Data Analysis & Visualisation

Practicals

Name :Ujjawal kumar

College roll no :20201441

Exam roll no :20020570038

Ramanujan College

Bsc. (H) Computer Science

5th Semester

Date: 10/11/2022

Submitted to:Mrs. Sheetal Mam

Q. 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]

Code)

In [1]:
```
d1 = {'Boys': [72, 68, 70, 69, 74], 'Girls': [63, 65, 69, 62, 61]}
n = d1['Boys'].__len__()

list_d = [{k: a[i] for k, a in d1.items()}
          for i in range(n)]


print(f'''
\t\t\t Que. 1 Output \n
Original Dictionary   :
{d1}\n
Derived List of dicts :
{list_d}
''')
```

```
                        Que. 1 Output

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

Derived List of dicts :
[{'Boys': 72, 'Girls': 63}, {'Boys': 68, 'Girls': 65}, {'Boys': 70, 'Girls': 69}, {'Boy
s': 69, 'Girls': 62}, {'Boys': 74, 'Girls': 61}]
```

Q. 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.
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 n x 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.

Code)

In [2]:

```
import numpy as np

# a. Computing mean, sd, var along with axis 2
arr = np.random.randint(100, size=(2, 2))
print('\t\t\tQue 2 Output \n')

print(f'''----------------------------------------------------------------------
\nA.\n\nRandom 2D array of {arr.shape} diameter:\n{arr} \n
Array Stats regard axis 2 :
Mean \t\t\t: {arr.mean(1)}
Standard Deviation \t: {np.sqrt(arr.var(1))}
Variance \t\t: {arr.var(1)}
''')

# b. Sorting array's indices
arr = np.array([56, 48, 22, 41, 78, 91, 24, 46, 8, 33])
index = arr.argsort()
print(f'''----------------------------------------------------------------------
\nB.\n\nGiven Numpy Integer Array    : {arr}
Indices of sorted elements   : {index}
Access Array using indices   : {arr[index[0::]]}
''')

# c. Simple Matrix Simulation
print('----------------------------------------------------------------------')
print('\nC.\n\nEnter the parameters of matrix :')
m, n = [int(x) for x in input("rows & columns (use space) : ").split()]
arr = np.random.randint(10*m*n, size=(m, n))
print(f'''
Created Array : \n{arr}\n
Array Details :
    Shape     : {arr.shape}
    Data Type : {arr.dtype}
    Obj Type  : {type(arr)} \n
Reshaped Array into {n} x {m}: \n{arr.reshape(n,m)}
''')

# d. Checking that elements are zero, non-zero or null


def cmp_arr(arr: np.ndarray, cmp):
    return np.array([i for i in range(len(arr)) if cmp(arr[i])])


x = np.array([2, np.NaN, 0, 4, np.NaN, 5, 0, -7, np.NaN])

zero = cmp_arr(x, cmp=lambda a: a == 0)
```

```
nzero = cmp_arr(x, cmp=lambda a: a > 0 or a < 0)
nan = cmp_arr(x, cmp=lambda a: np.isnan(a))

print(f'''------------------------------------------------------------------------
\nD.\n\nGiven Array (x) : {x}
\nIndices of array x that are equal to :
Zero     : {zero}
Non-Zero : {nzero}
NaN      : {nan}
------------------------------------------------------------------------''')
```

Que 2 Output

```
------------------------------------------------------------------------

A.

Random 2D array of (2, 2) diameter:
[[80 22]
 [81  5]]

Array Stats regard axis 2 :
Mean                    : [51. 43.]
Standard Deviation      : [29. 38.]
Variance                : [ 841. 1444.]

------------------------------------------------------------------------

B.

Given Numpy Integer Array    : [56 48 22 41 78 91 24 46  8 33]
Indices of sorted elements   : [8 2 6 9 3 7 1 0 4 5]
Access Array using indices   : [ 8 22 24 33 41 46 48 56 78 91]

------------------------------------------------------------------------

C.

Enter the parameters of matrix :
rows & columns (use space) : 3 4

Created Array :
[[86 44 18 86]
 [66 79 41 15]
 [56 64 83 79]]

Array Details :
    Shape     : (3, 4)
    Data Type : int32
    Obj Type  : <class 'numpy.ndarray'>

Reshaped Array into 4 x 3:
[[86 44 18]
 [86 66 79]
 [41 15 56]
 [64 83 79]]

------------------------------------------------------------------------

D.
```

```
Given Array (x) : [ 2. nan  0.  4. nan  5.  0. -7. nan]

Indices of array x that are equal to :
Zero     : [2 6]
Non-Zero : [0 3 5 7]
NaN      : [1 4 8]
----------------------------------------------------------------
```

Q. 3)

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.

Code)

In [3]:

```
import pandas as pd
from numpy import random, quantile

nrows = 50
# creating a dataframe
df = pd.DataFrame({'Age': random.randint(10, 90, nrows),
                   'Height': random.randint(150, 200, nrows),
                   'Weight': random.randint(50, 200, nrows),
                   })

# replacing 10% random values to null
ncols = len(df.columns)
while df.isnull().sum().sum() != (ncols * nrows // 10):
    df.iloc[random.randint(nrows), random.randint(ncols)] = None

print('''\t\t\tQ.3 Output \n
```

```
----------------------------------------------------------------------
\nGiven DataFrame Details :\n''')
print(df)
df.info()

# A. Identifying & counting null values
var1 = df.isnull().sum()

print(f'''\n----------------------------------------------------------------------
\nA.\n\nTotal null values in Given DataFrame : {sum(var1)}\n
{var1}
''')

# B. Dropping cols with more than 5 null
var1 = [i for i in df if var1[i] > 5]

print(f'''----------------------------------------------------------------------
\nB.\n\nColumns with more than 5 null values : {var1}\n
DataFrame after dropping columns : \n
{df.drop(columns=var1).head()}
''')

# C. Dropping row with max row_sum value
var1 = df.sum(axis=1).idxmax()

print(f'''----------------------------------------------------------------------
\nC.\n\nRow with max sum value : {var1}\n
DataFrame after dropping row {var1} : \n
{df.drop(index=var1)[(var1 - 2) : (var1 + 2)]}
''')

# D. Sorting DataFrame according to 1st col
df = df.sort_values(by=df.columns[0])

print(f'''----------------------------------------------------------------------
\nD.\n\nSorted dataFarme on the basis of first column : \n
{df.head()}\n{df.shape}
''')

# E. Removing duplicates from the 1st col
df.drop_duplicates(df.columns[0], inplace=True)

print(f'''----------------------------------------------------------------------
\nE.\n\nDataFarme after removing duplicates from the first column :
\n {df.head()}\n{df.shape}
''')

# F. Calculating correlation & covariance
var1 = df.columns

print(f'''----------------------------------------------------------------------
\nF.\n\nCorrelation between first & second column : {df[var1[0]].corr(df[var1[1]])}
\nCovariance between second & third column  : {df[var1[1]].cov(df[var1[2]])}
''')

# G. Detect & remove the row having outliers
q = df.quantile(q=[0.25, 0.75])
q.loc['IQR'] = q.iloc[1] - q.iloc[0]
q.loc['low'] = q.iloc[0] - 1.5 * q.iloc[2]
q.loc['high'] = q.iloc[1] + 1.5 * q.iloc[2]
```

```
df = df[~((df < (q.loc['low'])) | (df > (q.loc['high']))).any(axis=1)]

print(f'''------------------------------------------------------------------------
\nG.\n\nInter-Quartile Parameters for the outliers : \n\n{q}
\nDataFrame after removing outliers : \n\n{df.head()}\n{df.shape}
''')

# H. Discretizing second column & creating 5 bins
df['binned'] = pd.qcut(df[var1[1]], q=5)
print(f'''------------------------------------------------------------------------
\nH.\n\nDataFrame after discretizing & creating 5 bins for second column : \n
{df.head()}\n{df.shape}
------------------------------------------------------------------------''')
```

Q.3 Output

```
------------------------------------------------------------------------

Given DataFrame Details :

     Age  Height  Weight
0   46.0   188.0    74.0
1   43.0   176.0    56.0
2   36.0   153.0   108.0
3   74.0   161.0   119.0
4   82.0   170.0   166.0
5   62.0   156.0   187.0
6   55.0   197.0    92.0
7   71.0   173.0   140.0
8   34.0   196.0    92.0
9   62.0   196.0   148.0
10  58.0   154.0    89.0
11   NaN   188.0   146.0
12  68.0   193.0    65.0
13  12.0   186.0    67.0
14  63.0   151.0    90.0
15  38.0   160.0   197.0
16  53.0   190.0   142.0
17  47.0   179.0   111.0
18   NaN   186.0   154.0
19  26.0   170.0    88.0
20  66.0   174.0     NaN
21   NaN     NaN   178.0
22   NaN   150.0   167.0
23  60.0   162.0    52.0
24   NaN     NaN   128.0
25  38.0   170.0   182.0
26  48.0   177.0   102.0
27   NaN   162.0     NaN
28  37.0   161.0    97.0
29  59.0   177.0   118.0
30  25.0   197.0    76.0
31  35.0   193.0    73.0
32  31.0   160.0     NaN
33  47.0     NaN   199.0
34  43.0   171.0   120.0
35  40.0   192.0    70.0
36   NaN   186.0   183.0
37  84.0   178.0   184.0
38   NaN   185.0   139.0
```

```
39  42.0   180.0    63.0
40  27.0   173.0   128.0
41  44.0   175.0    64.0
42  51.0   199.0   192.0
43  33.0   151.0   119.0
44  66.0   151.0   188.0
45  71.0   194.0   114.0
46  27.0   197.0    79.0
47  79.0   161.0   128.0
48  60.0   177.0   108.0
49  34.0     NaN   140.0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Age     42 non-null     float64
 1   Height  46 non-null     float64
 2   Weight  47 non-null     float64
dtypes: float64(3)
memory usage: 1.3 KB

----------------------------------------------------------------------

A.

Total null values in Given DataFrame : 15

Age       8
Height    4
Weight    3
dtype: int64

----------------------------------------------------------------------

B.

Columns with more than 5 null values : ['Age']

DataFrame after dropping columns :

   Height  Weight
0   188.0    74.0
1   176.0    56.0
2   153.0   108.0
3   161.0   119.0
4   170.0   166.0

----------------------------------------------------------------------

C.

Row with max sum value : 37

DataFrame after dropping row 37 :

     Age  Height  Weight
35  40.0   192.0    70.0
36   NaN   186.0   183.0
38   NaN   185.0   139.0
```

```
39  42.0   180.0    63.0

----------------------------------------------------------------------

D.

Sorted dataFarme on the basis of first column :

     Age  Height  Weight
13  12.0   186.0    67.0
30  25.0   197.0    76.0
19  26.0   170.0    88.0
46  27.0   197.0    79.0
40  27.0   173.0   128.0
(50, 3)

----------------------------------------------------------------------

E.

DataFarme after removing duplicates from the first column :

      Age  Height  Weight
13  12.0   186.0    67.0
30  25.0   197.0    76.0
19  26.0   170.0    88.0
46  27.0   197.0    79.0
32  31.0   160.0     NaN
(34, 3)

----------------------------------------------------------------------

F.

Correlation between first & second column : -0.14357036225273387

Covariance between second & third column  : -45.68275862068966

----------------------------------------------------------------------

G.

Inter-Quartile Parameters for the outliers :

        Age  Height   Weight
0.25   36.0  161.00   73.750
0.75   62.0  190.50  140.500
IQR    26.0   29.50   66.750
low    -3.0  116.75  -26.375
high  101.0  234.75  240.625

DataFrame after removing outliers :

     Age  Height  Weight
13  12.0   186.0    67.0
30  25.0   197.0    76.0
19  26.0   170.0    88.0
46  27.0   197.0    79.0
32  31.0   160.0     NaN
(34, 3)
```

```
------------------------------------------------------------------

H.

DataFrame after discretizing & creating 5 bins for second column :

     Age  Height  Weight            binned
13  12.0   186.0    67.0    (179.2, 192.8]
30  25.0   197.0    76.0    (192.8, 199.0]
19  26.0   170.0    88.0    (161.0, 173.4]
46  27.0   197.0    79.0    (192.8, 199.0]
32  31.0   160.0     NaN  (150.999, 161.0]
(34, 4)
------------------------------------------------------------------
```

Q.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.

Code)

In [4]:
```
#first we are importing the essential libraries

import numpy as np
import pandas as pd
```

In [5]:
```
#taking the data from excel files
df1 = pd.read_excel('Day1_ujjawal.xlsx')
df2 = pd.read_excel('Day2_ujjawal.xlsx')
```

In [6]:
```
print(df1,"\n \n \n")
print(df2)
```

```
          Name Time of Joining  Duration
0    Abhimanyu         11:00:00        40
1     Abhishek         11:04:00        30
2        Aasif         11:08:00        30
3         Aman         11:01:00        40
4        Anand         11:12:00        50
5      Anubhav         11:10:00        50
6       Anurag         11:11:00        30
7        Arpit         11:07:00        40
8     Akanksha         11:08:00        50
9      Bhavana         11:15:00        30
10   Deepanshu         11:02:00        40
11      Ishant         11:03:00        30
12      Gourav         11:19:00        30
13     Harshit         11:13:00        40
14    Kartikey         11:05:00        50
```

```
          Name Time of Joining  Duration
0    Abhimanyu         11:00:00        40
1     Abhishek         11:06:00        30
2    Deepanshu         11:10:00        40
3         Aman         11:09:00        40
4      Anubhav         11:10:00        50
5       Bharat         11:12:00        30
6       Anurag         11:08:00        30
7        Arpit         11:08:00        40
8    Divyanshu         11:13:00        40
9      Bhavana         11:14:00        30
10      Deepak         11:02:00        50
11      Ishant         11:00:00        30
12      Jayesh         11:08:00        30
13     Harshit         11:09:00        40
14       Jeeva         11:06:00        30
```

Part A

In [7]:

```
mdf = df1.merge(df2, on = 'Name', suffixes=['_30', '_31'])
mdf
```

Out[7]:

	Name	Time of Joining_30	Duration_30	Time of Joining_31	Duration_31
0	Abhimanyu	11:00:00	40	11:00:00	40
1	Abhishek	11:04:00	30	11:06:00	30
2	Aman	11:01:00	40	11:09:00	40
3	Anubhav	11:10:00	50	11:10:00	50
4	Anurag	11:11:00	30	11:08:00	30
5	Arpit	11:07:00	40	11:08:00	40
6	Bhavana	11:15:00	30	11:14:00	30
7	Deepanshu	11:02:00	40	11:10:00	40

	Name	Time of Joining_30	Duration_30	Time of Joining_31	Duration_31
8	Ishant	11:03:00	30	11:00:00	30
9	Harshit	11:13:00	40	11:09:00	40

Part B

In [8]:
```
#Find names of all students who have attended workshop on either of the days.

either_day = pd.merge(df1,df2,how='outer',on='Name')
either_day
```

Out[8]:

	Name	Time of Joining_x	Duration_x	Time of Joining_y	Duration_y
0	Abhimanyu	11:00:00	40.0	11:00:00	40.0
1	Abhishek	11:04:00	30.0	11:06:00	30.0
2	Aasif	11:08:00	30.0	NaN	NaN
3	Aman	11:01:00	40.0	11:09:00	40.0
4	Anand	11:12:00	50.0	NaN	NaN
5	Anubhav	11:10:00	50.0	11:10:00	50.0
6	Anurag	11:11:00	30.0	11:08:00	30.0
7	Arpit	11:07:00	40.0	11:08:00	40.0
8	Akanksha	11:08:00	50.0	NaN	NaN
9	Bhavana	11:15:00	30.0	11:14:00	30.0
10	Deepanshu	11:02:00	40.0	11:10:00	40.0
11	Ishant	11:03:00	30.0	11:00:00	30.0
12	Gourav	11:19:00	30.0	NaN	NaN
13	Harshit	11:13:00	40.0	11:09:00	40.0
14	Kartikey	11:05:00	50.0	NaN	NaN
15	Bharat	NaN	NaN	11:12:00	30.0
16	Divyanshu	NaN	NaN	11:13:00	40.0
17	Deepak	NaN	NaN	11:02:00	50.0
18	Jayesh	NaN	NaN	11:08:00	30.0
19	Jeeva	NaN	NaN	11:06:00	30.0

PART C

In [9]:
```
#Merge two data frames row-wise and find the total number of records in the data frame.
#using the either day from part b
```

```
either_day['Name'].count()
```

Out[9]: 20

Part D

In [11]:
```
# Merge two data frames and use two columns names and duration as multi-row indexes.
#Generate descriptive statistics for this multi-index

both_days = pd.merge(df1,df2,how='outer',on=['Name','Duration']).copy() #  creates a co

both_days.fillna(value='-',inplace=True) # to fill out the missing values in the given

both_days.set_index(['Name','Duration']) # a method to set a List as index of a Data Fr
```

Out[11]:

Name	Duration	Time of Joining_x	Time of Joining_y
Abhimanyu	**40**	11:00:00	11:00:00
Abhishek	**30**	11:04:00	11:06:00
Aasif	**30**	11:08:00	-
Aman	**40**	11:01:00	11:09:00
Anand	**50**	11:12:00	-
Anubhav	**50**	11:10:00	11:10:00
Anurag	**30**	11:11:00	11:08:00
Arpit	**40**	11:07:00	11:08:00
Akanksha	**50**	11:08:00	-
Bhavana	**30**	11:15:00	11:14:00
Deepanshu	**40**	11:02:00	11:10:00
Ishant	**30**	11:03:00	11:00:00
Gourav	**30**	11:19:00	-
Harshit	**40**	11:13:00	11:09:00
Kartikey	**50**	11:05:00	-
Bharat	**30**	-	11:12:00
Divyanshu	**40**	-	11:13:00
Deepak	**50**	-	11:02:00
Jayesh	**30**	-	11:08:00
Jeeva	**30**	-	11:06:00

Q.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 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.

Code)

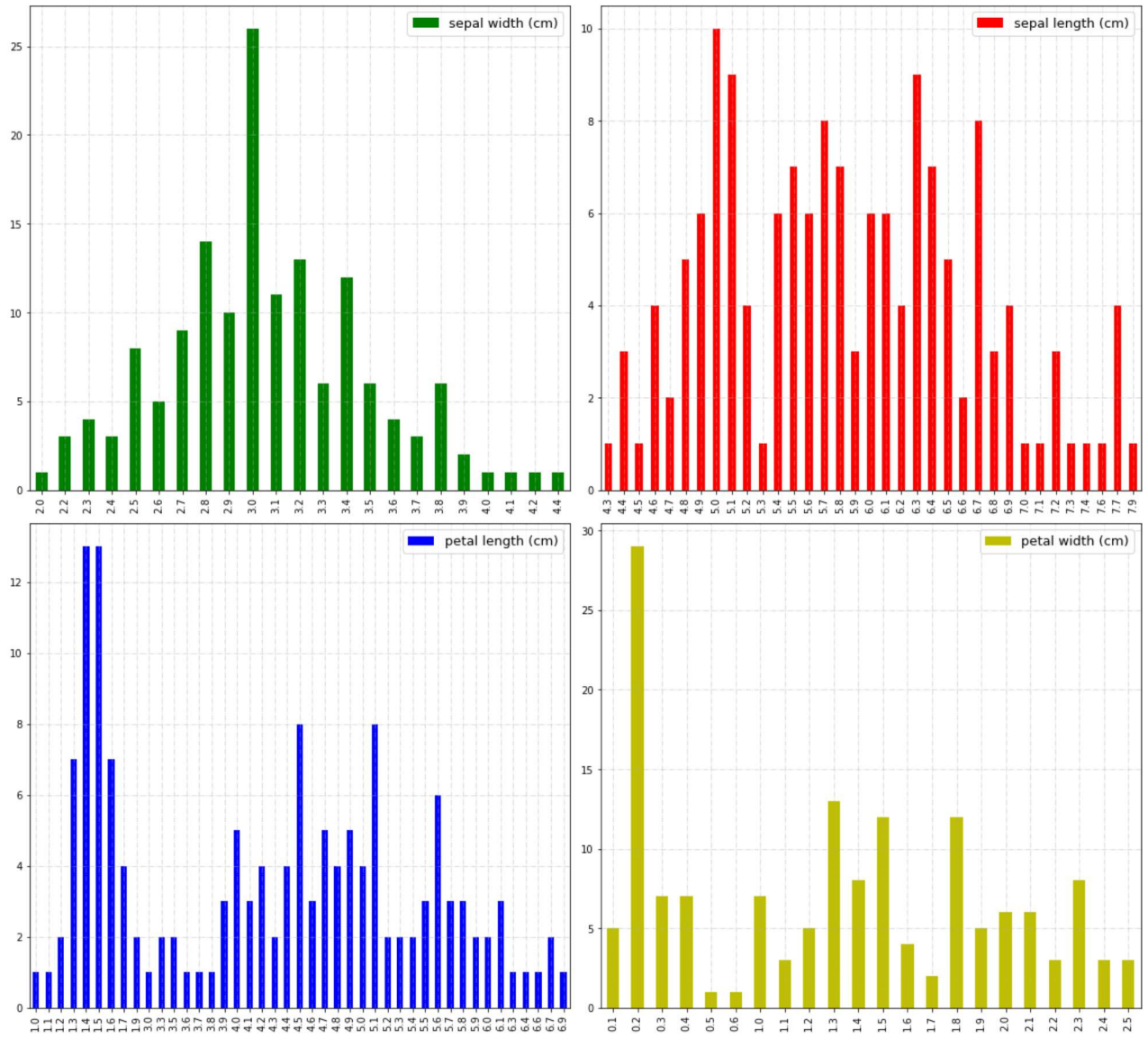
In [12]:

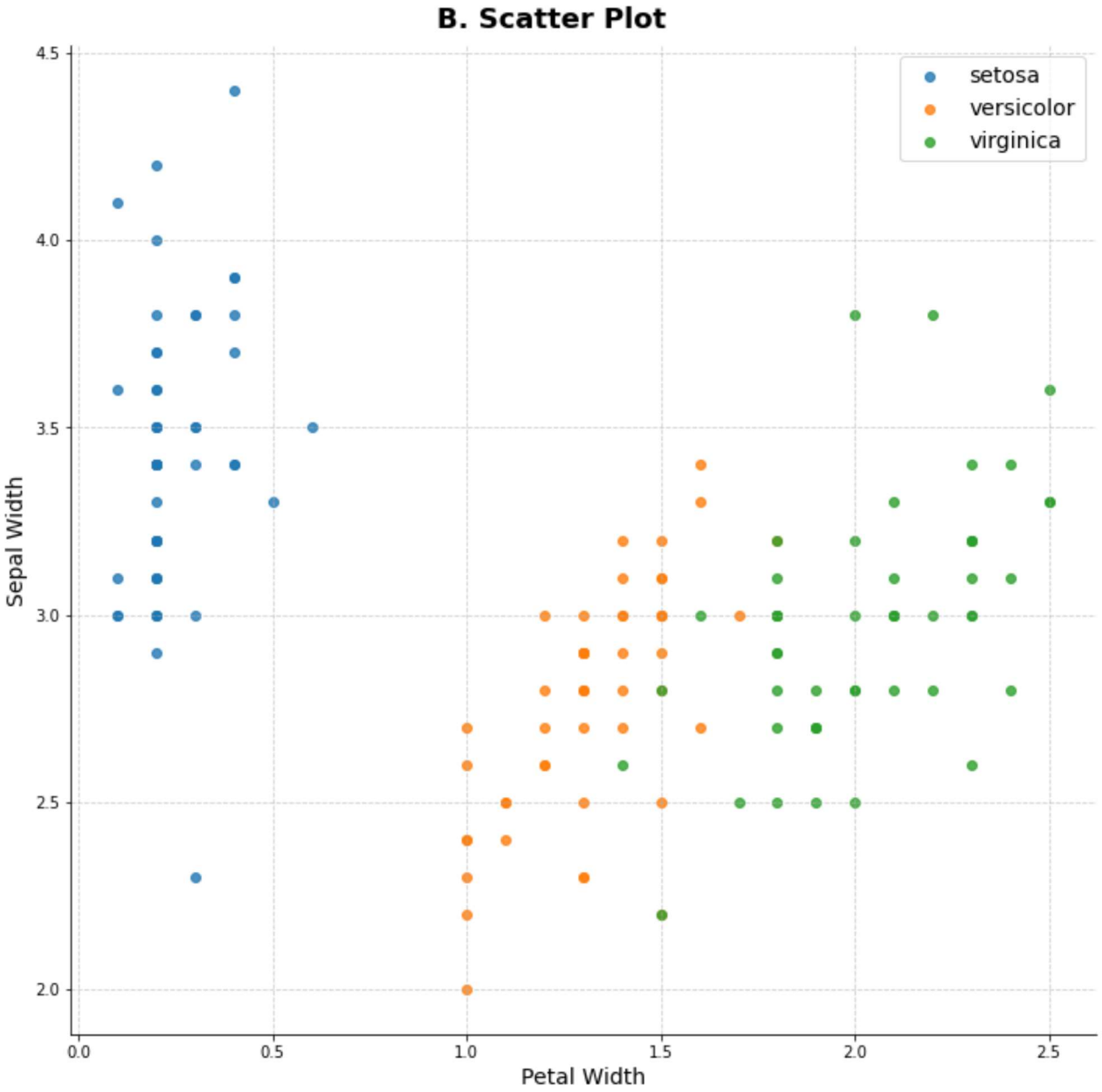
```
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


# iris dataset
df = datasets.load_iris()
iris = pd.DataFrame(data = df.data, columns = df.feature_names)
t_names = {0:df.target_names[0], 1: df.target_names[1], 2: df.target_names[2]}
iris['type'] = df.target
iris['type'] = iris['type'].map(t_names)
color = ['r','g','b','y']

# A. Bar Chart
fig, ax = plt.subplots(2,2)
fig.set_figwidth(16)
fig.set_figheight(16)

for i in range(len(ax)):
    for j in range(len(ax[0])):
        iris.iloc[:,i+1^j].value_counts().sort_index().plot(kind = 'bar',
                                                            ax = ax[i, j],
                                                            color = color[i+1^j])
        ax[i,j].legend(fontsize = 13)
        ax[i,j].grid(alpha = 0.5, linestyle = '-.')

fig.suptitle('Q.5 Outputs\n\nA. Bar Chart', fontweight = 'bold', fontsize = 25)
fig.tight_layout()
fig.subplots_adjust(top = 0.9)

# B. Scatter Plot
sc = sns.lmplot( x="petal width (cm)", y="sepal width (cm)", data=iris,
                fit_reg=False, hue='type', legend = False, height = 10)
sc.fig.suptitle('B. Scatter Plot', fontsize = 18, fontweight = 'bold')
sc.ax.legend(loc = 'upper right',fontsize = 14)
sc.ax.set_xlabel('Petal Width',fontsize = 14)
sc.ax.set_ylabel('Sepal Width',fontsize = 14)
```

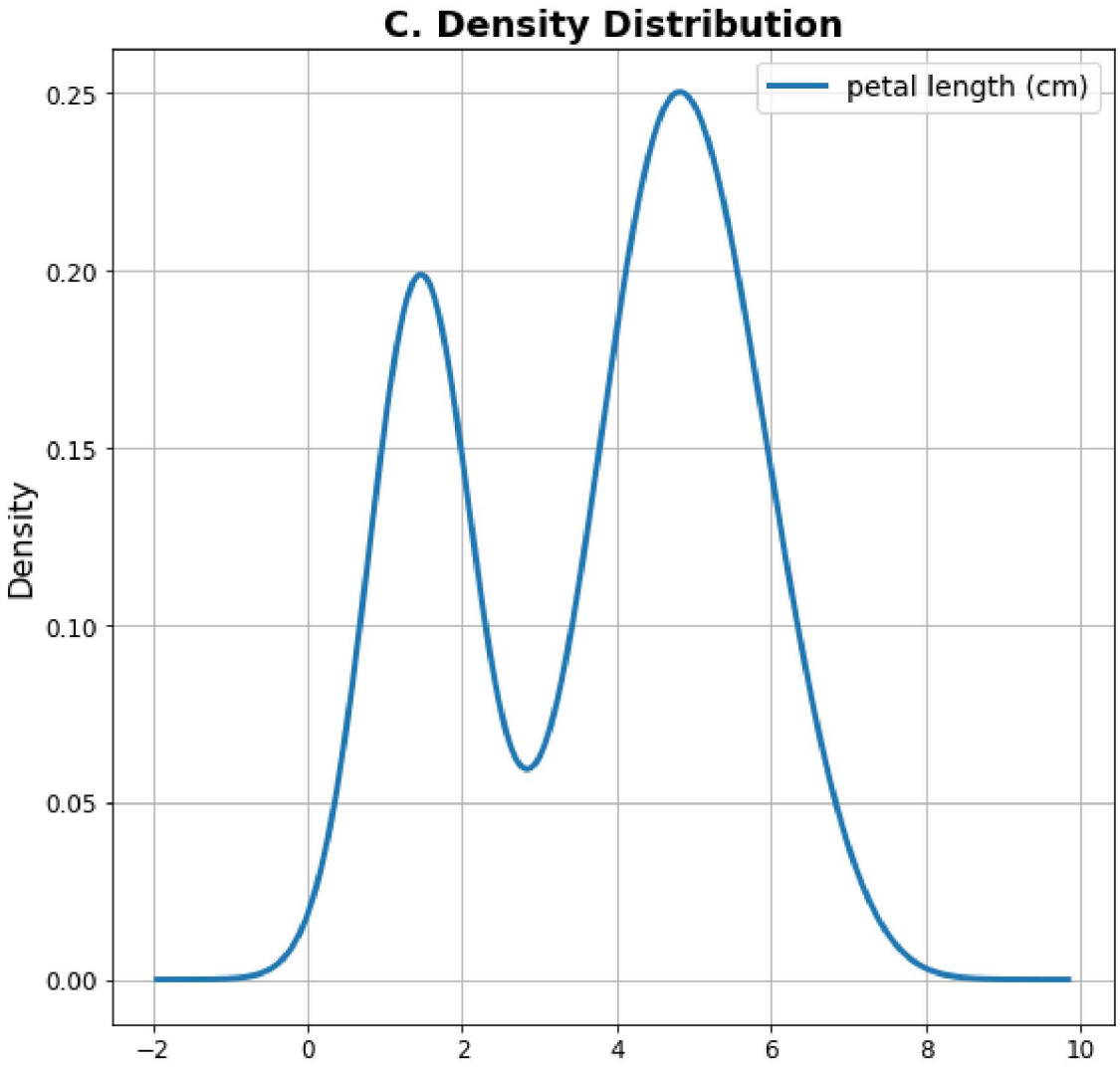
```
sc.ax.grid(linestyle = '--', alpha = 0.6)
sc.tight_layout()
plt.show()

# C. Density Plot
den = iris['petal length (cm)'].plot(kind = 'density', figsize = (9,9),
                                     linewidth = 3, fontsize = 12, grid = True)
den.legend(fontsize = 14)
den.set_ylabel('Density', fontsize = 16)
den.set_title('C. Density Distribution', fontsize = 18, fontweight = 'bold')

# D. Pair Plot
pair = sns.pairplot(iris, hue = 'type')
pair.fig.suptitle('D. Pair-Plot', fontsize = 18, fontweight = 'bold')
pair.tight_layout()
plt.show()
```

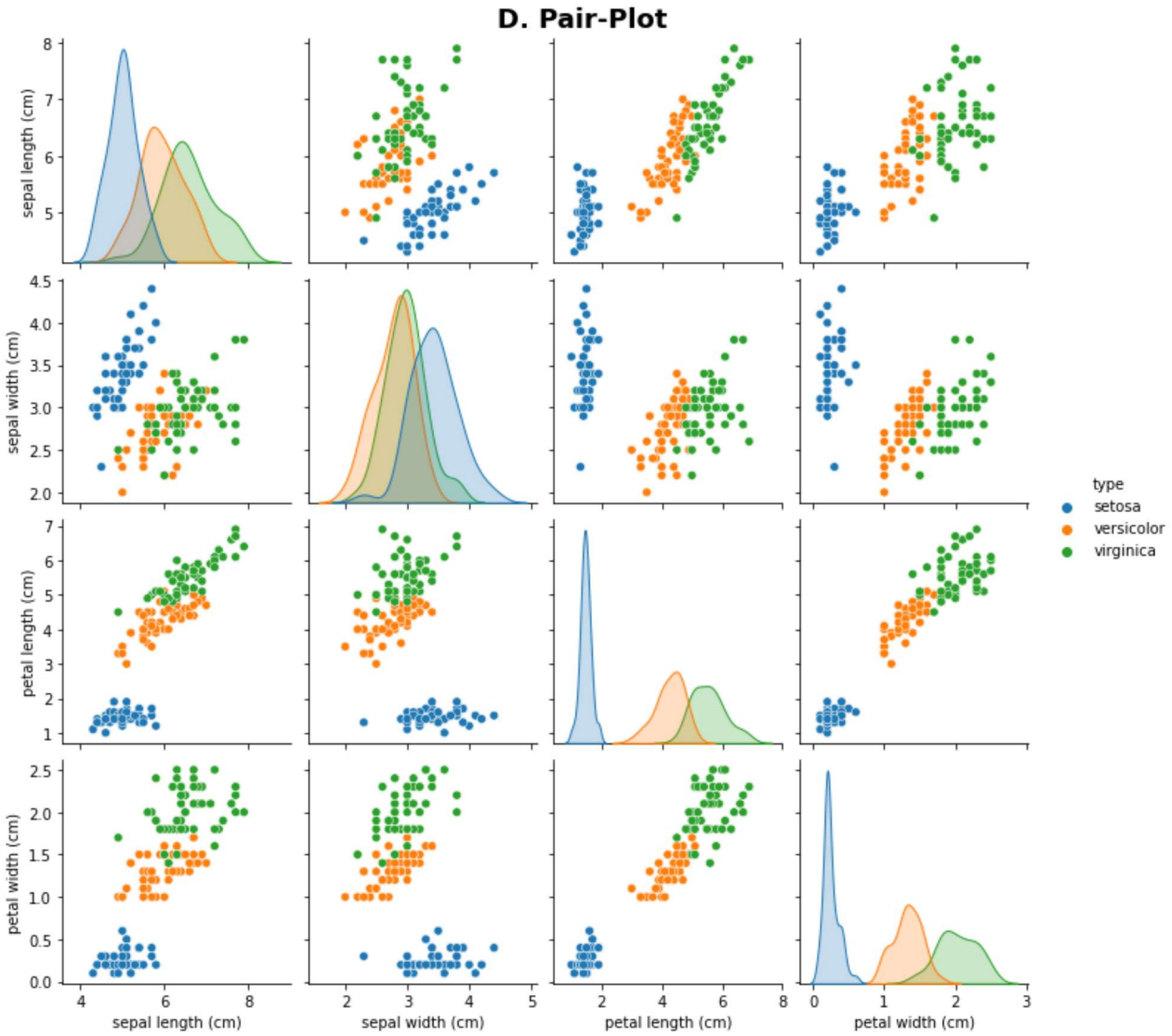
Q.5 Outputs

A. Bar Chart




B. Scatter Plot
setosa
versicolor
virginica
Sepal Width
Petal Width
4.5
4.0
3.5
3.0
2.5
2.0
0.0
0.5
1.0
1.5
2.0
2.5


C. Density Distribution
petal length (cm)
Density
0.25
0.20
0.15
0.10
0.05
0.00
−2
0
2
4
6
8
10



Q.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.

Code)

In [13]:
```
import pandas as pd

df = pd.read_csv('sales_data_sample.csv', encoding='Latin-1')
df.columns
```

Out[13]:
```
Index(['ORDERNUMBER', 'QUANTITYORDERED', 'PRICEEACH', 'ORDERLINENUMBER',
       'SALES', 'ORDERDATE', 'STATUS', 'QTR_ID', 'MONTH_ID', 'YEAR_ID',
       'PRODUCTLINE', 'MSRP', 'PRODUCTCODE', 'CUSTOMERNAME', 'PHONE',
       'ADDRESSLINE1', 'ADDRESSLINE2', 'CITY', 'STATE', 'POSTALCODE',
       'COUNTRY', 'TERRITORY', 'CONTACTLASTNAME', 'CONTACTFIRSTNAME',
       'DEALSIZE'],
      dtype='object')
```

In [14]:
```
df.head()
```

Out[14]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATU
0	10107	30	95.70	2	2871.00	01-07-2003 00:00	Shippe
1	10121	34	81.35	5	2765.90	05-07-2003 00:00	Shippe
2	10134	41	94.74	2	3884.34	NaN	Shippe
3	10145	45	83.26	6	3746.70	01-08-2003 00:00	Shippe
4	10159	49	100.00	14	5205.27	10-10-2003 00:00	Shippe

5 rows × 25 columns

Part A

In [15]:
```
mea= df.groupby(["PRODUCTLINE"]).mean()
print(mea)
```

```
                 ORDERNUMBER  QUANTITYORDERED  PRICEEACH  ORDERLINENUMBER  \
PRODUCTLINE
Classic Cars      10257.432265        35.152017  87.335781         6.445708
Motorcycles       10258.323263        35.235650  82.997553         5.930514
Planes            10269.790850        35.055556  81.740915         7.222222
Ships             10255.794872        34.730769  83.855470         6.799145
Trains            10255.246753        35.220779  75.654675         7.142857
Trucks and Buses  10258.784053        35.803987  87.527940         6.222591
Vintage Cars      10256.967051        34.710049  78.148204         6.316310

                       SALES    QTR_ID  MONTH_ID      YEAR_ID        MSRP
PRODUCTLINE
Classic Cars      4053.377104  2.775595  7.241986  2003.785936  119.885212
Motorcycles       3523.831843  2.670695  6.873112  2003.845921   97.069486
Planes            3186.286176  2.614379  6.800654  2003.918301   88.767974
```

```
Ships              3053.150128  2.645299  6.931624  2003.816239   86.128205
Trains             2938.226883  2.714286  7.129870  2003.792208   72.987013
Trucks and Buses   3746.810100  2.780731  7.342193  2003.797342  102.465116
Vintage Cars       3135.339110  2.700165  7.054366  2003.803954   86.461285
```

PART B

In [16]:
```
df.fillna(method='ffill').head()
```

Out[16]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATU
0	10107	30	95.70	2	2871.00	01-07-2003 00:00	Shippe
1	10121	34	81.35	5	2765.90	05-07-2003 00:00	Shippe
2	10134	41	94.74	2	3884.34	05-07-2003 00:00	Shippe
3	10145	45	83.26	6	3746.70	01-08-2003 00:00	Shippe
4	10159	49	100.00	14	5205.27	10-10-2003 00:00	Shippe

5 rows × 25 columns

Part C

In [17]:
```
df['ORDERDATE'] = pd.to_datetime(df['ORDERDATE'])
df.head()
```

Out[17]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATU
0	10107	30	95.70	2	2871.00	2003-01-07	Shippe
1	10121	34	81.35	5	2765.90	2003-05-07	Shippe
2	10134	41	94.74	2	3884.34	NaT	Shippe
3	10145	45	83.26	6	3746.70	2003-01-08	Shippe
4	10159	49	100.00	14	5205.27	2003-10-10	Shippe

5 rows × 25 columns

Part D

In [18]:
```
df_agg = df.groupby(['CUSTOMERNAME','PRODUCTCODE']).agg({'PRICEEACH':sum})
result = df_agg['PRICEEACH'].groupby(level=0, group_keys=False)
print("\nGroup on 'CUSTOMERNAME','PRODUCTCODE' and then sort sum of 'PRICEEACH' within
print(result.nlargest())
```

```
Group on 'CUSTOMERNAME','PRODUCTCODE' and then sort sum of 'PRICEEACH' within the group
s:
CUSTOMERNAME        PRODUCTCODE
AV Stores, Co.      S18_4409       189.27
                    S24_2766       164.87
                    S18_2325       164.69
                    S18_4933       161.29
                    S18_2795       152.67
                                    ...
giftsbymail.co.uk   S10_4757       100.00
                    S18_1662       100.00
                    S18_3140       100.00
                    S24_1785       100.00
                    S24_2011       100.00
Name: PRICEEACH, Length: 458, dtype: float64
```

Part E

In [19]:
```
groups = df.groupby(['ORDERNUMBER', pd.cut(df.QUANTITYORDERED, 3)])
result = groups.size().unstack()
print(result.head())
```

```
QUANTITYORDERED  (5.909, 36.333]  (36.333, 66.667]  (66.667, 97.0]
ORDERNUMBER
10100                          2                 2               0
10101                          2                 2               0
10102                          0                 2               0
10103                         11                 5               0
10104                          9                 4               0
```

Q. 7)

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

S.N.	Name	Birth_Month	Gender	Pass_Division
0	Mudit Chauhan	December	M	III
1	Seema Chopra	January	F	II
2	Rani Gupta	March	F	I
3	Aditya Narayan	October	M	I
4	Sanjeev Sahni	February	M	II

S.N.	Name	Birth_Month	Gender	Pass_Division
5	Prakash Kumar	December	M	III
6	Ritu Agarwal	September	F	I
7	Akshay Goel	August	M	I
8	Meeta Kulkarni	July	F	II
9	Preeti Ahuja	November	F	II
10	Sunil Das Gupta	April	M	III
11	Sonali Sapre	January	F	I
12	Rashmi Talwar	June	F	III
13	Ashish Dubey	May	M	II
14	Kiran Sharma	February	F	II
15	Sameer Bansal	October	M	I

---

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.

Code)

In [20]:

```
import pandas as pd
import numpy as np

bm = np.array(['January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', 'November', 'December'])
gen = np.array(['M', 'F'])
p_div = np.array(['I', 'II', 'III'])

df = pd.DataFrame({
    'Name' : np.array(['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': bm[[11,0,2,9,1,11,8,7,6,10,3,0,5,4,1,9]],
    'Gender' : gen[[0,1,1,0,0,0,1,0,1,1,0,1,1,0,1,0]],
    'Pass_Division' : p_div[[2,1,0,0,1,2,0,0,1,1,2,0,2,1,1,0]]
})

print(f'''\t\t\t Q.7 Output
\n-------------------------------------------------------------------------\n
Given DataFrame : \n\n{df}
\n-------------------------------------------------------------------------\n
A. \n
Performing one hot encoding on the last two columns : \n
{pd.get_dummies(df, columns=['Gender', 'Pass_Division'])}
''')
```

```
df['Birth_Month'] = pd.Categorical(df['Birth_Month'], categories=bm)
print(f'''--------------------------------------------------------------------------\n
B. \n
Sorting DataFrame by the Birth_Month : \n
{df.sort_values(by = 'Birth_Month')}
\n--------------------------------------------------------------------------
''')
```

Q.7 Output

```
--------------------------------------------------------------------------

Given DataFrame :

               Name Birth_Month Gender Pass_Division
0     Mudit Chauhan    December      M            III
1      Seema Chopra     January      F             II
2        Rani Gupta       March      F              I
3    Aditya Narayan     October      M              I
4     Sanjeev Sahni    February      M             II
5     Prakash Kumar    December      M            III
6      Ritu Agarwal   September      F              I
7       Akshay Goel      August      M              I
8    Meeta Kulkarni        July      F             II
9      Preeti Ahuja    November      F             II
10  Sunil Das Gupta       April      M            III
11     Sonali Sapre     January      F              I
12    Rashmi Talwar        June      F            III
13     Ashish Dubey         May      M             II
14     Kiran Sharma    February      F             II
15    Sameer Bansal     October      M              I

--------------------------------------------------------------------------

A.

Performing one hot encoding on the last two columns :

               Name Birth_Month  Gender_F  Gender_M  Pass_Division_I  \
0     Mudit Chauhan    December         0         1                0
1      Seema Chopra     January         1         0                0
2        Rani Gupta       March         1         0                1
3    Aditya Narayan     October         0         1                1
4     Sanjeev Sahni    February         0         1                0
5     Prakash Kumar    December         0         1                0
6      Ritu Agarwal   September         1         0                1
7       Akshay Goel      August         0         1                1
8    Meeta Kulkarni        July         1         0                0
9      Preeti Ahuja    November         1         0                0
10  Sunil Das Gupta       April         0         1                0
11     Sonali Sapre     January         1         0                1
12    Rashmi Talwar        June         1         0                0
13     Ashish Dubey         May         0         1                0
14     Kiran Sharma    February         1         0                0
15    Sameer Bansal     October         0         1                1

    Pass_Division_II  Pass_Division_III
0                  0                 1
1                  1                 0
```

```
2                  0                  0
3                  0                  0
4                  1                  0
5                  0                  1
6                  0                  0
7                  0                  0
8                  1                  0
9                  1                  0
10                 0                  1
11                 0                  0
12                 0                  1
13                 1                  0
14                 1                  0
15                 0                  0

----------------------------------------------------------------------

B.

Sorting DataFrame by the Birth_Month :

               Name Birth_Month Gender Pass_Division
1      Seema Chopra     January      F             II
11     Sonali Sapre     January      F              I
4     Sanjeev Sahni    February      M             II
14     Kiran Sharma    February      F             II
2        Rani Gupta       March      F              I
10  Sunil Das Gupta       April      M            III
13     Ashish Dubey         May      M             II
12    Rashmi Talwar        June      F            III
8    Meeta Kulkarni        July      F             II
7       Akshay Goel      August      M              I
6      Ritu Agarwal   September      F              I
3    Aditya Narayan     October      M              I
15    Sameer Bansal     October      M              I
9      Preeti Ahuja    November      F             II
0     Mudit Chauhan    December      M            III
5     Prakash Kumar    December      M            III

----------------------------------------------------------------------
```

Q. 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

Name	Gender	MonthlyIncome(Rs.)
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.

Code)

In [21]:

```
import pandas as pd
import numpy as np

name = np.array(['Shah', 'Vats', 'Kumar'])
gender = np.array(['Male', 'Female'])

f_inc = pd.DataFrame({
    'Name' : name[[0,1,1,2,1,2,0,0,2,1]],
    'Gender' : gender[[0,0,1,1,1,0,0,1,1,0]],
    'MonthlyIncome' : np.array([114000, 65000, 43150, 69500, 155000,
                                103000, 55000, 112400, 81030, 71900])
})

print(f'''\t\t\t Q.8 Output
\n-------------------------------------------------------------------------\n
Given DataFrame : \n\n{f_inc}
\n-------------------------------------------------------------------------\n
A. \n
Calculating Familywise Gross Monthly Income : \n
{f_inc.groupby(by = ['Name'])['MonthlyIncome'].sum()}
\n-------------------------------------------------------------------------\n
B. \n
Calculating Highest Monthly Income in family : \n
{f_inc.groupby(by = ['Name', 'Gender'])['MonthlyIncome'].max()}
\n-------------------------------------------------------------------------\n
C. \n
Calculating Monthly Income greater than Rs. 60000.00: \n
{f_inc[f_inc.MonthlyIncome > 60000]}
```

```
\n-------------------------------------------------------------------------\n
D. \n
Calculating the average monthly income of the female members
in the Shah family :
{f_inc[(f_inc.Name == 'Shah') & (f_inc.Gender == 'Female')]['MonthlyIncome'].mean()}
\n-------------------------------------------------------------------------
''')
```

Q.8 Output

```
-------------------------------------------------------------------------

Given DataFrame :

    Name  Gender  MonthlyIncome
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

-------------------------------------------------------------------------

A.

Calculating Familywise Gross Monthly Income :

Name
Kumar    253530
Shah     281400
Vats     335050
Name: MonthlyIncome, dtype: int32

-------------------------------------------------------------------------

B.

Calculating Highest Monthly Income in family :

Name   Gender
Kumar  Female     81030
       Male      103000
Shah   Female    112400
       Male      114000
Vats   Female    155000
       Male       71900
Name: MonthlyIncome, dtype: int32

-------------------------------------------------------------------------

C.

Calculating Monthly Income greater than Rs. 60000.00:

    Name  Gender  MonthlyIncome
```

```
0   Shah    Male         114000
1   Vats    Male          65000
3  Kumar  Female          69500
4   Vats  Female         155000
5  Kumar    Male         103000
7   Shah  Female         112400
8  Kumar  Female          81030
9   Vats    Male          71900

--------------------------------------------------------------------------

D.

Calculating the average monthly income of the female members
in the Shah family :
112400.0

--------------------------------------------------------------------------
```

End