.0, 9.0])
# Create a boolean mask where rainfall > 0 (i.e., it rained that day)
rainy_days = rainfall_mm > 0
# Count rainy days
num_rainy_days = np.sum(rainy_days)
# Create a mask for heavy rain (e.g., more than 10 mm)
heavy_rain_days = rainfall_mm > 10

# Count heavy rain days
num_heavy_rain_days = np.sum(heavy_rain_days)

# Output results
print("Rainfall Data (mm):", rainfall_mm)
print("Number of Rainy Days (>0 mm):", num_rainy_days)
print("Number of Heavy Rain Days (>10 mm):", num_heavy_rain_days)
```

Output:

```
Rainfall Data (mm): [ 0.   5.2  0.   0.  12.4  3.3  0.   0.   0.   7.8  0.
14.5  0.   0.

  0.   2.1  0.   0.  20.3  0.   1.1  0.   0.   0.   5.6  0.   0.   0.

  0.   9. ]

Number of Rainy Days (>0 mm): 10

Number of Heavy Rain Days (>10 mm): 3
```

Experiment No.04: Write a NumPy program to create a 3x3 identity matrix.

Aim: To learn how to create an identity matrix using NumPy and understand its properties and applications in linear algebra.

Code:

```
import numpy as np

# Create a 3x3 identity matrix
identity_matrix = np.eye(3)

# Output
print("3x3 Identity Matrix:")
print(identity_matrix)
```

Output:

```
3x3 Identity Matrix:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Experiment No.05: Write a NumPy program to create a vector of length 10 with values evenly distributed between 5 and 50.

Aim: To create a one-dimensional NumPy vector of specified length with values evenly spaced between two given numbers using linear spacing.

Code:

```
import numpy as np

# Create a vector of 10 evenly spaced values from 5 to 50
vector = np.linspace(5, 50, 10)

# Output
print("Vector of length 10 evenly spaced between 5 and 50:")
print(vector)
```

Output:

```
Vector of length 10 evenly spaced between 5 and 50:
[ 5. 10. 15. 20. 25. 30. 35. 40. 45. 50.]
```

Experiment No.11: Write a NumPy program to compute the mean, standard deviation, nd variance of a given array along the second axis.

Aim: To compute the mean, standard deviation, and variance of a NumPy array along the second axis (rows), demonstrating statistical analysis across columns in a 2D array.

Code:

```
import numpy as np

# Create a sample 2D array

arr = np.array([[10, 20, 30],

                [40, 50, 60],

                [70, 80, 90]])

# Compute along the second axis (axis=1 = row-wise)

mean_values = np.mean(arr, axis=1)

std_dev_values = np.std(arr, axis=1)

variance_values = np.var(arr, axis=1)

# Display results

print("Original Array:")

print(arr)

print("\nMean along second axis (rows):", mean_values)

print("Standard Deviation along second axis (rows):",
std_dev_values)

print("Variance along second axis (rows):", variance_values)
```

output:

```
Original Array:
[[10 20 30]
 [40 50 60]
 [70 80 90]]

Mean along second axis (rows): [20. 50. 80.]
Standard Deviation along second axis (rows): [8.16496581
8.16496581 8.16496581]
Variance along second axis (rows): [66.66666667 66.66666667
66.66666667]
```

Experiment No.12: Write a NumPy program to sort the student id with increasing height of the students from given students id and height. Print the integer indices that describes the sort order by multiple columns and the sorted data.

Aim: To sort student IDs based on increasing height using NumPy, and to understand how to obtain and apply sort order indices using multiple columns with functions like lexsort.

Code:

```
import numpy as np

# Sample data: student IDs and their heights

student_ids = np.array([1003, 1001, 1004, 1002])

heights = np.array([160, 170, 165, 170])  # in cm


# Use argsort to get the indices that would sort by height
(increasing order)

sort_indices = np.lexsort((student_ids, heights))

# Apply sorting to both arrays

sorted_ids = student_ids[sort_indices]

sorted_heights = heights[sort_indices]

# Print results

print("Original Student IDs:", student_ids)

print("Original Heights:", heights)

print("\nIndices that describe the sort order:", sort_indices)

print("\nSorted Data:")

for sid, h in zip(sorted_ids, sorted_heights):

    print(f"Student ID: {sid}, Height: {h} cm")
```

Output:

```
Original Student IDs: [1003 1001 1004 1002]
Original Heights: [160 170 165 170]

Indices that describe the sort order: [0 2 1 3]

Sorted Data:
Student ID: 1003, Height: 160 cm
Student ID: 1004, Height: 165 cm
Student ID: 1001, Height: 170 cm
Student ID: 1002, Height: 170 cm
```