Name: Husain Hashemi
Roll no. : 20222727
B.voc Software Development – semester 7

Data Analysis & Visualization
Class Assignment

Submitted to: Ms. Sonali Chawla

Ramanujan College (DU) – November 2025

(Use any small dataset — e.g., CSV of students, sales, or weather data.)

1. **Load a CSV file into a pandas DataFrame. Display:**
   - **First 5 rows**
   - **Shape and column names**

Code:

```
import numpy as np
import pandas as  pd
#Q1
df  = pd.read_csv("data.csv")

print("First 5 rows: \n", df.head())
print("Shape of the dataframe: \n", df.shape)
print("Names of columns are: \n", df.columns)
```

Output:

```
First 5 rows:
    No        Name  Type1   Type2  Height  Weight  Legendary
0   1   Bulbasaur  Grass  Poison     0.7     6.9          0
1   2     Ivysaur  Grass  Poison     1.0    13.0          0
2   3    Venusaur  Grass  Poison     2.0   100.0          0
3   4  Charmander   Fire     NaN     0.6     8.5          0
4   5  Charmeleon   Fire     NaN     1.1    19.0          0
Shape of the dataframe:
 (150, 7)
Names of columns are:
 Index(['No', 'Name', 'Type1', 'Type2', 'Height', 'Weight', 'Legendary'], dtype='object')
```

2. **Check for missing values and replace them with the mean (for numeric columns).**

Code:

```
# Q2
print("Missing Values: ")
print(df.isnull().sum())
df.fillna(df.mean(numeric_only=True), inplace=True)
```

Output:

```
Missing Values:
No            0
Name          0
Type1         0
Type2        83
Height        0
Weight        0
Legendary     0
dtype: int64
```

3. **Add a new column named Total which is the sum of two numeric columns (e.g., Maths + Science).**

Code:

```
#Q3
df["Total"] = df["Height"] + df["Weight"]
print(df.head())
```

Output:

```
   No       Name  Type1   Type2  Height  Weight  Legendary  Total
0   1  Bulbasaur  Grass  Poison     0.7     6.9          0    7.6
1   2    Ivysaur  Grass  Poison     1.0    13.0          0   14.0
2   3   Venusaur  Grass  Poison     2.0   100.0          0  102.0
3   4  Charmander   Fire     NaN     0.6     8.5          0    9.1
4   5  Charmeleon   Fire     NaN     1.1    19.0          0   20.1
```

4. **Sort the DataFrame by one of the columns in descending order.**

Code:

```
#Q4
df_sorted = df.sort_values(by="Height", ascending=False)
print(df_sorted)
```

Output:

```
      No       Name     Type1   Type2  Height  Weight  Legendary  Total
94    95       Onix      Rock  Ground     8.8   210.0          0  218.8
129  130   Gyarados     Water  Flying     6.5   235.0          0  241.5
147  148  Dragonair    Dragon     NaN     4.0    16.5          0   20.5
23    24     Arbok    Poison     NaN     3.5    65.0          0   68.5
130  131     Lapras     Water     Ice     2.5   220.0          0  222.5
..   ...       ...       ...     ...     ...     ...        ...    ...
80    81  Magnemite  Electric   Steel     0.3     6.0          0    6.3
132  133     Eevee    Normal     NaN     0.3     6.5          0    6.8
89    90   Shellder     Water     NaN     0.3     4.0          0    4.3
131  132     Ditto    Normal     NaN     0.3     4.0          0    4.3
49    50    Diglett    Ground     NaN     0.2     0.8          0    1.0

[150 rows x 8 columns]
```

5. **Filter the DataFrame to show only rows where a column (e.g., Marks) > 80.**

Code:

```
#Q5
df_filter = df[df["Weight"] > 80]
print(df_filter.head())
```

Output:

	No	Name	Type1	Type2	Height	Weight	Legendary	Total
2	3	Venusaur	Grass	Poison	2.0	100.0	0	102.0
5	6	Charizard	Fire	Flying	1.7	90.5	0	92.2
8	9	Blastoise	Water	NaN	1.6	85.5	0	87.1
58	59	Arcanine	Fire	NaN	1.9	155.0	0	156.9
67	68	Machamp	Fighting	NaN	1.6	130.0	0	131.6

6. **Group the data by a categorical column (e.g., Gender) and compute average of numeric columns.**

Code:

```
#Q6
new_df = df.groupby("Type1")
print(new_df.mean(numeric_only=True))
```

Output:

Type1	No	Height	Weight	Legendary	Total
Bug	42.916667	0.900000	22.991667	0.000000	23.891667
Dragon	148.000000	2.666667	76.600000	0.000000	79.266667
Electric	91.111111	0.855556	31.788889	0.111111	32.644444
Fairy	35.500000	0.950000	23.750000	0.000000	24.700000
Fighting	75.285714	1.185714	54.285714	0.000000	55.471429
Fire	64.166667	1.216667	48.025000	0.083333	49.241667
Ghost	93.000000	1.466667	13.566667	0.000000	15.033333
Grass	55.583333	1.083333	27.991667	0.000000	29.075000
Ground	73.500000	0.850000	45.262500	0.000000	46.112500
Ice	134.000000	1.550000	48.000000	0.500000	49.550000
Normal	71.727273	0.986364	50.086364	0.000000	51.072727
Poison	51.071429	1.221429	27.314286	0.000000	28.535714
Psychic	93.857143	1.371429	58.357143	0.142857	59.728571
Rock	113.333333	1.844444	87.611111	0.000000	89.455556
Water	85.928571	1.300000	57.967857	0.000000	59.267857

7. **Create a NumPy array of shape (4,4) containing random integers from 10 to 99.**

Code:

```
#Q7
array = np.random.randint(10,100, size=(4,4))
print("array: \n", array)
print("first 2 rows of array: \n", array[:2])
print("all elements is 2nd column: \n", array[:, 1:2])
```

Output:

- Display the array

```
array:
 [[34 92 13 19]
 [24 99 47 43]
 [53 30 52 26]
 [70 57 12 57]]
```

- Print the first two rows

```
first 2 rows of array:
 [[34 92 13 19]
 [24 99 47 43]]
```

- Print all elements in the second column

```
all elements is 2nd column:
 [[92]
 [99]
 [30]
 [57]]
```

8. **Create a 1D NumPy array [10, 20, 30, 40, 50].**
   - **Swap the first and last elements using indexing.**

Code:

```
#Q8
d1 = np.array([10,20,30,40,50])
print("before swap: \n", d1)
a = d1[0]
d1[0] = d1[-1]
d1[-1] = a
print("after swap: \n", d1)
```

Output:

```
before swap:
 [10 20 30 40 50]
after swap:
 [50 20 30 40 10]
```

9. **Create a 3×3 identity matrix and transpose it.**
    - **Verify that the transpose of an identity matrix is the same as the original.**

Code:

```
#Q9
arr = np.array([[1,0,0],
                [0,1,0],
                [0,0,1]])
print("before transpose:\n",arr)
print("after transpose:\n",arr.transpose())
```

Output:

```
before transpose:
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
after transpose:
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
```

10. **Create a 1D NumPy array arr = np.arange(1, 11).**
    - **Slice elements from index 3 to 7.**
    - **Replace all even numbers with -1.**

Code:

```
#Q10
arr1 = np.arange(1,11)
print("array created:\n",arr1)
print("sliced 3rd to 7th index:\n",arr1[3:8])
arr1[arr1 %2==0] = -1
print("replaced even no. with -1:\n",arr1)
```

Output:

```
array created:
 [ 1  2  3  4  5  6  7  8  9 10]
sliced 3rd to 7th index:
 [4 5 6 7 8]
replaced even no. with -1:
 [ 1 -1  3 -1  5 -1  7 -1  9 -1]
```

11. Write a NumPy program to compute:

- **Mean, Median, and Standard Deviation of a random array of 10 elements.**

Code:

```
#Q11
ran_arr = np.random.randint(1,100, size=10)
print("array:\n",ran_arr)
print("\nmean: ",np.mean(ran_arr))
print("median: ",np.median(ran_arr))
print("std deviation: ",np.std(ran_arr))
```

Output:

```
array:
 [94 37 66 40 31 83 18 75 92 92]

mean:  62.8
median:  70.5
std deviation:  27.29395537477117
```

12. Consider the dataframe Score given below:

NAme	Class	Score1	Score2	Score3
A	1	85	90	88
B	2	74	86	80
C	1	83	71	92
D	2	64	68	73
E	2	77	62	72
F	1	90	87	92

```
# Q12
data = {"Name":['A', 'B', 'C', 'D', 'E','F'],
        "Class":[1,2,1,2,2,1],
        "Score1": [85,74,83,64,77,90],
        "Score2": [90,86,71,68,62,87],
        "Score3": [88,80,92,73,72,92]}
score = pd.DataFrame(data)
print("Dataframe:\n",score)
print(score[["Name","Class"]])
print(score[score["Class"]==1]["Name"])
print(score[score["Score3"] < 80])
print(score["Class"].value_counts().sort_index())
print(score.sum(axis="columns", numeric_only=True))
```

Give the output of following commands :

(i) Score[['Name','Class']]

Output:

```
  Name  Class
0    A      1
1    B      2
2    C      1
3    D      2
4    E      2
5    F      1
```

(ii) Score[Score['Class'] ==1]['Name']

Output:

```
0    A
2    C
5    F
Name: Name, dtype: object
```

(iii) Score [Score [' Score3'] < 80]

Output:

```
  Name  Class  Score1  Score2  Score3
3    D      2      64      68      73
4    E      2      77      62      72
```

(iv) Score['Class'].value_counts().sort index()

Output:

```
Class
1    3
2    3
Name: count, dtype: int64
```

(v) Score.sum(axis="columns")

Output:

```
0    264
1    242
2    247
3    207
4    213
5    270
dtype: int64
```

(vi) Write a function diff to compute the difference between the maximum and minimum of each column of dataframe Score and apply it to dataframe Score

Code:

```
def min_max(min,max):
    return max - min
a = min_max( score["Score1"].min(), score["Score1"].max() )
print("difference btw max and min of 'Score1' column = ", end="")
print(a)
```

Output:

```
difference btw max and min of 'Score1' column = 26
```

13. Consider the series a given below and write commands to perform the following operations :

a : pd. Series([6, np.nan, -4, np.nan, 3, 8, np.nan, 5])

Code:

```
# Q13
a = pd.Series([6, np.nan, -4, np.nan, 3, 8, np.nan, 5])
print(a.sort_values(na_position="first"))
print(a.rank(ascending=False))
print(a.dropna())
```

i. Sort the values and keep NaN in initial positions.

Output:

```
1    NaN
3    NaN
6    NaN
2   -4.0
4    3.0
7    5.0
0    6.0
5    8.0
```

ii. Assign rank in descending order

```
0    2.0
1    NaN
2    5.0
3    NaN
4    4.0
5    1.0
6    NaN
7    3.0
```

iii. Retrieve all values except NaN.

```
0    6.0
2   -4.0
4    3.0
5    8.0
7    5.0
dtype: float64
```

14. Create a DataFrame having five rows and four columns and populate it with random values in the range I to 100. Set the index of the rows as ['L', 'M', 'N', 'O', 'P'] and column indexes as ['Col1', 'Col2', 'Col3', 'Col4']

Code :

```
# Q14
df = pd.DataFrame(np.random.randint(1,101,size=(5,4)),
                  index=['L','M','N','O','P'],
                  columns=['col1','col2','col3','col4'])
print(df)
```

Output:

```
   col1  col2  col3  col4
L    41    71    36    74
M    40     8    53    41
N    97    98    20    40
O    74    16    57    63
P    97    35    23    36
```

- **Consider the following piece of code and give the output:**

  import pandas as pd
  df1 : pd.DataFrame({'id': [1,3,6,7], 'val' : ['a', 'b','c','d']})
  df2 : pd.DataFrame({'id' : [1,2,3,5,6,8], 'val' :['p','q','r','s','t','u']])
  df3: pd.merge(df1, df2, on : 'id', how = 'outer')
  print(df3)

Output:

```
   id val_x val_y
0   1     a     p
1   2   NaN     q
2   3     b     r
3   5   NaN     s
4   6     c     t
5   7     d   NaN
6   8   NaN     u
```

- **How many NaN values are there in the data frame df3? Write pandas command to replace NaN with the last known valid value in df3.**

Code :

```
print(df3.isnull().sum())
```

Output:

```
id       0
val_x    3
val_y    1
dtype: int64
```

total 3+1=4 NaN

Code: for replacing

```
df3.fillna(method='ffill', inplace=True)
print(df3)
```

Output:

```
   id val_x val_y
0   1     a     p
1   2     a     q
2   3     b     r
3   5     b     s
4   6     c     t
5   7     d     t
6   8     d     u
```

15. Consider numpy array arr given below: (5)

```
arr = [ [0, 1, 2, 3],
        [4, 5, 6, 7],
        [8, 9, 10, 11],
        [12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23] ]
```

Code:

```
# Q15
a = np.array([[0,1,2,3],
              [4,5,6,7],
              [8,9,10,11],
              [12,13,14,15],
              [16,17,18,19],
              [20,21,22,23]])
print("array:\n",a)
```

Output:

```
array:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

Write numpy commands to retrieve following elements:

- **(i) (1,4), (3, 1), (5,0), and (2,3)**

Code:

```
print("element at (1,3):",a[1,3])
print("element at (3,1):",a[3,1])
print("element at (5,0):",a[5,0])
print("element at (2,3):",a[2,3])
```

Output:

```
element at (1,3): 7
element at (3,1): 13
element at (5,0): 20
element at (2,3): 11
```

- **(ii) Retrieve 0,2,4 rows (use positive index)**

Code:

```
print("rows at 0,2,4: \n",a[::2, :])
```

Output:

```
rows at 0,2,4:
 [[ 0  1  2  3]
 [ 8  9 10 11]
 [16 17 18 19]]
```

- **(iii) Retrieve l, 3, 5 rows (use negative index)**

Code:

```
print("rows at 1,3,5:\n", a[1::2, :])
```

Output:

```
rows at 1,3,5:
 [[ 4  5  6  7]
 [12 13 14 15]
 [20 21 22 23]]
```

- **(iv) Retrieve values greater than 10**

Code:

```
print("values greater than 10:\n",a[a > 10])
```

Output:

```
values greater than 10:
 [11 12 13 14 15 16 17 18 19 20 21 22 23]
```

- **(v) Retrieve rows 1 to 4.**

Code:

```
print("rows from 1 to 4 index:\n",a[1:5])
```

Output:

```
rows from 1 to 4 index:
 [[ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]
```