**DATA ANALYSIS AND VISUALISATION FILE**

**SUBMITTED BY**

**- VIVEK SHARMA**

**COURSE – B.SC(H) CS**

**YEAR – 3RD**

**SEMESTER – 5TH**

**ROLL NO. – 21013570104**

**Clg ROLL NO. – 2K21/CS/111**

**SUBMITTED TO**

**–GEETIKA MA’AM**

1.Given below is a dictionary having two keys ‘Boys’ and ‘Girls’ and having two lists of heights of five Boys and Five Girls respectively as values associated with these keys Original dictionary of lists:

{'Boys': [72, 68, 70, 69, 74], 'Girls': [63, 65, 69, 62, 61]}

From the given dictionary of lists create the following list of dictionaries:

[{'Boys': 72, 'Girls': 63}, {'Boys': 68, 'Girls': 65}, {'Boys': 70, 'Girls': 69}, {'Boys': 69, 'Girls': 62}, {‘Boys’:74, ‘Girls’:61]

[ ]

original\_dictionary = {'Boys': [72, 68, 70, 69, 74], 'Girls': [63, 65, 69, 62, 61]}

result\_list = []

boy\_list = original\_dictionary['Boys']

girl\_list = original\_dictionary['Girls']

for i in range(len(boy\_list)):

    boy = boy\_list[i]

    girl = girl\_list[i]

    result\_list.append({'Boys': boy, 'Girls': girl})

print(result\_list)



2.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. Get the indices of the sorted elements of a given array. a. B = [56, 48, 22, 41, 78, 91, 24, 46, 8, 33]

c. 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 nx m array, n and m are user inputs given at the run time.

d. Test whether the elements of a given array are zero, non-zero and NaN. Record the indices of these elements in three separate arrays.

[ ]

#a

import numpy as np

#random 2D array

random\_array = np.random.randint(1, 100, size=(5, 5))

#mean along the second axis (axis=1)

mean\_values = np.mean(random\_array, axis=1)

# standard deviation along the second axis (axis=1)

std\_dev\_values = np.std(random\_array, axis=1)

#variance along the second axis (axis=1)

variance\_values = np.var(random\_array, axis=1)

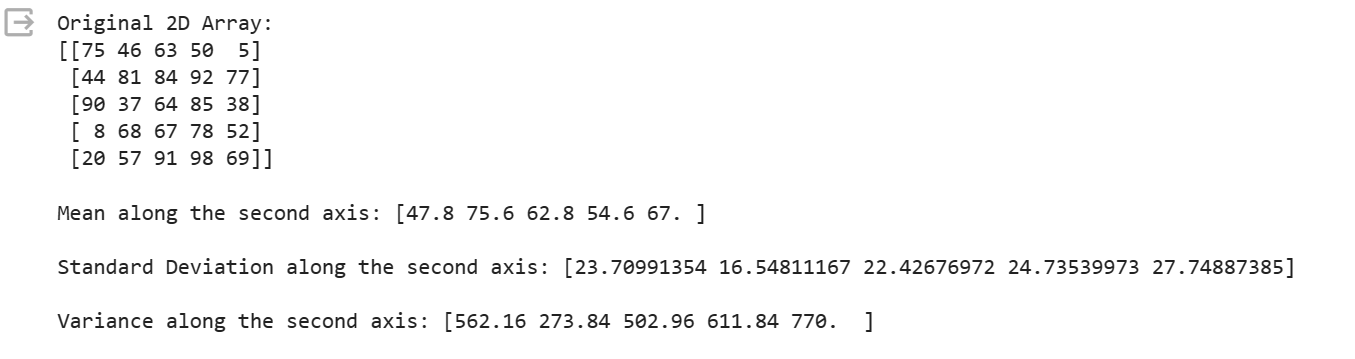
print("Original 2D Array:")

print(random\_array)

print("\nMean along the second axis:", mean\_values)

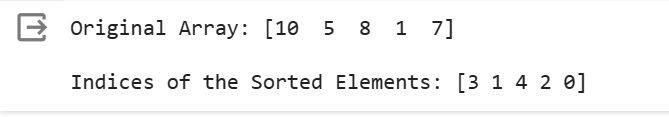
print("\nStandard Deviation along the second axis:", std\_dev\_values)

print("\nVariance along the second axis:",variance\_values)



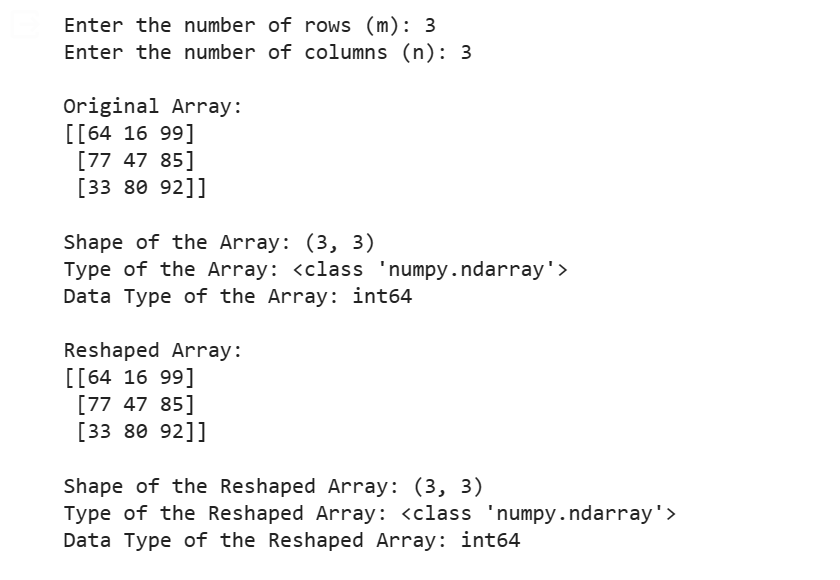
[ ]

#b  
import numpy as np  
  
#random array  
original\_array = np.array([10, 5, 8, 1, 7])  
  
print("Original Array:",original\_array)  
print()  
print("Indices of the Sorted Elements:",np.argsort(original\_array))#indices of the sorted elements



[ ]

#c  
import numpy as np  
  
# user inputs for m and n  
m = int(input("Enter the number of rows (m): "))  
n = int(input("Enter the number of columns (n): "))  
  
#2D array of size m x n with random integer elements  
original\_array = np.random.randint(1, 100, size=(m, n))  
  
print("\nOriginal Array:")  
print(original\_array)  
  
print("\nShape of the Array:", original\_array.shape)  
print("Type of the Array:", type(original\_array))  
print("Data Type of the Array:", original\_array.dtype)  
  
# Reshape the array into an nxm array  
reshaped\_array = original\_array.reshape((n, m))  
  
print("\nReshaped Array:")  
print(reshaped\_array)  
  
print("\nShape of the Reshaped Array:", reshaped\_array.shape)  
print("Type of the Reshaped Array:", type(reshaped\_array))  
print("Data Type of the Reshaped Array:", reshaped\_array.dtype)



[ ]

#d

import numpy as np

#array with some zeros, non-zeros, and NaN values

sample\_array = np.array([1, 0, 5, 0, np.nan, 3, 0])

# Test whether the elements are zero, non-zero, or NaN

zero\_indices = np.where(sample\_array == 0)[0]

non\_zero\_indices = np.where(sample\_array != 0)[0]

nan\_indices = np.where(np.isnan(sample\_array))[0]

# original array

print("Original Array:")

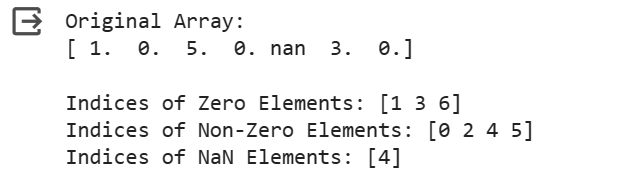
print(sample\_array)

# indices of zero, non-zero, and NaN elements

print("\nIndices of Zero Elements:", zero\_indices)

print("Indices of Non-Zero Elements:", non\_zero\_indices)

print("Indices of NaN Elements:", nan\_indices)



1. Create a dataframe having at least 3 columns and 50 rows to store numeric data generated using a random function. Replace 10% of the values by null values whose index positions are generated using random function. Do the following:

a. Identify and count missing values in a dataframe.

b. Drop the column having more than 5 null values.

c. Identify the row label having maximum of the sum of all values in a row and drop that row.

d. Sort the dataframe on the basis of the first column.

e. Remove all duplicates from the first column.

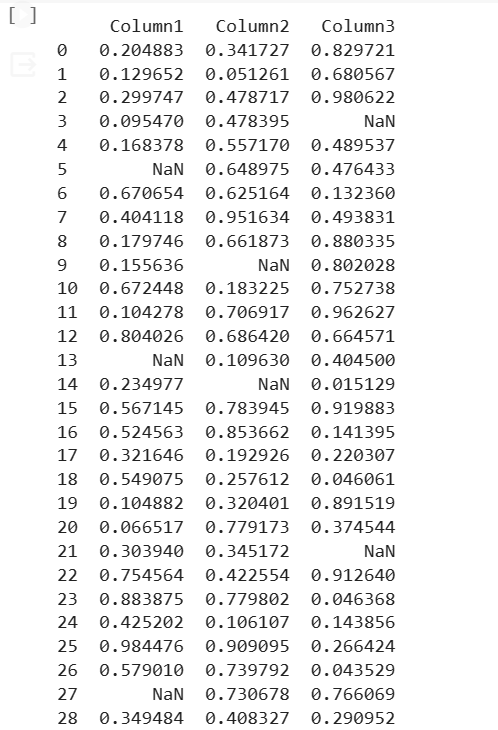
f. Find the correlation between first and second column and covariance between second and third column.

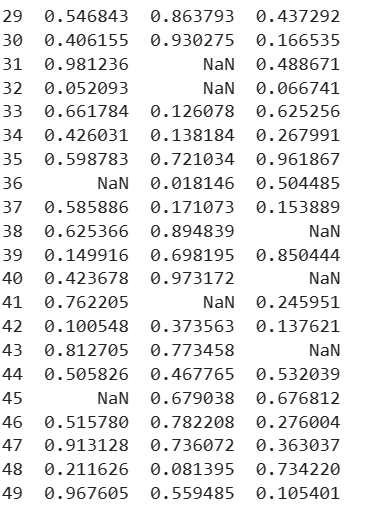
g. Detect the outliers and remove the rows having outliers.

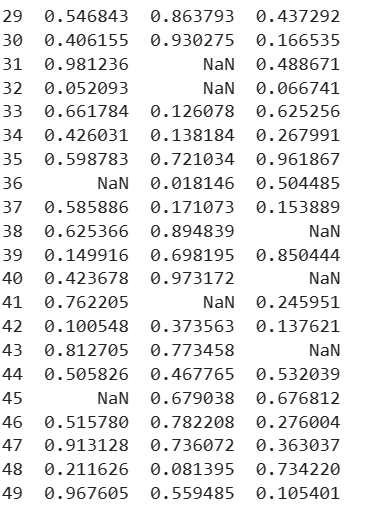
h. Discretize second column and create 5 bins

[ ]

import pandas as pd  
import numpy as np  
  
# Create a dataframe with random numeric data  
data = {'Column1': np.random.rand(50), 'Column2': np.random.rand(50), 'Column3': np.random.rand(50)}  
df = pd.DataFrame(data)  
  
#nan values  
nan\_positions = np.random.choice(df.size, size=int(0.1 \* df.size), replace=False)  
df.values.flat[nan\_positions] = np.nan  
print(df)







[ ]

#a  
missing\_values\_count = df.isnull().sum()  
print("a. Missing Values Count:")  
print(missing\_values\_count)

output

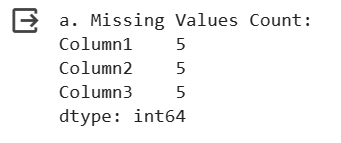
a. Missing Values Count:

Column1 5

Column2 5

Column3 5

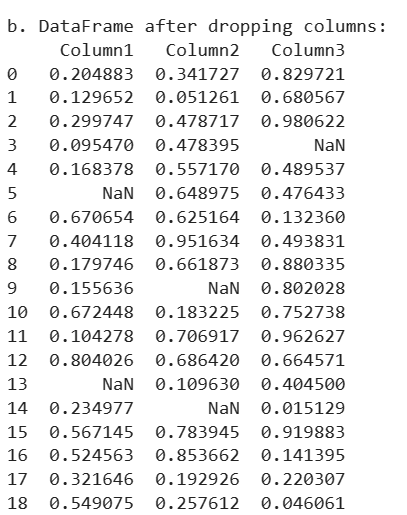
dtype: int64

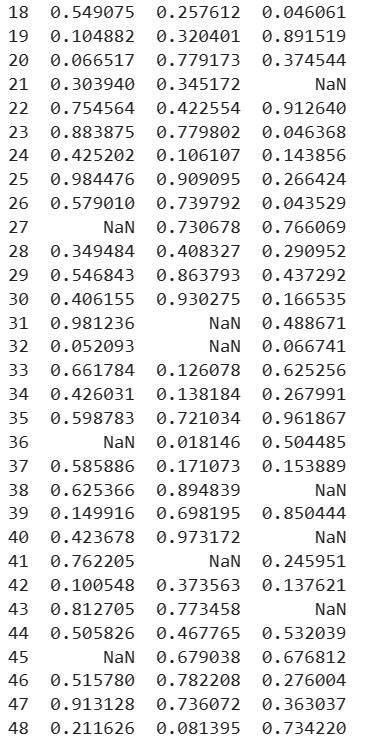


[ ]

#b  
df = df.dropna(axis=1, thresh=df.shape[0] - 5)  
print("\nb. DataFrame after dropping columns:")  
print(df)

output







[ ]

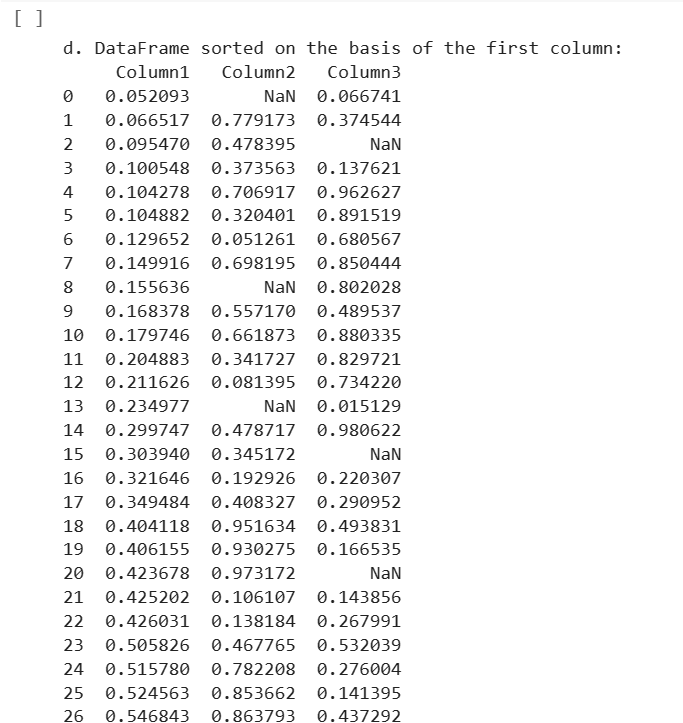
#c  
max\_sum\_row\_label = df.sum(axis=1).idxmax()  
df = df.drop(index=max\_sum\_row\_label)

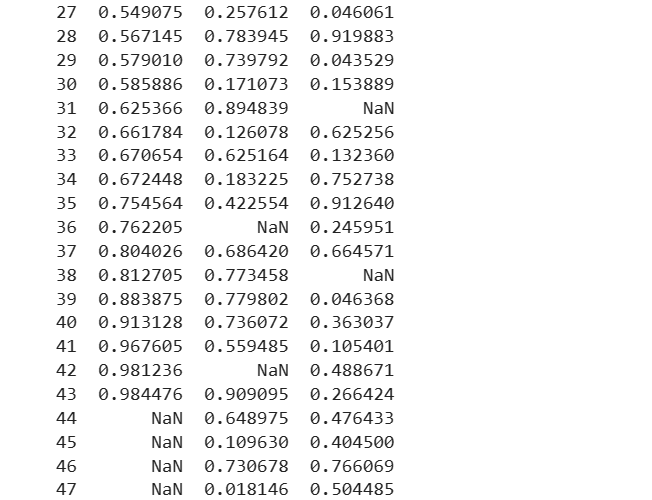
print("Column names:", df.columns)

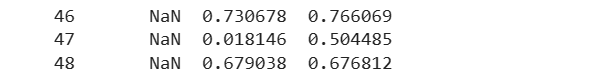


[ ]

#d  
df = df.sort\_values(by=df.columns[0]).reset\_index(drop=True)  
print("\nd. DataFrame sorted on the basis of the first column:")  
print(df)

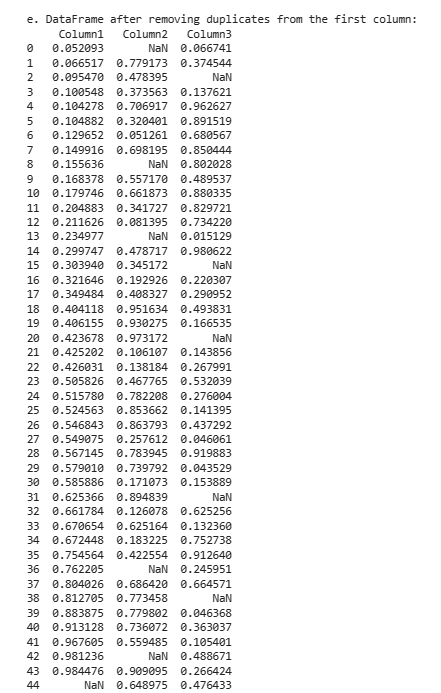






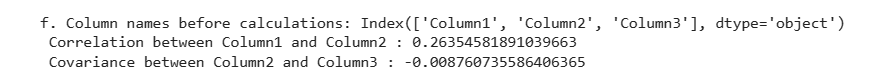
[ ]

#e  
df = df.drop\_duplicates(subset=df.columns[0], ignore\_index=True)  
print("\ne. DataFrame after removing duplicates from the first column:")  
print(df)



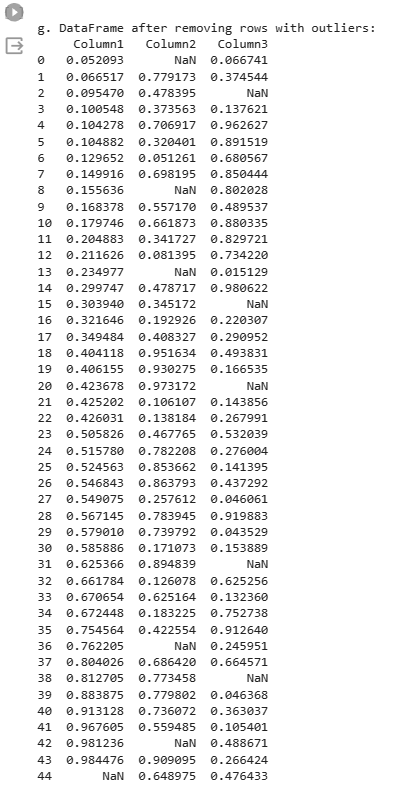
[ ]

print("\nf. Column names before calculations:", df.columns)  
if len(df.columns) >= 3:  
  correlation = df[df.columns[0]].corr(df[df.columns[1]])  
  covariance = df[df.columns[1]].cov(df[df.columns[2]])  
  print(" Correlation between", df.columns[0], "and", df.columns[1], ":", correlation)  
  print(" Covariance between", df.columns[1], "and", df.columns[2], ":", covariance)  
else:  
  print(" Insufficient columns to calculate correlation and covariance.")



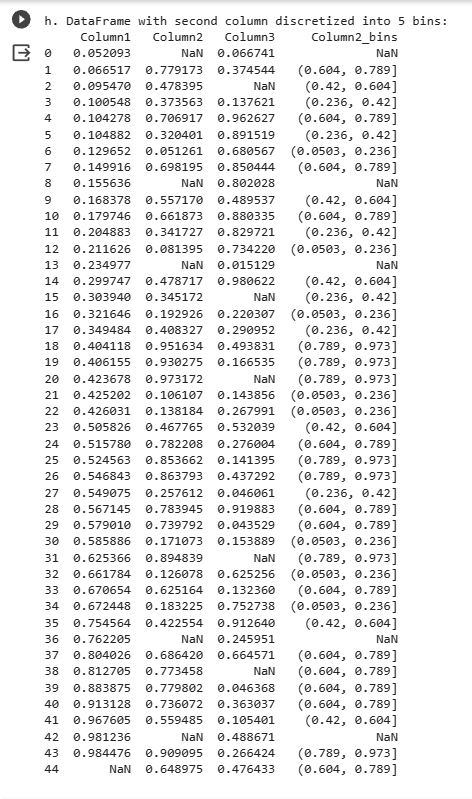
[ ]

#g  
Q1 = df.quantile(0.25)  
Q3 = df.quantile(0.75)  
IQR = Q3 - Q1  
outliers = ((df < (Q1 - 1.5 \* IQR)) | (df > (Q3 + 1.5 \* IQR)))  
df = df[~outliers.any(axis=1)]  
print("\ng. DataFrame after removing rows with outliers:")  
print(df)



[ ]

# h  
df['Column2\_bins'] = pd.cut(df['Column2'], bins=5)  
print("\nh. DataFrame with second column discretized into 5 bins:")  
print(df)



4.Consider two excel files having attendance of a workshop’s participants for two days. Each file has three fields ‘Name’, ‘Time of joining’, 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 dataframes and do the following:

a. Perform merging of the two dataframes to find the names of students who had attended the workshop on both days.

b. Find names of all students who have attended workshop on either of the days.

c. Merge two data frames row-wise and find the total number of records in the data frame.

d. Merge two data frames and use two columns names and duration as multi-row indexes. Generate descriptive statistics for this multi-index.

import pandas as pd

# Load data from Excel files into two dataframes

file1\_path = 'Book1.xlsx'

file2\_path = 'Book2.xlsx'

df\_day1 = pd.read\_excel(file1\_path)

df\_day2 = pd.read\_excel(file2\_path)

# a. Perform merging to find names of students who attended the workshop on both days

common\_names = pd.merge(df\_day1, df\_day2, on='Name', how='inner')['Name']

print("a. Names of students who attended the workshop on both days:")

print(common\_names)

# b. Find names of all students who attended the workshop on either of the days

all\_names = pd.merge(df\_day1, df\_day2, on='Name', how='outer')['Name']

print("\nb. Names of all students who attended the workshop on either of the days:")

print(all\_names)

# c. Merge two data frames row-wise and find the total number of records

merged\_df = pd.concat([df\_day1, df\_day2], ignore\_index=True)

total\_records = len(merged\_df)

print("\nc. Total number of records in the merged data frame:", total\_records)

# d. Merge two data frames and use two columns 'Name' and 'Duration' as multi-row indexes.

# Generate descriptive statistics for this multi-index.

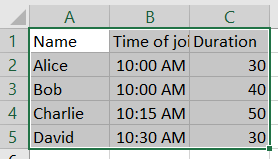
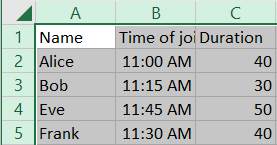
merged\_multiindex\_df = pd.merge(df\_day1, df\_day2, on=['Name', 'Duration'],how='outer')

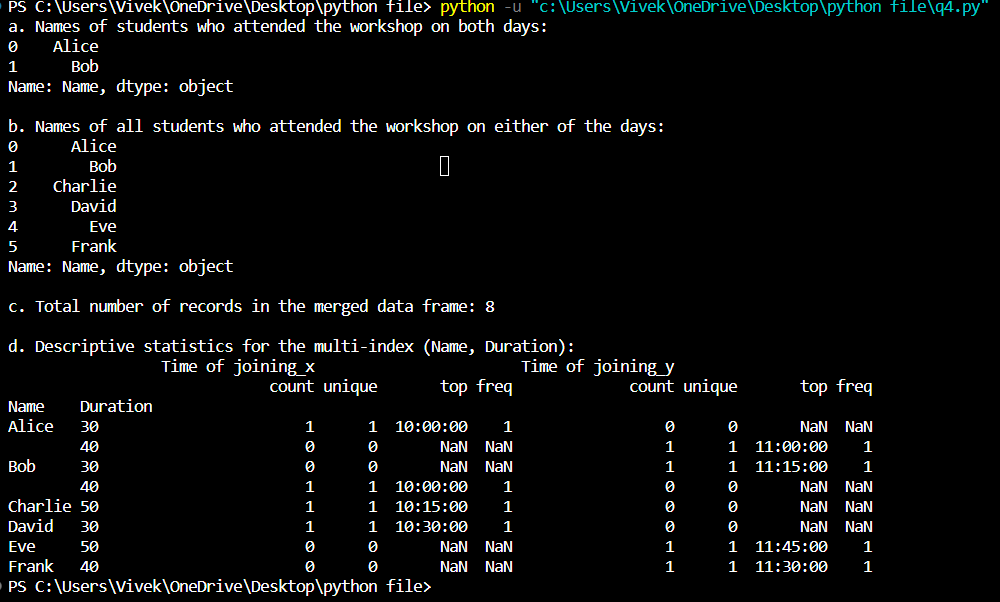
statistics\_multiindex = merged\_multiindex\_df.groupby(['Name', 'Duration']).describe()

print("\nd. Descriptive statistics for the multi-index (Name, Duration):")

print(statistics\_multiindex)

BOOK1 BOOK2



5. Taking Iris data, plot the following with proper legend and axis labels: (Download IRIS data from: [https://archive.ics.uci.edu/ml/datasets/iris](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Farchive.ics.uci.edu%2Fml%2Fdatasets%2Firis) or import it from sklearn.datasets)

a. Plot bar chart to show the frequency of each class label in the data.

b. Draw a scatter plot for Petal width vs sepal width.

c. Plot density distribution for feature petal length.

d. Use a pair plot to show pairwise bivariate distribution in the Iris Dataset

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load the Iris dataset from the file

file\_path = 'bezdekIris.data'  # Assuming both the Python script and the file are in the same folder

column\_names = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)', 'class']

iris\_df = pd.read\_csv(file\_path, header=None, names=column\_names)

# a. Plot bar chart to show the frequency of each class label

class\_counts = iris\_df['class'].value\_counts()

class\_counts.plot(kind='bar', color='skyblue')

plt.title('Class Frequency in Iris Dataset')

plt.xlabel('Class')

plt.ylabel('Frequency')

plt.show()

# b. Draw a scatter plot for Petal width vs Sepal width

plt.figure(figsize=(8, 5))

colors = {'Iris-setosa': 'red', 'Iris-versicolor': 'green', 'Iris-virginica': 'blue'}

for flower\_class, color in colors.items():

    subset = iris\_df[iris\_df['class'] == flower\_class]

    plt.scatter(subset['petal width (cm)'], subset['sepal width (cm)'], label=flower\_class, color=color)

plt.title('Scatter Plot: Petal Width vs Sepal Width')

plt.xlabel('Petal Width (cm)')

plt.ylabel('Sepal Width (cm)')

plt.legend()

plt.show()

# c. Plot density distribution for feature petal length

plt.figure(figsize=(8, 5))

sns.kdeplot(data=iris\_df, x='petal length (cm)', hue='class', fill=True, common\_norm=False, palette='viridis')

plt.title('Density Distribution of Petal Length')

plt.xlabel('Petal Length (cm)')

plt.ylabel('Density')

plt.show()

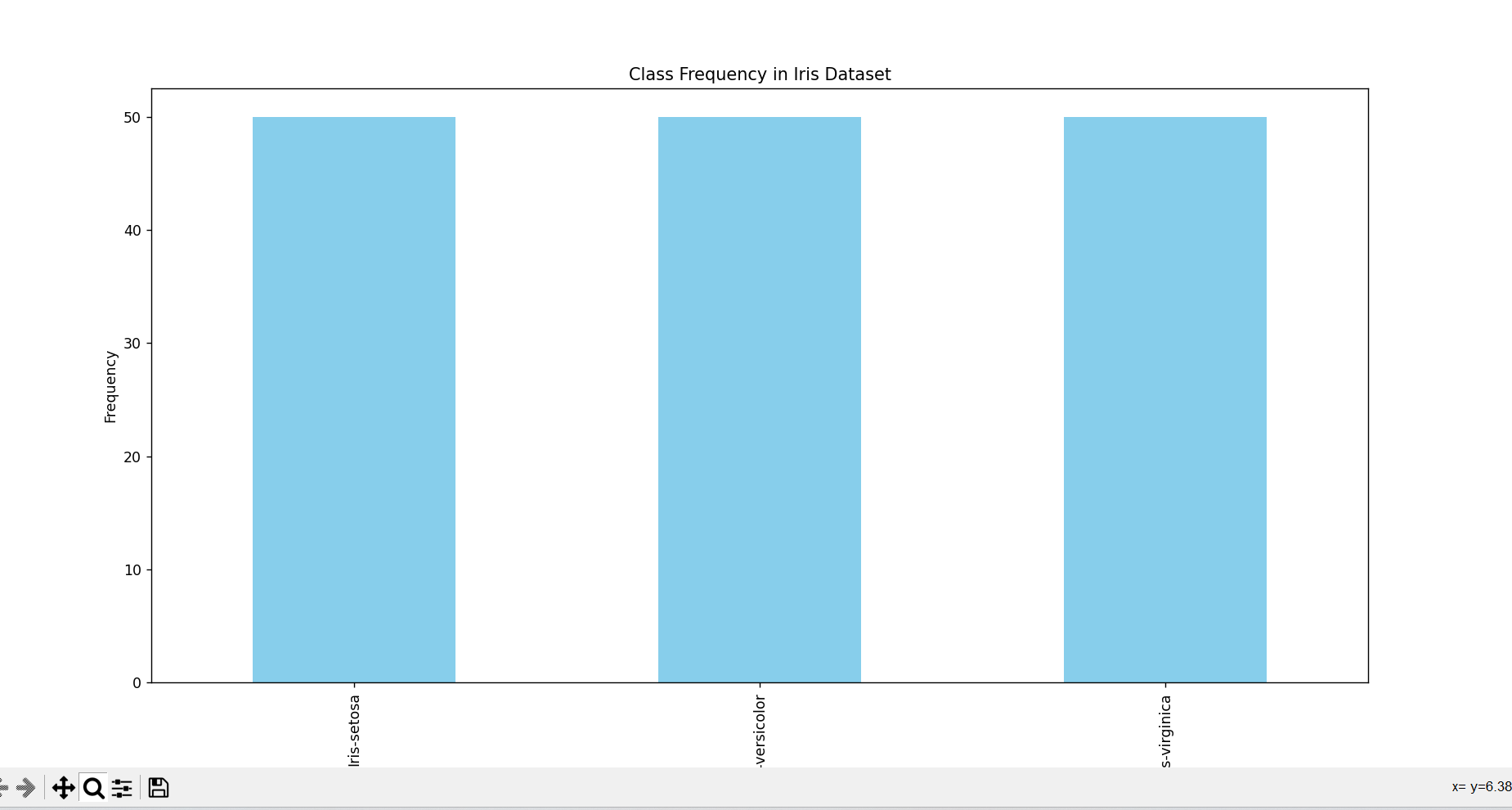
# d. Use a pair plot to show pairwise bivariate distribution in the Iris Dataset

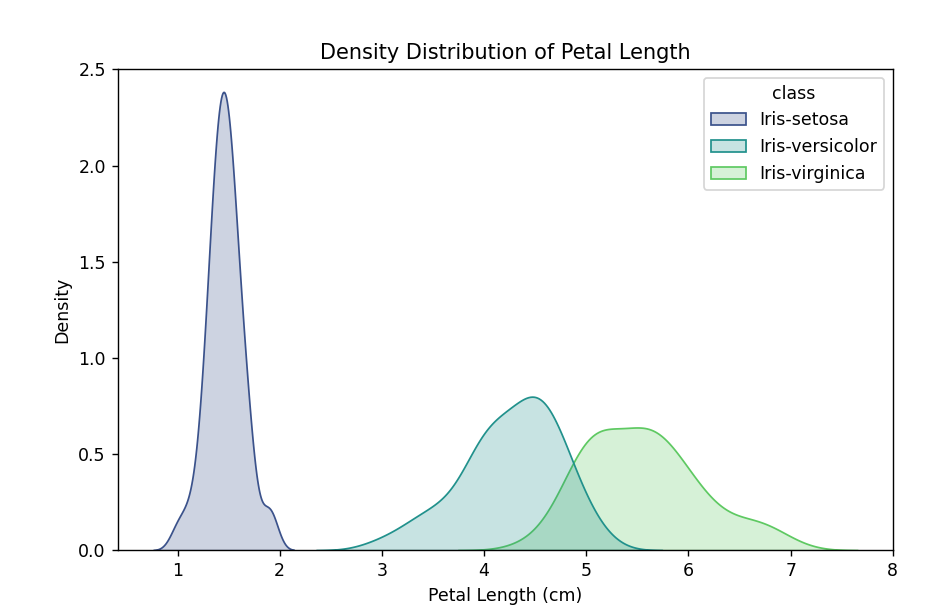
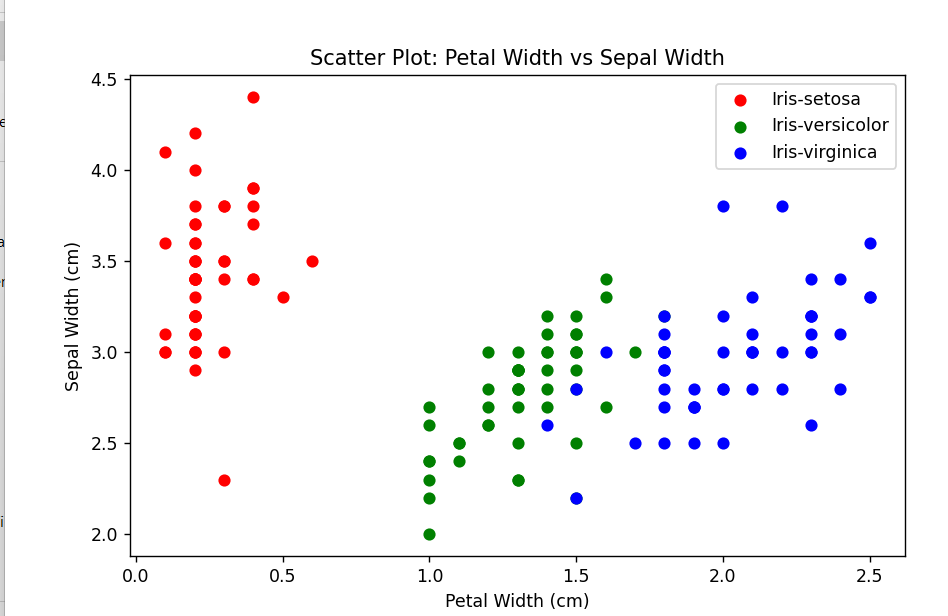
plt.figure(figsize=(10, 8))

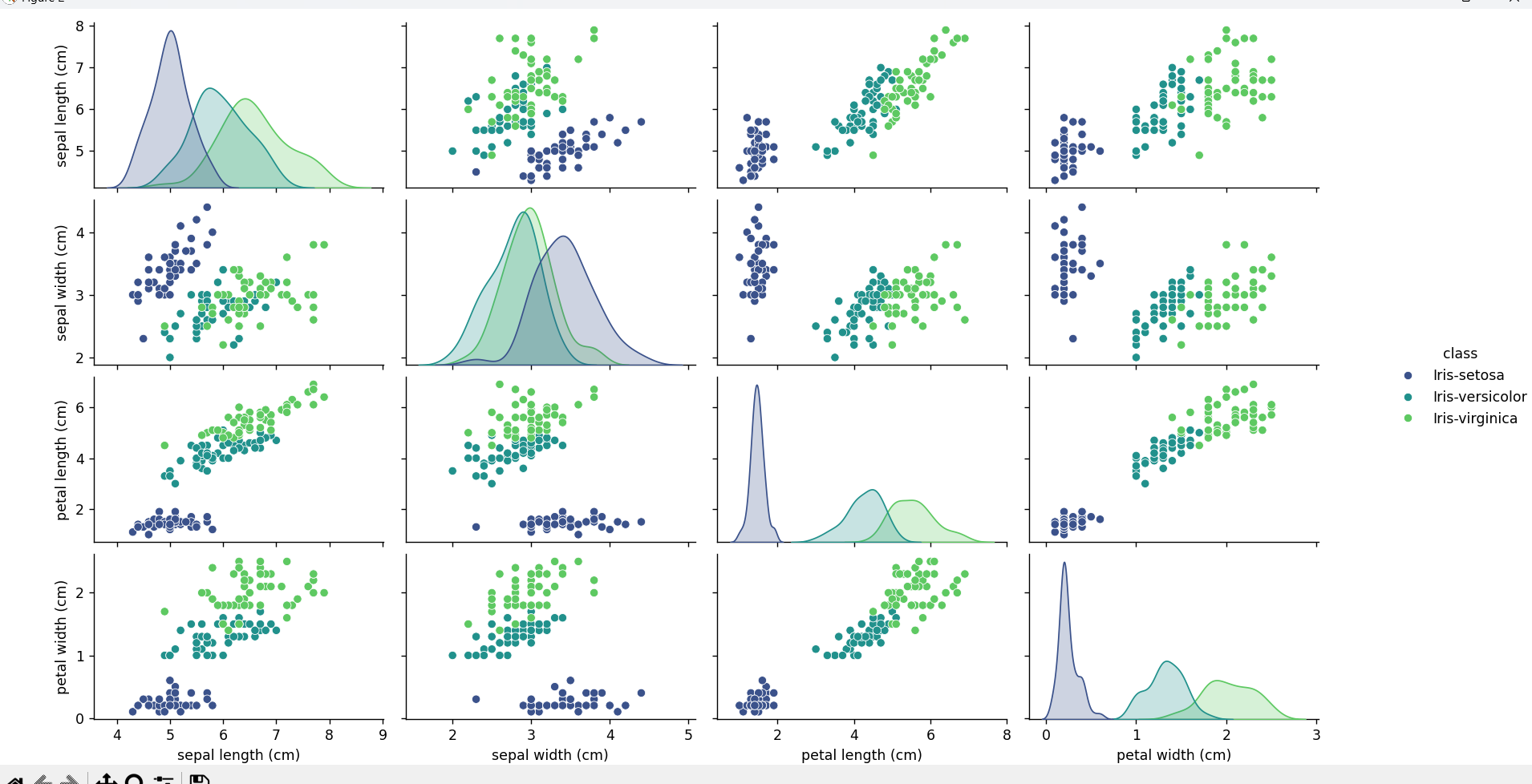
sns.pairplot(iris\_df, hue='class', palette='viridis', height=2.5)

plt.suptitle('Pairwise Bivariate Distribution in Iris Dataset', y=1.02)

plt.show()







6.Consider any sales training/ weather forecasting dataset

a. Compute mean of a series grouped by another series

b. Fill an intermittent time series to replace all missing dates with values of previous non-missing date.

c. Perform appropriate year-month string to dates conversion.

d. Split a dataset to group by two columns and then sort the aggregated results within the groups.

e. Split a given dataframe into groups with bin counts.

[1]

0s

import pandas as pd

import numpy as np

data = {

'Date': pd.date\_range(start='2022-01-01', end='2022-01-10', freq='D').tolist() +

pd.date\_range(start='2022-01-15', end='2022-01-25', freq='D').tolist(),

'Product': ['A'] \* 10 + ['B'] \* 11,

'Sales': [100, 120, 80, 110, 90, np.nan, 130, 150, 140, 120, 200, 180, 160, 190, np.nan, 210, 220, 230, 240, 250, 260]

}

df = pd.DataFrame(data)

# a.

mean\_sales\_by\_product = df.groupby('Product')['Sales'].mean()

print("a. Mean sales by product:")

print(mean\_sales\_by\_product)

# b.

df\_filled = df.set\_index('Date').asfreq('D').ffill()

print("\nb. DataFrame after filling missing dates:")

print(df\_filled)

# c.

df['YearMonth'] = pd.to\_datetime(df['Date']).dt.to\_period('M')

print("\nc. DataFrame with YearMonth column:")

print(df)

# d.

sorted\_sales\_by\_product = df.groupby(['Product', 'YearMonth']).agg({'Sales': 'mean'}).sort\_values(by=['Product', 'YearMonth'])

print("\nd. Sorted sales by product and year-month:")

print(sorted\_sales\_by\_product)

# e.

bin\_counts = 3

df['SalesBin'] = pd.cut(df['Sales'], bins=bin\_counts)

grouped\_by\_bins = df.groupby('SalesBin')

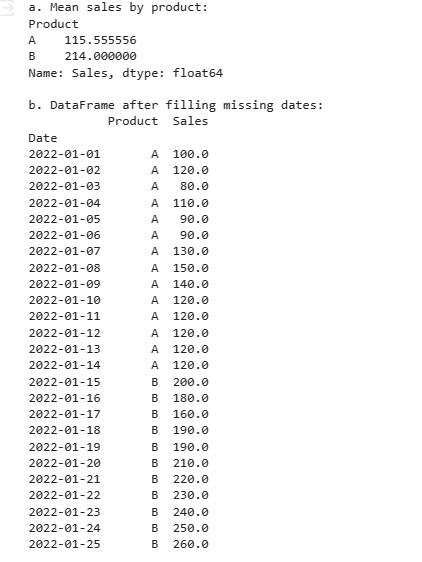
print("\ne. Dataframe split into groups with bin counts:")

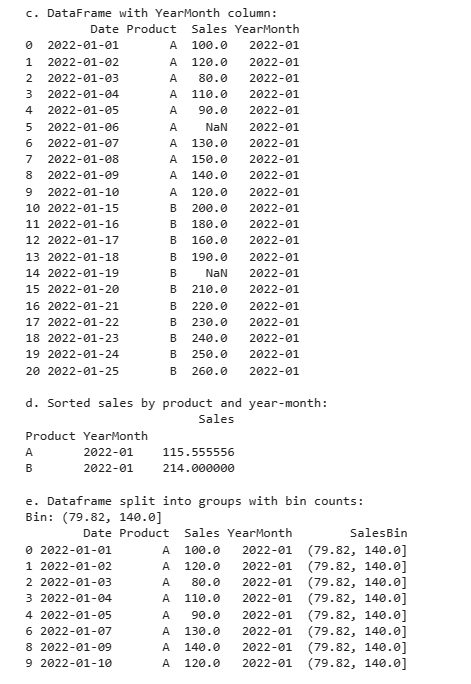
for name, group in grouped\_by\_bins:

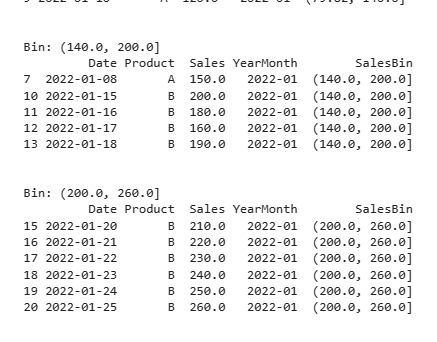
  print(f"Bin: {name}")

  print(group)

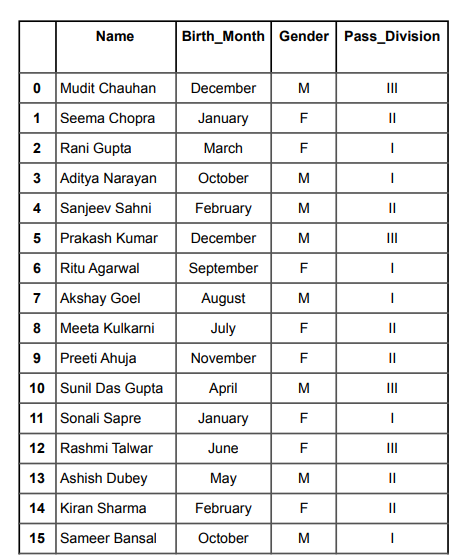
  print("\n")







1. Consider a data frame containing data about students i.e. name, gender and passing division:



a. Perform one hot encoding of the last two columns of categorical data using the get\_dummies() function.

b. Sort this data frame on the “Birth Month” column (i.e. January to December). Hint: Convert Month to Categorical.

[3]

0s

import pandas as pd

# Your provided data

data = {

'Name': ['Mudit Chauhan', 'Seema Chopra', 'Rani Gupta', 'Aditya Narayan', 'Sanjeev Sahni','Prakash Kumar', 'Ritu Agarwal', 'Akshay Goel', 'Meeta Kulkarni', 'Preeti Ahuja','Sunil Das Gupta', 'Sonali Sapre', 'Rashmi Talwar', 'Ashish Dubey', 'Kiran Sharma', 'Sameer Bansal'],

'Birth\_Month': ['December', 'January', 'March', 'October', 'February', 'December','September','August', 'July', 'November', 'April', 'January', 'June', 'May', 'February',

'October'],

'Gender': ['M', 'F', 'F', 'M', 'M', 'M', 'F', 'M', 'F', 'F', 'M', 'F', 'F', 'M', 'F', 'M'],

'Pass\_Division': ['III', 'II', 'I', 'I', 'II', 'III', 'I', 'I', 'II', 'II', 'III', 'I', 'III', 'II', 'II', 'I']

}

df = pd.DataFrame(data)

# a. Perform one hot encoding of the last two columns of categorical data using the get\_dummies() function.

df\_encoded = pd.get\_dummies(df, columns=['Gender', 'Pass\_Division'])

# b. Sort this data frame on the “Birth Month” column.

# Convert 'Birth\_Month' to Categorical with custom order

month\_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August','September', 'October', 'November', 'December']

df\_encoded['Birth\_Month'] = pd.Categorical(df\_encoded['Birth\_Month'],categories=month\_order, ordered=True)

# Sort the DataFrame based on 'Birth\_Month'

df\_sorted = df\_encoded.sort\_values(by='Birth\_Month')

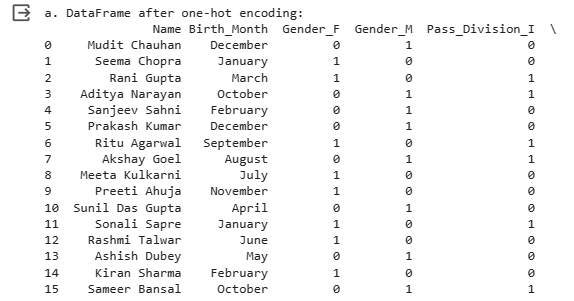
# Display the results

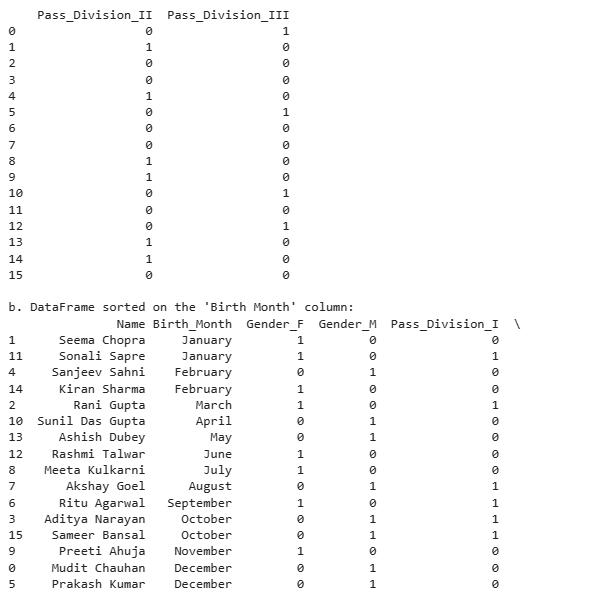
print("a. DataFrame after one-hot encoding:")

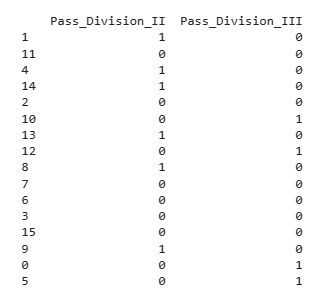
print(df\_encoded)

print("\nb. DataFrame sorted on the 'Birth Month' column:")

print(df\_sorted)







8. 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. Calculate and display the member with the highest monthly income in a family.

c. Calculate and display monthly income of all members with income greater than Rs. 60000.00.

d. Calculate and display the average monthly income of the female members in the Shah family.

[5]

0s

import pandas as pd

# Your provided data

data = {

'Name': ['Shah', 'Vats', 'Vats', 'Kumar', 'Vats', 'Kumar',

'Shah', 'Shah', 'Kumar', 'Vats'],

'Gender': ['Male', 'Male', 'Female', 'Female', 'Female',

'Male', 'Male', 'Female', 'Female', 'Male'],

'MonthlyIncome': [114000.00, 65000.00, 43150.00, 69500.00,155000.00, 103000.00, 55000.00, 112400.00, 81030.00, 71900.00]

}

df = pd.DataFrame(data)

# a. Calculate and display familywise gross monthly income.

familywise\_income = df.groupby('Name')['MonthlyIncome'].sum()

print("a. Familywise Gross Monthly Income:")

print(familywise\_income)

# b. Calculate and display the member with the highest monthly income in a family.

max\_income\_member =df.loc[df.groupby('Name')['MonthlyIncome'].idxmax()]

print("\nb. Member with the Highest Monthly Income in Each Family:")

print(max\_income\_member)

# c. Calculate and display monthly income of all members with income greater than Rs. 60000.00.

high\_income\_members = df[df['MonthlyIncome'] > 60000.00]

print("\nc. Monthly Income of Members with Income Greater Than Rs. 60000.00:")

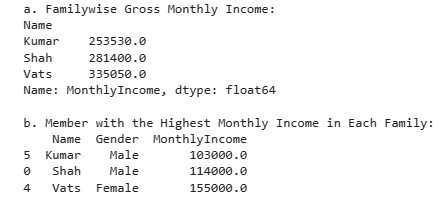
print(high\_income\_members)

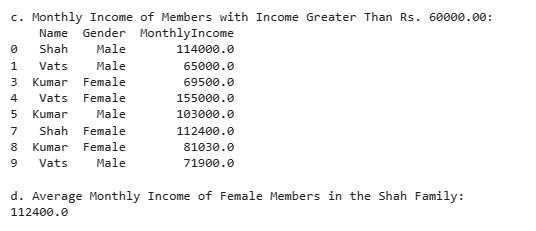
# d. Calculate and display the average monthly income of the female members in the Shah family.

average\_income\_shah\_females = df[(df['Name'] == 'Shah') & (df['Gender'] == 'Female')]['MonthlyIncome'].mean()

print("\nd. Average Monthly Income of Female Members in the Shah Family:")

print(average\_income\_shah\_females)



–––––––––-–