Name: Husain Hashemi
roll no.: 20222727

B.voc. Software Development - semester 7

Data Analysis & Visualization
Practical File (1-8)

Submitted to: Ms. Sonia Sodhi

Ramanujan College (DU) – November 2025

Q1

1. Write programs in Python using NumPy library to do the following:
   a. Compute the mean, standard deviation, and variance of a two dimensional random integer array along the second axis.
   b. Create a 2-dimensional array of size m x n integer elements, also print the shape, type and data type of the array and then reshape it into an n x m array, where n and m are user inputs given at the run time.
   c. Test whether the elements of a given 1D array are zero, non-zero and NaN. Record the indices of these elements in three separate arrays.
   d. Create three random arrays of the same size: Array1, Array2 and Array3. Subtract Array 2 from Array3 and store in Array4. Create another array Array5 having two times the values in Array1. Find Co-variance and Correlation of Array1 with Array4 and Array5 respectively.
   e. Create two random arrays of the same size 10: Array1, and Array2. Find the sum of the first half of both the arrays and product of the second half of both the arrays.

Code a)

```
ria = np.random.randint(0,100, size=(4,5))
# print(ria)
print(np.mean(ria, axis=1))
print(np.var(ria, axis=1))
print(np.std(ria, axis=1))
```

Output

```
[55.6 55.4 60.8 30. ]
[311.44 624.24 438.56 316.4 ]
[17.64766273 24.98479538 20.94182418 17.78763616]
```

b)

```
n = int(input("give num for n cols: "))
m = int(input("give num for m rows: "))
d2 = np.array([[1,2,3,4,5],
               [6,7,8,9,10]])
print(d2.shape)
print(d2.ndim)
print(d2.dtype)
d2 = d2.reshape(m,1,n)
print(d2)
```

output

```
give num for n cols: 2
give num for m rows: 5
(2, 5)
2
int64
[[ 1  2]
 [ 3  4]
 [ 5  6]
 [ 7  8]
 [ 9 10]]
```

c)

```
d1 = np.array([1,2,3,4,5,0,6])
allzero = d1[d1 <= 0]
allnonzero = d1[d1 > 0]
nAn = d1[d1 == np.nan]
print(allzero)
print(allnonzero)
print(nAn)
```

Output

```
[0]
[1 2 3 4 5 6]
[]
```

d)

```
array1 = np.array([[1,2,3],[4,5,6]])
array2 = np.array([[7,8,9],[6,5,4]])
array3 = np.array([[5,6,8],[4,7,9]])

array4 = array2 - array3
array5 = array1 * 2
print(array4)
print(array5)
print(np.cov(array1,array4))
print(np.corrcoef(array1,array4))
print(np.cov(array1,array5))
print(np.corrcoef(array1,array5))
```

Output

```
[[ 2  2  1]
 [ 2 -2 -5]]
[[ 2  4  6]
 [ 8 10 12]]
[[ 1.         1.        -0.5       -3.5      ]
 [ 1.         1.        -0.5       -3.5      ]
 [-0.5       -0.5        0.33333333  1.66666667]
 [-3.5       -3.5        1.66666667 12.33333333]]
[[ 1.         1.        -0.8660254 -0.9966159]
 [ 1.         1.        -0.8660254 -0.9966159]
 [-0.8660254 -0.8660254  1.         0.82199494]
 [-0.9966159 -0.9966159  0.82199494 1.        ]]
[[1. 1. 2. 2.]
 [1. 1. 2. 2.]
 [2. 2. 4. 4.]
 [2. 2. 4. 4.]]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

e)

```
a1 = np.array([1,2,3,4,5,6,7,8,9,10])
a2 = np.array([10,9,8,7,6,5,4,3,2,1])
print(a1[0:5] + a2[0:5])
print(a1[5:] * a2[5:])
```

Output

```
[11 11 11 11 11]
[30 28 24 18 10]
```

Q2

Consider the following data frame containing a family name, gender of the family member and her/his monthly income in each record.

Name	Gender	MonthlyIncome (Rs.)
Shah	Male	114000.00
Vats	Male	65000.00
Vats	Female	43150.00
Kumar	Female	69500.00
Vats	Female	155000.00
Kumar	Male	103000.00
Shah	Male	55000.00
Shah	Female	112400.00
Kumar	Female	81030.00
Vats	Male	71900.00

Write a program in Python using Pandas to perform the following:

a. Calculate and display familywise gross monthly income.
b. Display the highest and lowest monthly income for each family name
c. Calculate and display monthly income of all members earning income less than Rs. 80000.00.
d. Display total number of females along with their average monthly income
e. Delete rows with Monthly income less than the average income of all members

a)

```
import pandas as pd

df = pd.read_csv("dff.csv")
print(df)

group_name = df.groupby("Name")
print(group_name["MonthlyIncome (Rs.)"].sum())
```

Output

```
    Name  Gender  MonthlyIncome (Rs.)
0   Shah    Male               114000
1   Vats    Male                65000
2   Vats  Female                43150
3  Kumar  Female                69500
4   Vats  Female               155000
5  Kumar    Male               103000
6   Shah    Male                55000
7   Shah  Female               112400
8  Kumar  Female                81030
9   Vats    Male                71900
Name
Kumar    253530
Shah     281400
Vats     335050
Name: MonthlyIncome (Rs.), dtype: int64
```

b)

```
print("lowest income is: \n",group_name["MonthlyIncome (Rs.)"].min())
print("highest income is: \n",group_name["MonthlyIncome (Rs.)"].max())
```

Output

```
lowest income is:
 Name
Kumar    69500
Shah     55000
Vats     43150
Name: MonthlyIncome (Rs.), dtype: int64
highest income is:
 Name
Kumar    103000
Shah     114000
Vats     155000
Name: MonthlyIncome (Rs.), dtype: int64
```

c)

```
low_income = df[df["MonthlyIncome (Rs.)"] < 80000]
print(low_income)
```

Output

```
    Name  Gender  MonthlyIncome (Rs.)
1   Vats    Male                65000
2   Vats  Female                43150
3  Kumar  Female                69500
6   Shah    Male                55000
9   Vats    Male                71900
```

d)

```
female = df[df['Gender'] == "Female"]
fe_sum = female["MonthlyIncome (Rs.)"].sum()
print(female)
print("all females tot income is: ",fe_sum)
```

Output

```
    Name  Gender  MonthlyIncome (Rs.)
2   Vats  Female                43150
3  Kumar  Female                69500
4   Vats  Female               155000
7   Shah  Female               112400
8  Kumar  Female                81030
all females tot income is:  461080
```

e)

```
avg_income = df["MonthlyIncome (Rs.)"].mean()
print(avg_income)
df_filter = df[df["MonthlyIncome (Rs.)"] > avg_income]
print(df_filter)
```

Output:

```
86998.0
    Name  Gender  MonthlyIncome (Rs.)
0   Shah    Male               114000
4   Vats  Female               155000
5  Kumar    Male               103000
7   Shah  Female               112400
```

Q3

Use a dataset of your choice from Open Data Portal (https:// data.gov.in/, UCI repository) or load from scikit, seaborn library for the following exercises to practice the concepts learnt.

- Load a Pandas dataframe with a selected dataset. Identify and count the missing values in a dataframe. Clean the data after removing noise as follows
  a) Drop duplicate rows.
  b) Detect the outliers and remove the rows having more than two outliers identified using boxplot.
  c) Identify the most correlated positively correlated attributes and negatively correlated attributes

Code:

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

df = sns.load_dataset("titanic")
print(df.head())
```

I: Identify and count missing values

```
missing_vals = df.isnull().sum()
print(missing_vals)
```

II: Drop duplicate rows

```
dropped_dup = df.drop_duplicates()
print("\nDataFrame shape after dropping duplicates:", dropped_dup.shape)
```

Step 3: Detect outliers using boxplot

```
num_cols = df_clean.select_dtypes(include=[np.number]).columns
print("\nNumerical columns considered for outlier detection:", list(num_cols))

# Detect outliers based on IQR for each numerical column
def detect_outliers_iqr(data, col):
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return (data[col] < lower_bound) | (data[col] > upper_bound)
```

```
# Create a DataFrame to hold outlier flags for each numerical column
outlier_flags = pd.DataFrame()
for col in num_cols:
    outlier_flags[col] = detect_outliers_iqr(df_clean, col)
```

```
# Count outliers per row
outlier_flags['outlier_count'] = outlier_flags.sum(axis=1)
```

```
# Remove rows having more than two outliers
df_no_outliers = df_clean[outlier_flags['outlier_count'] <= 2]
print("\nDataFrame shape after removing rows with more than two outliers:", df_no_outliers.shape)
```

```
# Optional: Plot boxplots before and after cleaning (
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
sns.boxplot(x=df_clean['age'])
plt.title("Age - Before Outlier Removal")

plt.subplot(1, 2, 2)
sns.boxplot(x=df_no_outliers['age'])
plt.title("Age - After Outlier Removal")
plt.show()
```

Step 4: Correlation matrix and find most correlated positive and negative pairs

```
# Step 4: Correlation matrix and find most correlated positive and negative pairs
corr_matrix = df_no_outliers[num_cols].corr()
print("\nCorrelation matrix:\n", corr_matrix)
```

```
# Extract upper triangle of the correlation matrix without diagonal
upper_tri = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
# Find the most positively correlated pair
max_corr = upper_tri.max().max()
max_pair = [(col, row) for col in upper_tri.columns for row in upper_tri.index if upper_tri.loc[row, col] == max_corr]

# Find the most negatively correlated pair
min_corr = upper_tri.min().min()
min_pair = [(col, row) for col in upper_tri.columns for row in upper_tri.index if upper_tri.loc[row, col] == min_corr]
print(f"\nMost positively correlated attributes: {max_pair} with correlation {max_corr:.3f}")
print(f"Most negatively correlated attributes: {min_pair} with correlation {min_corr:.3f}")
```

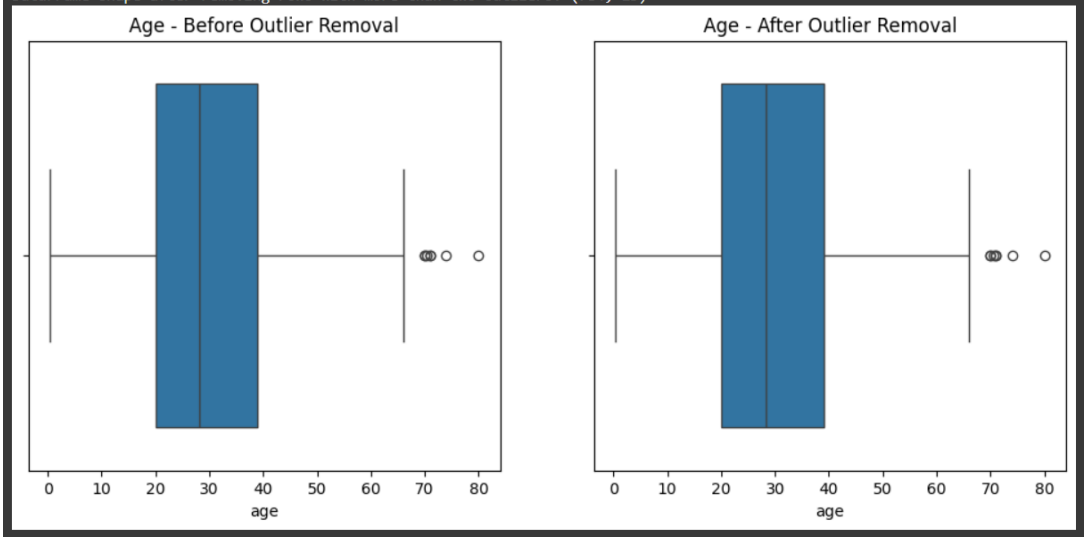
output:

```
sibsp           0
parch           0
fare            0
embarked        2
class           0
who             0
adult_male      0
deck          688
embark_town     2
alive           0
alone           0
dtype: int64

DataFrame shape after dropping duplicates: (784, 15)

Numerical columns considered for outlier detection: ['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']
```

```
DataFrame shape after removing rows with more than two outliers: (784, 15)
```




```
Correlation matrix:
          survived    pclass       age     sibsp     parch      fare
survived  1.000000 -0.332658 -0.086548 -0.036589  0.070307  0.246769
pclass   -0.332658  1.000000 -0.369361  0.088014  0.040296 -0.549216
age      -0.086548 -0.369361  1.000000 -0.315116 -0.195036  0.092707
sibsp    -0.036589  0.088014 -0.315116  1.000000  0.381433  0.135147
parch     0.070307  0.040296 -0.195036  0.381433  1.000000  0.191942
fare      0.246769 -0.549216  0.092707  0.135147  0.191942  1.000000

Most positively correlated attributes: [('parch', 'sibsp')] with correlation 0.381
Most negatively correlated attributes: [('fare', 'pclass')] with correlation -0.549
```

Q4

Import iris data using sklearn library or (Download IRIS data from: https://archive.ics.uci.edu/ml/datasets/iris or import it from sklearn.datasets)

a. Compute mean, mode, median, standard deviation, confidence interval and standard error for each feature

b. Compute correlation coefficients between each pair of features and plot heatmap

c. Find covariance between length of sepal and petal iv. Build contingency table for class feature

Code

```
import pandas as pd
import numpy as np
from sklearn import datasets
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns

iris = datasets.load_iris()
df = pd.DataFrame(data = iris.data, columns=iris.feature_names)
print(df.head())
```

df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)

(a) Descriptive Statistics

```
# Mean
mean_vals = df.iloc[: , :].mean()
print(mean_vals)
```

```
# # Mode
mode_vals = df.iloc[: , :].mean().iloc[0]
print(mode_vals)
```

```
# Median
med_vals = df.iloc[: , :].median()
print(med_vals)
```

```
# Standard variation
std_vals = df.iloc[: , :].std()
print(std_vals)
```

```
# Standard error
st_error = df.iloc[: , :].sem()
print(st_error)
```

```
# 95% Confidence Interval
conf_int = {}
for col in df.columns[:-1]:
    mean = df[col].mean()
    se = df[col].sem()
    ci = stats.t.interval(0.95, len(df[col])-1, loc=mean, scale=se)
    conf_int[col] = ci
print("\n95% Confidence Intervals:")
for col, ci in conf_int.items():
    print(f"{col}: {ci}")
```

(b) Correlation Coefficients + Heatmap

```
# (b) Correlation Coefficients + Heatmap
print("\n(b) Correlation Matrix:")

corr_matrix = df.iloc[: , :].corr()
print(corr_matrix)

# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Heatmap of Feature Correlations")
plt.show()
```

(c) Covariance: Sepal Length vs Petal Length

```
# (c) Covariance: Sepal Length vs Petal Length
print("\n(c) Covariance between Sepal Length and Petal Length:")

cov_matrix = np.cov(df["sepal length (cm)"], df["petal length (cm)"])
cov_val = cov_matrix[0, 1]
print("Covariance:", cov_val)
```

(d) Contingency Table for Class Feature

```
# (d) Contingency Table for Class Feature
print("\n(d) Contingency Table:")

contingency_table = pd.crosstab(index=df['species'], columns="count")
print(contingency_table)
```

Output:

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
PS C:\Users\Husain\Desktop\gg>
```

```
(a) Descriptive Statistics:

Mean:
 sepal length (cm)    5.843333
sepal width (cm)      3.057333
petal length (cm)     3.758000
petal width (cm)      1.199333
dtype: float64
```

```
Mode:
 sepal length (cm)    5.0
sepal width (cm)      3.0
petal length (cm)     1.4
petal width (cm)      0.2
Name: 0, dtype: float64
```

```
Median:
 sepal length (cm)    5.80
sepal width (cm)      3.00
petal length (cm)     4.35
petal width (cm)      1.30
dtype: float64
```

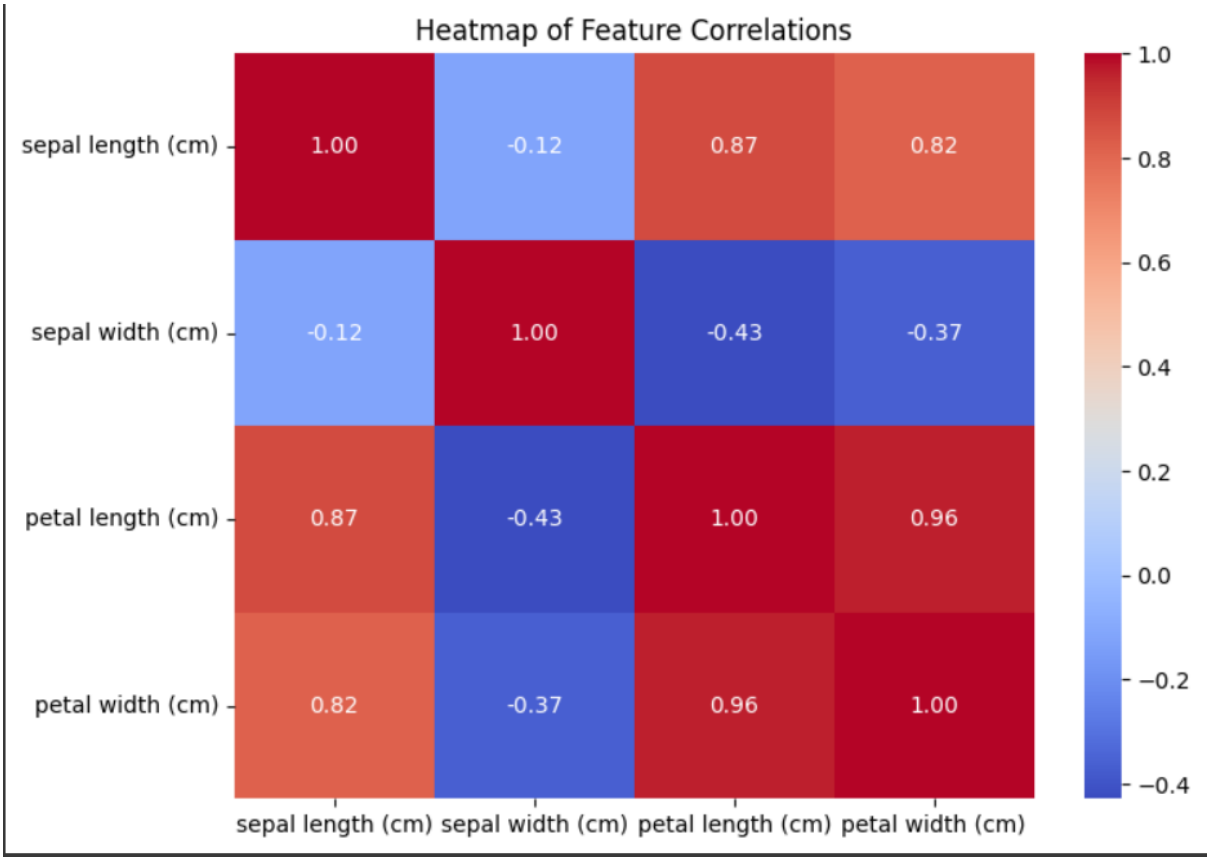
```
Standard Deviation:
 sepal length (cm)    0.828066
sepal width (cm)      0.435866
petal length (cm)     1.765298
petal width (cm)      0.762238
dtype: float64
```

```
Standard Error:
 sepal length (cm)    0.067611
sepal width (cm)      0.035588
petal length (cm)     0.144136
petal width (cm)      0.062236
dtype: float64
```

```
95% Confidence Intervals:
sepal length (cm): (np.float64(5.709732481507366), np.float64(5.976934185159301))
sepal width (cm): (np.float64(2.9870103180785432), np.float64(3.127656348588124))
petal length (cm): (np.float64(3.473185370199511), np.float64(4.04281462980049))
petal width (cm): (np.float64(1.0763532977706853), np.float64(1.3223133688959818))

(b) Correlation Matrix:
                   sepal length (cm)  sepal width (cm)  petal length (cm)  \
sepal length (cm)           1.000000         -0.117570           0.871754
sepal width (cm)           -0.117570          1.000000          -0.428440
petal length (cm)           0.871754         -0.428440           1.000000
petal width (cm)            0.817941         -0.366126           0.962865

                   petal width (cm)
sepal length (cm)          0.817941
sepal width (cm)          -0.366126
petal length (cm)          0.962865
petal width (cm)           1.000000
```




```
(c) Covariance between Sepal Length and Petal Length:
Covariance: 1.2743154362416111

(d) Contingency Table:
col_0       count
species
setosa         50
versicolor     50
virginica      50
```

Q5

Consider two data files (in CSV format) having attendance of two workshops. Each file has three fields 'Name', 'Date, duration (in minutes) where names are unique within a file. Note that duration may take one of three values (30, 40, 50) only. Import the data into two data frames and do the following:

a. Perform merging of the two data frames to find the names of students who had attended both workshops.
b. Find names of all students who have attended a single workshop only.
c. Merge two data frames row-wise and find the total number of records in the data frame.
d. Merge two data frames row-wise and use two columns viz. names and dates as multi-row indexes. Generate descriptive statistics for this hierarchical data frame.

Code:

```
import pandas as pd

df1 = pd.read_csv("workshop1.csv")
df2 = pd.read_csv("workshop2.csv")

print("Workshop 1 Data:\n", df1, "\n")
print("Workshop 2 Data:\n", df2, "\n")
```

(a) Find names of students who attended BOTH workshops

```
# (a) Find names of students who attended BOTH workshops
both = pd.merge(df1, df2, on="Name", suffixes=("_w1", "_w2"))
print("(a) Students who attended both workshops:\n", both["Name"].tolist(), "\n")
```

(b) Find names of students who attended ONLY ONE workshop

```
# (b) Find names of students who attended ONLY ONE workshop
# Use outer merge and find where Name appears only once
merged_outer = pd.merge(df1, df2, on="Name", how="outer", indicator=True)
single_only = merged_outer[merged_outer["_merge"] != "both"]["Name"]
print("(b) Students who attended a single workshop only:\n", single_only.tolist(), "\n")
```

(c) Merge two data frames row-wise (stack them)

```
# (c) Merge two data frames row-wise (stack them)
merged_rowwise = pd.concat([df1, df2], axis=0)
print("(c) Row-wise merged DataFrame:\n", merged_rowwise, "\n")
print("Total number of records:", len(merged_rowwise), "\n")
```

(d) Merge row-wise again, set Name and Date as multi-row index

```
# (d) Merge row-wise again, set Name and Date as multi-row index
#     and generate descriptive statistics
df_hierarchical = merged_rowwise.set_index(["Name", "Date"])
print("(d) Hierarchical DataFrame:\n", df_hierarchical, "\n")

print("Descriptive Statistics for Hierarchical DataFrame:")
print(df_hierarchical.describe())
```

Output:

```
Workshop 1 Data:
        Name        Date  Duration
0     Rudra  08-10-2025        40
1    Husain  08-10-2025        38
2      Amit  08-10-2025        40
3  Mozammil  08-10-2025        30
4  Rudransh  08-10-2025        25

Workshop 2 Data:
      Name        Date  Duration
0  Ankush  09-10-2025        50
1  Pankaj  09-10-2025        45
2   Dhruv  09-10-2025        40
3  Akshat  09-10-2025        30
4   Rudra  09-10-2025        35
```

```
(a) Students who attended both workshops:
 ['Rudra']

(b) Students who attended a single workshop only:
 ['Akshat', 'Amit', 'Ankush', 'Dhruv', 'Husain', 'Mozammil', 'Pankaj', 'Rudransh']

(c) Row-wise merged DataFrame:
        Name        Date  Duration
0     Rudra  08-10-2025        40
1    Husain  08-10-2025        38
2      Amit  08-10-2025        40
3  Mozammil  08-10-2025        30
4  Rudransh  08-10-2025        25
0    Ankush  09-10-2025        50
1    Pankaj  09-10-2025        45
2     Dhruv  09-10-2025        40
3    Akshat  09-10-2025        30
4     Rudra  09-10-2025        35

Total number of records: 10
```

```
(d) Hierarchical DataFrame:
                     Duration
Name     Date
Rudra    08-10-2025        40
Husain   08-10-2025        38
Amit     08-10-2025        40
Mozammil 08-10-2025        30
Rudransh 08-10-2025        25
Ankush   09-10-2025        50
Pankaj   09-10-2025        45
Dhruv    09-10-2025        40
Akshat   09-10-2025        30
Rudra    09-10-2025        35

Descriptive Statistics for Hierarchical DataFrame:
        Duration
count  10.000000
mean   37.300000
std     7.498889
min    25.000000
25%    31.250000
50%    39.000000
75%    40.000000
max    50.000000
```

Q6

Load Titanic data from sklearn library , plot the following with proper legend and axis labels:

a. Plot bar chart to show the frequency of survivors and non-survivors for male and female passengers separately

b. Draw a scatter plot for any two selected features

c. Compare density distribution for features age and passenger fare

d. Use a pair plot to show pairwise bivariate distribution

Code

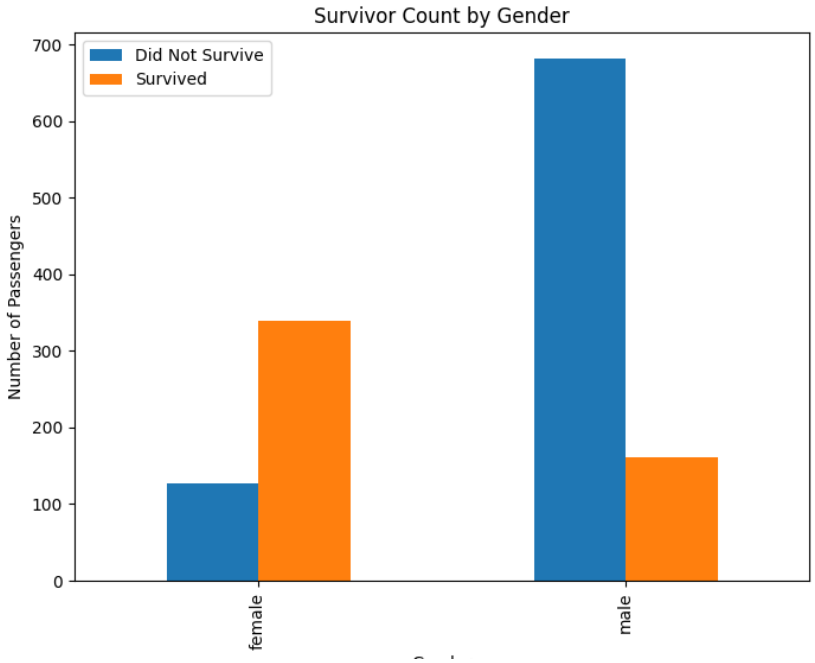
```python
from sklearn.datasets import fetch_openml
import matplotlib.pyplot as plt
import seaborn as sns

titanic = fetch_openml('titanic',  version=1,  as_frame=True)
df = titanic.frame
```

A)

```
# Count survivors by gender
survivor_counts = df.groupby(['sex', 'survived']).size().unstack()
# Plot
survivor_counts.plot(kind='bar', figsize=(8,6))
plt.title('Survivor Count by Gender')
plt.xlabel('Gender')
plt.ylabel('Number of Passengers')
plt.legend(['Did Not Survive', 'Survived'])
plt.show()
```

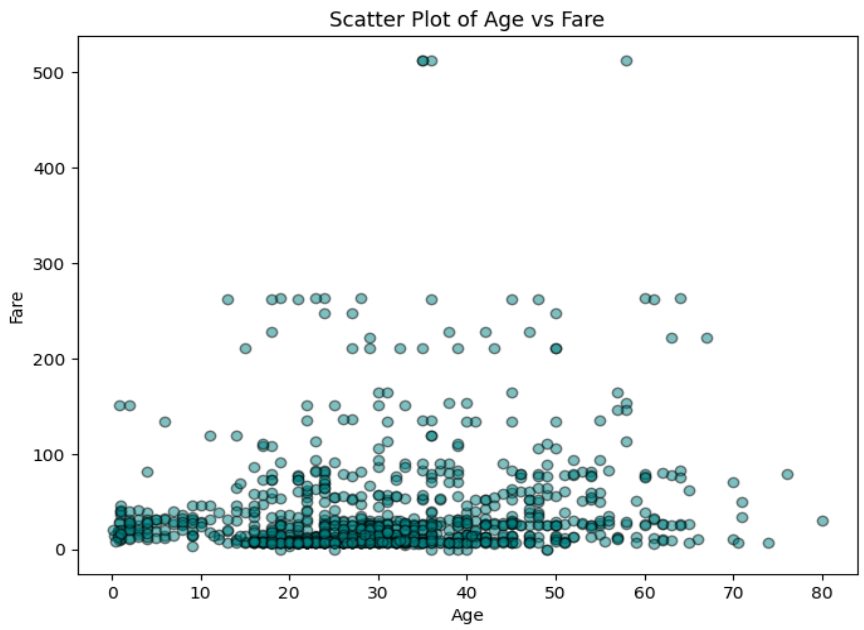
Output:



B)

```
# Drop missing values for scatter plot
df_scatter = df[['age', 'fare']].dropna()
plt.figure(figsize=(8,6))
plt.scatter(df_scatter['age'], df_scatter['fare'], alpha=0.5, c='teal', edgecolor='k')
plt.title('Scatter Plot of Age vs Fare')
plt.xlabel('Age')
plt.ylabel('Fare')
plt.show()
```

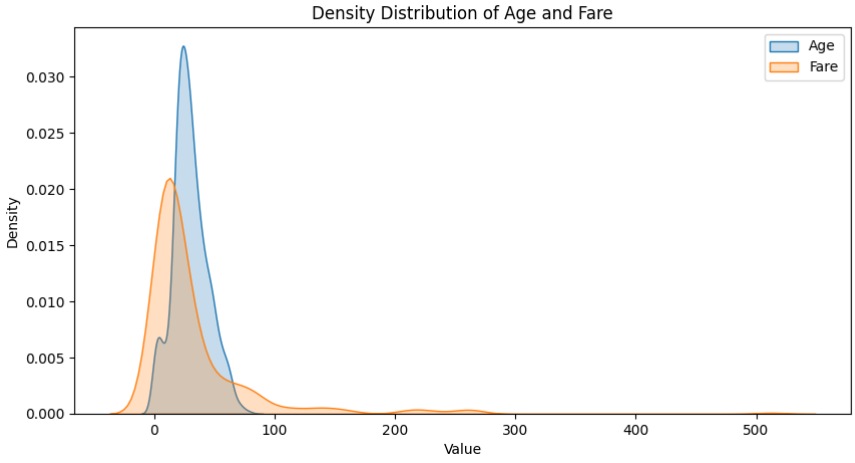
Output:



C)

```
plt.figure(figsize=(10,5))
sns.kdeplot(df['age'].dropna(), label='Age', fill=True)
sns.kdeplot(df['fare'].dropna(), label='Fare', fill=True)
plt.title('Density Distribution of Age and Fare')
plt.xlabel('Value')
plt.ylabel('Density')
plt.legend()
plt.show()
```

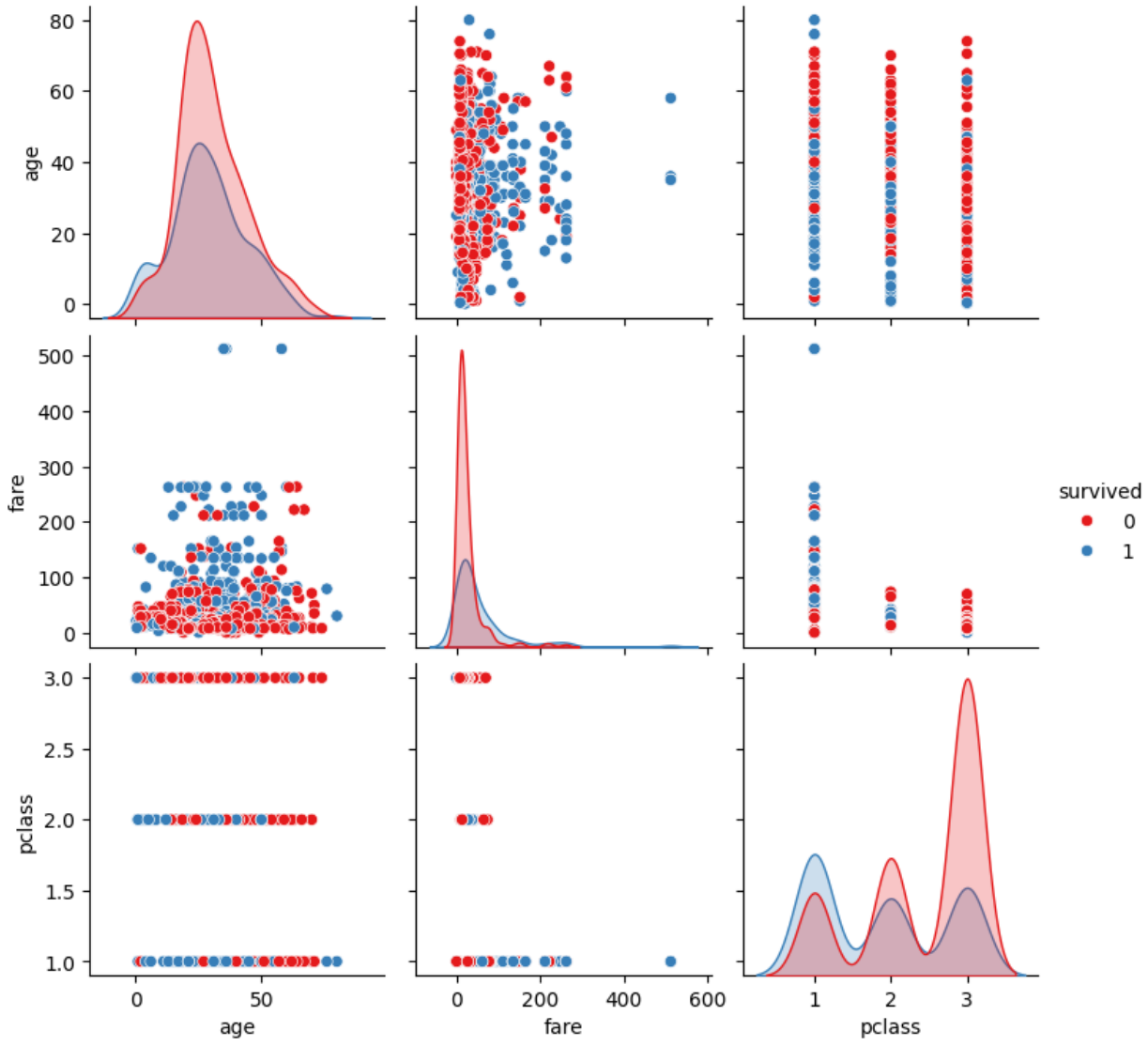
Output:



D)

```
numeric_features = ['age', 'fare', 'pclass', 'survived']
sns.pairplot(df[numeric_features].dropna(), hue='survived', palette='Set1', diag_kind='kde')
plt.suptitle('Pairwise Bivariate Distribution', y=1.02)
plt.show()
```

Output:



Q7

Using Titanic dataset, do the following

a. Find total number of passengers with age less than 30

b. Find total fare paid by passengers of first class

c. Compare number of survivors of each passenger class

Code:

```
from sklearn.datasets import fetch_openml
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

titanic = fetch_openml('titanic',  version=1,  as_frame=True)
df = titanic.frame
```

```
# Convert 'survived' column to numeric
df['survived'] = pd.to_numeric(df['survived'])

# Drop missing ages before filtering
num_passengers_under_30 = df[df['age'].notna() & (df['age'] < 30)].shape[0]
print("Total number of passengers with age less than 30:", num_passengers_under_30)
```

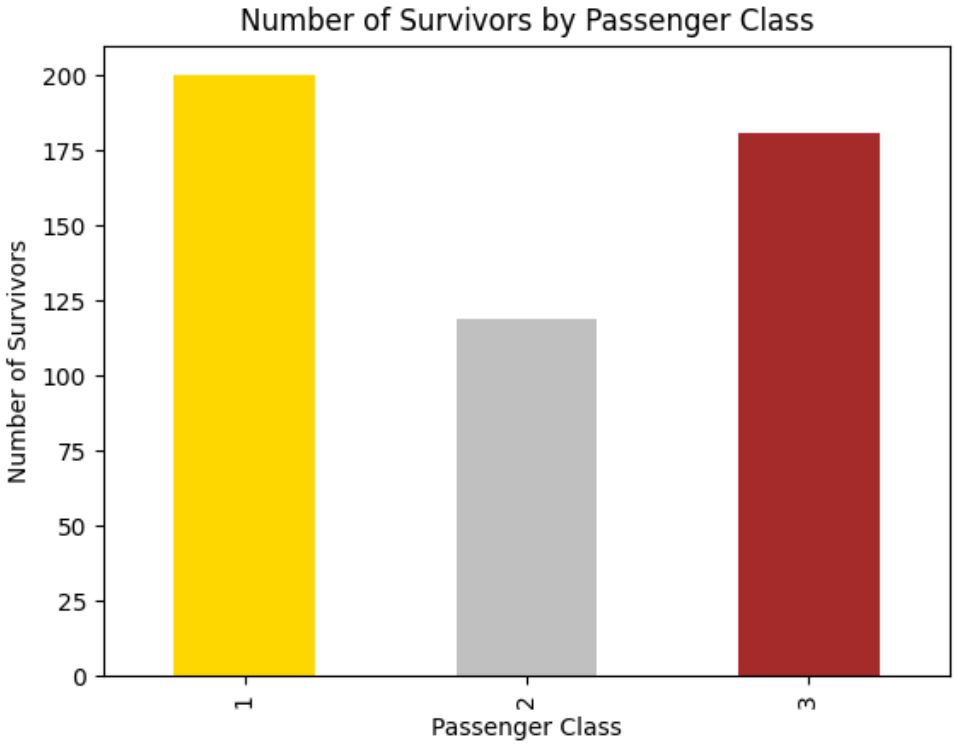
```
total_fare_first_class = df[df['pclass'] == 1]['fare'].sum()
print("Total fare paid by first-class passengers:", total_fare_first_class)

survivors_by_class = df[df['survived'] == 1].groupby('pclass').size()
print("Number of survivors by passenger class:")
print(survivors_by_class)
```

```
survivors_by_class.plot(kind='bar', color=['gold','silver','brown'])
plt.title('Number of Survivors by Passenger Class')
plt.xlabel('Passenger Class')
plt.ylabel('Number of Survivors')
plt.show()
```

Output:

```
Total number of passengers with age less than 30: 569
Total fare paid by first-class passengers: 28265.404300000002
Number of survivors by passenger class:
pclass
1    200
2    119
3    181
dtype: int64
```



Q8

5. Download any dataset and do the following
   a. Count number of categorical and numeric features
   b. Remove one correlated attribute (if any)
   c. Display five-number summary of each attribute and show it visually

Code:

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

# Load dataset
df = sns.load_dataset('penguins')
```

```
# Identify numeric and categorical columns
numeric_features = df.select_dtypes(include=[np.number]).columns
categorical_features = df.select_dtypes(include=['object', 'category']).columns
print("Number of numeric features:", len(numeric_features))
print("Number of categorical features:", len(categorical_features))
print("Numeric features:", list(numeric_features))
print("Categorical features:", list(categorical_features))
```

```
# Correlation matrix
corr_matrix = df[numeric_features].corr()
print("Correlation matrix:\n", corr_matrix)
```

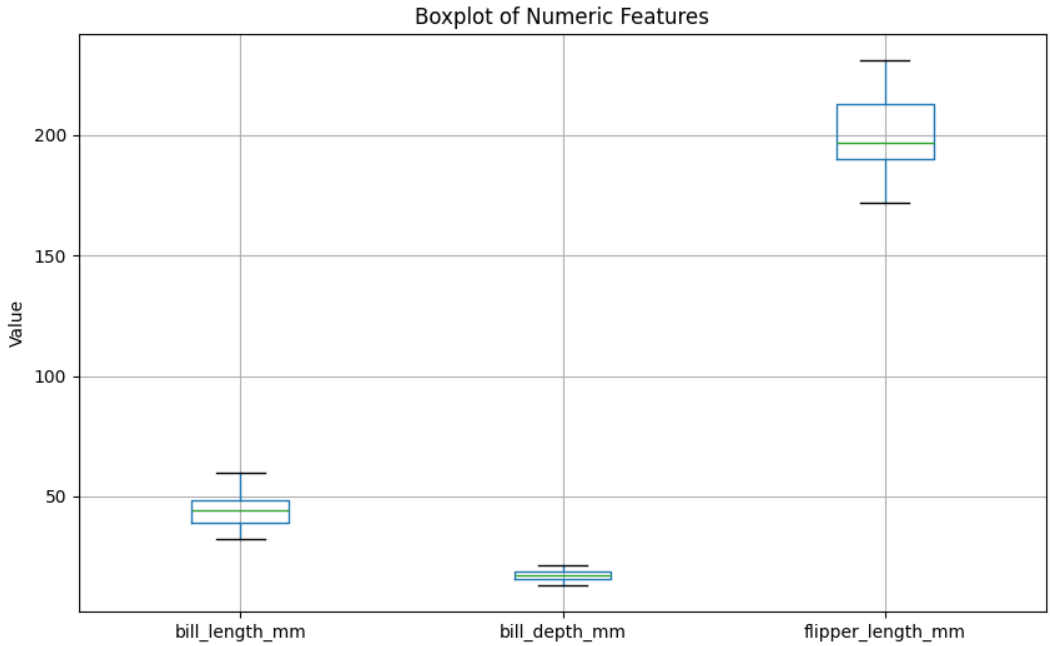
```
# Example: if correlation > 0.8, drop one column
# Here, let's check for correlations > 0.8 (excluding self-correlation)
high_corr = np.where((corr_matrix.abs() > 0.8) & (corr_matrix.abs() < 1))
high_corr_pairs = [(numeric_features[i], numeric_features[j]) for i,j in zip(*high_corr) if i < j]
print("Highly correlated pairs:", high_corr_pairs)
# If any, remove the second column of the first pair
if high_corr_pairs:
    col_to_drop = high_corr_pairs[0][1]
    df = df.drop(columns=[col_to_drop])
    print(f"Dropped column '{col_to_drop}' due to high correlation")
    # Update numeric_features after dropping the column
    numeric_features = df.select_dtypes(include=[np.number]).columns
```

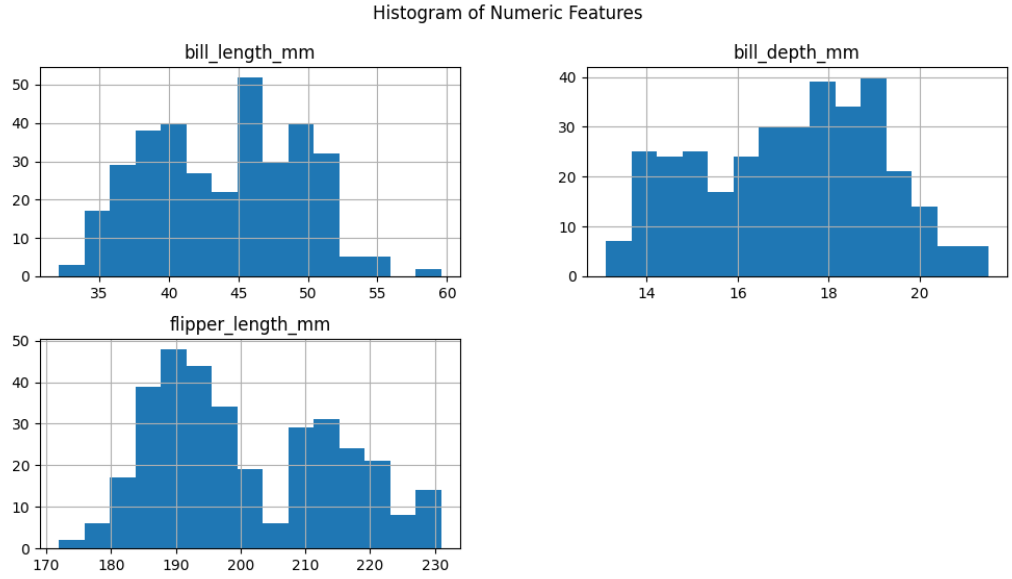
```
# Five-number summary
five_num_summary = df[numeric_features].describe().loc[['min','25%','50%','75%','max']]
print("Five-number summary:\n", five_num_summary)
```

```
# Visualize using boxplots
plt.figure(figsize=(10,6))
df[numeric_features].boxplot()
plt.title("Boxplot of Numeric Features")
plt.ylabel("Value")
plt.show()
df[numeric_features].hist(figsize=(12,6), bins=15)
plt.suptitle("Histogram of Numeric Features")
plt.show()
```

Output:

```
Number of numeric features: 4
Number of categorical features: 3
Numeric features: ['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']
Categorical features: ['species', 'island', 'sex']
Correlation matrix:
                   bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g
bill_length_mm           1.000000      -0.235053           0.656181     0.595110
bill_depth_mm           -0.235053       1.000000          -0.583851    -0.471916
flipper_length_mm        0.656181      -0.583851           1.000000     0.871202
body_mass_g              0.595110      -0.471916           0.871202     1.000000
Highly correlated pairs: [('flipper_length_mm', 'body_mass_g')]
Dropped column 'body_mass_g' due to high correlation
Five-number summary:
      bill_length_mm  bill_depth_mm  flipper_length_mm
min           32.100           13.1              172.0
25%           39.225           15.6              190.0
50%           44.450           17.3              197.0
75%           48.500           18.7              213.0
max           59.600           21.5              231.0
```


Boxplot of Numeric Features
Value
200
150
100
50
bill_length_mm
bill_depth_mm
flipper_length_mm


Histogram of Numeric Features
bill_length_mm
bill_depth_mm
flipper_length_mm