Name Aparna Raj

Div-F

Roll no-605

ASSIGNMENT-1a

```
f1=open("/content/sample_data/cityaparna.csv","r")
f2=open("/content/sample_data/salaparna.csv","r")
f3=open("/content/sample_data/emp.csv","w")
contents1=f1.read()
contents2=f2.read()
print(contents1)
print(contents2)
nm=[]
sal=[]
lines1=contents1.split("\n")
lines2=contents2.split("\n")
for l1 in lines1:
    words1=l1.split(",")

    for l2 in lines2:
        words2=l2.split(",")
        if(words1[0]==words2[0]):
            l1=l1+","+words2[1]+words2[2]+"\n"
            f3.write(l1)
```

```
        nm.append(words1[1])
        sal.append(int(words2[2]))
```

output:-

```
1,Shruti,Pune
2,Komal,Mumbai
3,Mansi,Kanpur
4,Divya,Lucknow

Manager,6789987
Sr.Manager,6976578
Manager,797046
Sr .Manager,67890
```

ASSIGNMENT-1a

#Find Statical analysis of employee records

```
f=open("/content/sample_data/emp.csv",”r”)
contents= f.read()
lines=contents.split("\n")
eid=[]; nm=[];desgn=[];sal=[];
for l in lines:
    words = l.split(",")
    print(words)
    eid.append(int(words[0]))
```

```
    nm.append(words[1])
    desgn.append(words[2])
    sal.append(int(words[3]))

    print("Employee ID:",eid)
    print("Employee Names:",nm)
    print("Employee DEsignations:",desgn)
    print("Employee Salary:",sal)
    print("Maximum Salary:",max(sal))
    print("Minimum Salary:",min(sal))
    print("Average Salary:",sum(sal)/len(sal))
    print("Total Salary:",sum(sal))
    print("Employee Name whose salary is maximum",nm[sal.index(max(sal))])
    print("Employee Name whose designation is manager",end="")
    for i in range(len(desgn)):
        if desgn[i]=="Manager"or desgn[i]=="manager":
            print(nm[i])
print("Employee Name whose salary is 500000:", nm[sal.index(500000)])
print("Employee Name whose salary is minimum:",nm[sal.index(min(sal))])
print("Employee Name whose designation is Sr.Manager",end="")
for i in range(len(desgn)):
    if desgn[i]=="Sr.Manager" or desgn[i]=="Sr.manager":
        print(nm[i],end="")

for i in range(len(sal)):
```

```
    if sal[i]==45000:
        print("\nEmployee Name whose salary is 45000;",nm[i])
        f=1
if(f==0):
     print("\nNo any Employee present whose salary is 45000:",nm[i])
```

OUTPUT

```
['1', 'SANVI', 'MANAGER', '100000']
['2', 'MRUNMAYEE', 'SR.MANAGER', '95000']
['3', 'JAYESH', 'MANAGER', '8000']
['4', 'GAURI', 'SR.MANAGER', '95000']
['5', 'MAHESH', 'SUPERVISOR', '500000']
Employee IDs: [1, 2, 3, 4, 5]
Employee Names: ['SANVI', 'MRUNMAYEE', 'JAYESH', 'GAURI', 'MAHESH']
Employee DEsignations: ['MANAGER', 'SR.MANAGER', 'MANAGER', 'SR.MANAGER', 'SUPERVISOR']
Employee Salary: [100000, 95000, 8000, 95000, 500000]
maximum salary: 500000
maximum salary: 8000
average salary: 159600.0
total salary: 798000
Employee name whose salary is maximum MAHESH
Employee name whose designation is manager Employee name whose salary is 100000 :    SANVI
Employee name whose designation is Sr.manager,SANVI JAYESH
Employee name whose salary is 95000: MRUNMAYEE

Employee name whose salary is 95000: GAURI
```

ASSIGNMENT-1b

```
import csv
def top_5_emp(d3):
    d3.sort(key= lambda x: int(x[4]),reverse= True)
    print("Sorted Data:",d3)


    print("\n\nTop1 Employee",d3[0][1] , d3[0][4])
    print("Top2 Employee",d3[1][1] , d3[1][4])
    print("Top1 Employee",d3[2][1] ,d3[2][4])
    print("Top2 Employee",d3[3][1] , d3[3][4])
    print("Top2 Employee",d3[4][1] ,d3[4][4])



f1=open("/sal.csv","r")
f2=open("/city.csv","r")
f3=open("/city_sal.csv","w")

d1=list(csv.reader(f1,delimiter=','))
d2=list(csv.reader(f2,delimiter=','))

print("\n\nFile1 Contents:",d1)
print("\n\nFile2 Contents:",d2)
d3=[]
for i in range(len(d1)):
        d3.append(d1[i]+d2[i])
```

```
#print(d3)

cw=csv.writer(f3)

cw.writerows(d3)


##top_5_emp(d3)




f1.close()

f2.close()

f3.close()
```

OUTPUT

file1 contents: [['shruti', 'nashik'], ['vaishnavi', 'nagpur'], ['komal', 'solapur'], ['aparna', 'pune'], ['shruti', 'mumbai']]

file2 contents: [['shruti', 'manager', '1000'], ['vaishnavi ', 'SR.manager', '2000'], ['komal', 'peon', '3000'], ['aparna', 'CEO', '4000'], ['shruti', 'employee', '5000']]

[['shrutii', 'nashik', 'shruti', 'manager', '1000'], ['vaishnavi', 'nagpur', 'vaishnavi ', 'SR.manager', '2000'], ['komal', 'solapur', 'komal', 'peon', '3000'], ['aparna', 'pune', 'aparna', 'CEO', '4000'], ['shruti', 'mumbai', 'shruti', 'employee', '5000']]

Sorted data : [['shruti', 'nashik', 'shruti', 'employee', '5000'], ['aparna', 'pune', 'aparna', 'CEO', '4000'], ['komal', 'solapur', 'komal', 'peon', '3000'], ['vaishnavi', 'nagpur', 'vaishnavi ', 'SR.manager', '2000'], ['shruti', 'nashik', 'shruti', 'manager', '1000']]

Top1 Employee mumbai 5000

top2 Employee pune 4000

top1 Employee solapur 3000

top2 Employee nagpur 2000

top2 Employee nashik 1000

ASSIGNMENT-1c

Read the birth date of the employee record perform data transformation for birthday age and also salary

#which is in rupees to salary in dollars.

```
import datetime

import csv

f=open("/content/sample_data/empbday.csv","r")


data=list(csv.reader(f))


print(data)


from datetime import date


def calculateAge(birthdate):

    today=date.today()

    age=today.year-birthdate.year-((today.month,today.day)<(birthdate.month,birthdate.day))

    return age


    bdate=[]
```

```
    age=[]
    dollars=[]

for i in range(len(data)):
    print(data[i][1])

    bdate.append(datetime.datetime.strptime(data[i][3],'%d-%m-%Y').date())
    print("birthdate=",bdate)

for i in range(len(data)):
    age.append(calculateAge(bdate[i]))
    dollars.append((float(data[i][4]))/82)

print("Age=",age)
print("salary=",dollars)
```

OUTPUT

[['1', 'shruti', 'nashik', '6-3-2004', '1000'], ['2', 'aparna', 'pune', '12-1-2004', '500'], ['3', 'vaishnavi', 'sambajinagar', '13-12-2005', '300'], ['4', 'omkar', 'shirdi', '4-6-2002', '100'], ['5', 'vaishali', 'ahamadnagar', '6-12-3005', '400']]

shruti

bithdate= [datetime.date(2004, 3, 6)]

aparna

bithdate= [datetime.date(2004, 3, 6), datetime.date(2004, 1, 12)]

vaishnavi

bithdate= [datetime.date(2004, 3, 6), datetime.date(2004, 1, 12), datetime.date(2005, 12, 13)]

omkar

bithdate= [datetime.date(2004, 3, 6), datetime.date(2004, 1, 12), datetime.date(2005, 12, 13), datetime.date(2002, 6, 4)]

vaishali

bithdate= [datetime.date(2004, 3, 6), datetime.date(2004, 1, 12), datetime.date(2005, 12, 13), datetime.date(2002, 6, 4), datetime.date(3005, 12, 6)]

Age= [19, 19, 17, 20, -983]

salary= [12.195121951219512, 6.097560975609756, 3.658536585365854, 1.2195121951219512, 4.878048780487805]

_____

NAME APARNA RAJ

DIV-F

Roll no-605

PRN no-202201070142

```
product_details=[]
Supplier_details=dict()
Customer_details=[]
Gender={}
fpl=open("/content/sample_data/sales620.csv","r")
data=fpl.readline()
while(True):
data=fpl.readline()
if not data:
break;
#print(data)
data=data.replace("¥n","")
temp=data.split(",")
Product_details.append(temp[1])
Customer_details.append(temp[3])
Supplier_details.update({temp[0]:temp[2]})
Gender.update({temp[3]:temp[4]})
fpl.close()
```

```
Customer_details=tuple(Customer_details)

print(type(Customer_details))

print("¥nProduct_details¥n",Product_details,end="")

print("¥n¥nCustomer_details¥n",Customer_details,end="")

print("¥n¥nSupplier_details¥n",Supplier_details,end="")

print("¥n¥nGender_details¥n",Gender,end="")
```

OUTPUT:

```
<class 'tuple'>

Product_details

['Lenovo Laptop', 'Samsung A9', 'Realmi 10pro', 'Oppo F21', 'LG TV']

Customer_details

('Riya Jha', 'Chirag Gupta', 'Priya Acharya', 'David Bonal', 'Shlok Gupta')

Supplier_details

{'S0001': 'Vijay sales', 'S0002': 'Surya Ele.', 'S0003': 'Vijay sales', 'S0004': 'Surya Ele.', 'S0005': 'Vijay

sales'}

Gender_details

{'Riya Jha': 'Female', 'Chirag Gupta': 'Male', 'Priya Acharya': 'Female', 'David Bonal': 'Male', 'Shlok

Gupta': 'Male'}
```

Most popular product for sales.

```
frequency = {}#{Lenovo Laptop:3}

for item in Product_details:

if item in frequency:

frequency[item] += 1

else:

frequency[item] = 1
```

```
print(frequency)

marklist = sorted(frequency.items(),key=lambda x:x[1],reverse=True)

sortdict = dict(marklist)

print(sortdict)

print("The most popular product for sales",list(sortdict.keys())[0],"sold",list(sortdict.values())[0],"times")
```

OUTPUT:

```
{'Lenovo Laptop': 1, 'Samsung A9': 1, 'Realmi 10pro': 1, 'Oppo F21': 1, 'LG TV': 1}

{'Lenovo Laptop': 1, 'Samsung A9': 1, 'Realmi 10pro': 1, 'Oppo F21': 1, 'LG TV': 1}

The most popular product for sales Lenovo Laptop sold 1 times
```

Best supplier for sales.

```
frequency = {}

for item in Supplier_details.values():

if item in frequency:

frequency[item] += 1

else:

frequency[item] = 1

print(frequency)

marklist = sorted(frequency.items(),key=lambda x:x[1],reverse=True)

sortdict = dict(marklist)

print(sortdict)

print("The most popular supplier for sales",list(sortdict.keys())[0],"sold",list(sortdict.values())[0],"items")
```

OUTPUT:

```
{'Vijay sales': 3, 'Surya Ele.': 2}

{'Vijay sales': 3, 'Surya Ele.': 2}

The most popular supplier for sales Vijay sales sold 3 items
```

Customer who buys most of the products.

```
frequency = {}

for item in Customer_details:

if item in frequency:

frequency[item] += 1

else:

frequency[item] = 1

print("Freqenct is as below:¥n",frequency)

marklist = sorted(frequency.items(),key=lambda x:x[1],reverse=True)

sortdict = dict(marklist)

print("¥nSorted dict is as below:¥n",sortdict)

print("¥n¥nThe customer who buys most of the

products",list(sortdict.keys())[0],"buy",list(sortdict.values())[0],"items")
```

OUTPUT:

```
Freqenct is as below:

{'Riya Jha': 1, 'Chirag Gupta': 1, 'Priya Acharya': 1, 'David Bonal': 1, 'Shlok Gupta': 1}

Sorted dict is as below:

{'Riya Jha': 1, 'Chirag Gupta': 1, 'Priya Acharya': 1, 'David Bonal': 1, 'Shlok Gupta': 1}

The customer who buys most of the products Riya Jha buy 1 items
```

Identify Unique Cusomer

```
from collections import Counter

counter = dict(Counter(Customer_details))

names=list(counter.keys())

print(names)

male=0
```

```
female=0

for name in names:

if Gender[name]=="Male":

male=male+1

if Gender[name]=="Female":

female+=1

print("Total no of Male=",male)

print("Total no of Female=",female)
```

OUTPUT:

```
['Riya Jha', 'Chirag Gupta', 'Priya Acharya', 'David Bonal', 'Shlok Gupta']

Total no of Male= 3

Total no of Female= 2
```

Name :APARNA RAJ

, RollNO -

605,DIV-F

BATCH-: F1

PRN-: 202201070142

```
import numpy as np

# Load the datasets into arrays
data1 = np.genfromtxt('testmarks1.csv', delimiter='\t', skip_header=1)
data2 = np.genfromtxt('testmarks2.csv', delimiter='\t', skip_header=1)

# Matrix Operations
# Addition
matrix_sum = data1 + data2

# Subtraction
matrix_diff = data1 - data2

# Multiplication
matrix_product = np.matmul(data1[:, 1:], data2[:, 1:].T)

# Transpose
matrix_transpose = data1.T

# Horizontal and Vertical Stacking
horizontal_stack = np.hstack((data1, data2))
vertical_stack = np.vstack((data1, data2))

# Custom Sequence Generation
custom_sequence = np.arange(10, 51, 10)
```

```
# Arithmetic and Statistical Operations
# Mean
mean = np.mean(data1)

# Standard Deviation
std_dev = np.std(data1)

# Minimum
minimum = np.min(data1)

# Maximum
maximum = np.max(data1)

# Mathematical Operations
# Square Root
sqrt = np.sqrt(data1)

# Exponential
exp = np.exp(data1)

# Bitwise Operators
bitwise_and = np.bitwise_and(data1.astype(int), data2.astype(int))
bitwise_or = np.bitwise_or(data1.astype(int), data2.astype(int))

# Copying and Viewing Arrays
copy_array = data1.copy()
view_array = data1.view()

# Data Stacking
data_stack = np.column_stack((data1, data2))
```

```
# Searching
index = np.where(data1 == 40.9)

# Sorting
sorted_data = np.sort(data1, axis=0)

# Counting
unique_values, counts = np.unique(data1[:, 1], return_counts=True)

# Broadcasting
broadcasted_array = data1 + 10

# Displaying the results
print("Matrix Sum:")
print(matrix_sum)
print("\nMatrix Difference:")
print(matrix_diff)
print("\nMatrix Product:")
print(matrix_product)
print("\nMatrix Transpose:")
print(matrix_transpose)
print("\nHorizontal Stack:")
print(horizontal_stack)
print("\nVertical Stack:")
print(vertical_stack)
print("\nCustom Sequence:")
print(custom_sequence)
print("\nMean:")
print(mean)
print("\nStandard Deviation:")
```

```
print(std_dev)
print("\nMinimum:")
print(minimum)
print("\nMaximum:")
print(maximum)
print("\nSquare Root:")
print(sqrt)
print("\nExponential:")
print(exp)
print("\nBitwise AND:")
print(bitwise_and)
print("\nBitwise OR:")
print(bitwise_or)
print("\nCopied Array:")
print(copy_array)
print("\nView Array:")
print(view_array)
print("\nData Stack:")
print(data_stack)
print("\nIndex of 40.9 in data1:")
print(index)
print("\nSorted Data:")
print(sorted_data)
print("\nUnique Values and Counts:")
print(unique_values, counts)
print("\nBroadcasted Array:")
print(broadcasted_array)
```

Output:

```
Matrix Sum:
[[1602.    71.53   61.97   59.26   50.02]
 [1604.    71.57   62.24   59.66   50.71]
 [1606.    68.4    59.55   56.36   48.16]
 [1608.    65.4    57.55   54.94   47.09]
 [1610.    67.     57.35   55.49   46.47]
```

```
 [1612.     64.92   56.85   54.04   46.26]
 [1614.     67.84   57.02   55.8    45.97]
 [1616.     69.63   60.54   56.96   48.29]
 [1618.     73.38   62.7    60.86   50.89]
 [1620.     77.3    65.3    62.68   51.63]]

Matrix Difference:
[[ 0.   14.57 -6.39 -1.86  5.56]
 [ 0.   15.37 -5.2  -1.7   5.07]
 [ 0.   16.08 -3.23 -0.04  3.1 ]
 [ 0.   13.08 -5.23 -2.62  5.23]
 [ 0.   14.8  -5.29 -0.95  4.83]
 [ 0.   14.02 -4.23 -1.42  4.16]
 [ 0.   15.52 -5.76 -0.22  4.95]
 [ 0.   14.75 -5.32 -0.7   4.13]
 [ 0.   16.12 -6.   -1.2   5.53]
 [ 0.   16.6  -7.54 -0.08  5.43]]

Matrix Product:
[[3670.7699 3661.4676 3433.9648 3406.1468 3382.4896 3325.1596 3372.376
  3537.4409 3707.9462 3861.2343]
 [3718.4627 3708.7576 3478.0157 3450.2001 3426.2988 3368.0122 3416.1717
  3583.285  3756.0027 3911.6643]
 [3595.8285 3585.3246 3360.4967 3335.8215 3312.727  3255.4027 3303.3737
  3464.1376 3631.7204 3783.285 ]
 [3392.6904 3384.3192 3174.7776 3148.0944 3126.3816 3073.6692 3116.964
  3270.     3427.0908 3568.878 ]
 [3458.1081 3448.9982 3233.9342 3208.7108 3186.342  3131.9908 3176.9399
  3332.01   3493.0276 3637.5752]
 [3387.8333 3378.7632 3168.3294 3143.2532 3121.5366 3068.2657 3112.4063
  3264.5992 3421.9367 3564.0835]
 [3478.318  3469.046  3252.1663 3227.5485 3204.8906 3150.0459 3195.457
  3351.0376 3513.4454 3658.6088]
 [3587.5821 3577.6888 3354.1456 3328.525  3305.425  3248.7103 3295.8567
  3456.5956 3623.6199 3774.1931]
 [3782.1961 3772.3736 3537.3438 3509.5092 3485.0318 3425.7029 3474.6919
  3644.3812 3820.4427 3978.3859]
 [3915.0043 3904.4672 3660.1961 3632.7021 3607.1972 3545.3782 3596.6185
  3771.6478 3954.5059 4117.9791]]

Matrix Transpose:
[[801.   802.   803.   804.   805.   806.   807.   808.   809.   810.
]
 [ 43.05  43.47  42.24  39.24  40.9   39.47  41.68  42.19  44.75
46.95]
 [ 27.79  28.52  28.16  26.16  26.03  26.31  25.63  27.61  28.35
28.88]
 [ 28.7   28.98  28.16  26.16  27.27  26.31  27.79  28.13  29.83  31.3
]
 [ 27.79  27.89  25.63  26.16  25.65  25.21  25.46  26.21  28.21
28.53]]

Horizontal Stack:
[[801.    43.05  27.79  28.7   27.79 801.    28.48  34.18  30.56
22.23]
 [802.    43.47  28.52  28.98  27.89 802.    28.1   33.72  30.68
22.82]
```

```
 [803.    42.24  28.16  28.16  25.63 803.    26.16  31.39  28.2
22.53]
 [804.    39.24  26.16  26.16  26.16 804.    26.16  31.39  28.78
20.93]
 [805.    40.9   26.03  27.27  25.65 805.    26.1   31.32  28.22
20.82]
 [806.    39.47  26.31  26.31  25.21 806.    25.45  30.54  27.73
21.05]
 [807.    41.68  25.63  27.79  25.46 807.    26.16  31.39  28.01
20.51]
 [808.    42.19  27.61  28.13  26.21 808.    27.44  32.93  28.83
22.08]
 [809.    44.75  28.35  29.83  28.21 809.    28.63  34.35  31.03
22.68]
 [810.    46.95  28.88  31.3   28.53 810.    30.35  36.42  31.38  23.1
]]

Vertical Stack:
[[801.    43.05  27.79  28.7   27.79]
 [802.    43.47  28.52  28.98  27.89]
 [803.    42.24  28.16  28.16  25.63]
 [804.    39.24  26.16  26.16  26.16]
 [805.    40.9   26.03  27.27  25.65]
 [806.    39.47  26.31  26.31  25.21]
 [807.    41.68  25.63  27.79  25.46]
 [808.    42.19  27.61  28.13  26.21]
 [809.    44.75  28.35  29.83  28.21]
 [810.    46.95  28.88  31.3   28.53]
 [801.    28.48  34.18  30.56  22.23]
 [802.    28.1   33.72  30.68  22.82]
 [803.    26.16  31.39  28.2   22.53]
 [804.    26.16  31.39  28.78  20.93]
 [805.    26.1   31.32  28.22  20.82]
 [806.    25.45  30.54  27.73  21.05]
 [807.    26.16  31.39  28.01  20.51]
 [808.    27.44  32.93  28.83  22.08]
 [809.    28.63  34.35  31.03  22.68]
 [810.    30.35  36.42  31.38  23.1 ]]

Custom Sequence:
[10 20 30 40 50]

Mean:
186.03499999999997

Standard Deviation:
309.7929965912722

Minimum:
25.21

Maximum:
810.0

Square Root:
[[28.3019434   6.56124988  5.27162214  5.35723809  5.27162214]
 [28.31960452  6.59317829  5.34041197  5.38330753  5.28109837]
 [28.33725463  6.49923072  5.30659966  5.30659966  5.06260802]
```

```
 [28.35489376  6.26418391  5.11468474  5.11468474  5.11468474]
 [28.37252192  6.39531078  5.10196041  5.22206856  5.0645829 ]
 [28.39013913  6.28251542  5.12932744  5.12932744  5.02095608]
 [28.40774542  6.45600496  5.06260802  5.27162214  5.04579032]
 [28.42534081  6.49538298  5.25452186  5.30377224  5.11957029]
 [28.44292531  6.68954408  5.3244718   5.46168472  5.31130869]
 [28.46049894  6.85200701  5.37401154  5.59464029  5.34134814]]
```

```
Exponential:
[[           inf 4.97024098e+18 1.17231319e+12 2.91240408e+12
  1.17231319e+12]
 [           inf 7.56451570e+18 2.43264437e+12 3.85348866e+12
  1.29560645e+12]
 [           inf 2.21105179e+18 1.69719839e+12 1.69719839e+12
  1.35197161e+11]
 [           inf 1.10081787e+17 2.29690824e+11 2.29690824e+11
  2.29690824e+11]
 [           inf 5.78954335e+17 2.01690463e+11 6.96964281e+11
  1.37928325e+11]
 [           inf 1.38548938e+17 2.66862665e+11 2.66862665e+11
  8.88308645e+10]
 [           inf 1.26297282e+18 1.35197161e+11 1.17231319e+12
  1.14061088e+11]
 [           inf 2.10321752e+18 9.79198288e+11 1.64703859e+12
  2.41467325e+11]
 [           inf 2.72068377e+19 2.05233647e+12 9.01580262e+12
  1.78421561e+12]
 [           inf 2.45542077e+20 3.48678073e+12 3.92118456e+13
  2.45709285e+12]]
```

```
Bitwise AND:
[[801   8   2  28  18]
 [802   8   0  28  18]
 [803  10  28  28  16]
 [804   2  26  24  16]
 [805   8  26  24  16]
 [806   1  26  26  17]
 [807   8  25  24  16]
 [808  10   0  28  18]
 [809  12   0  29  20]
 [810  14   4  31  20]]
```

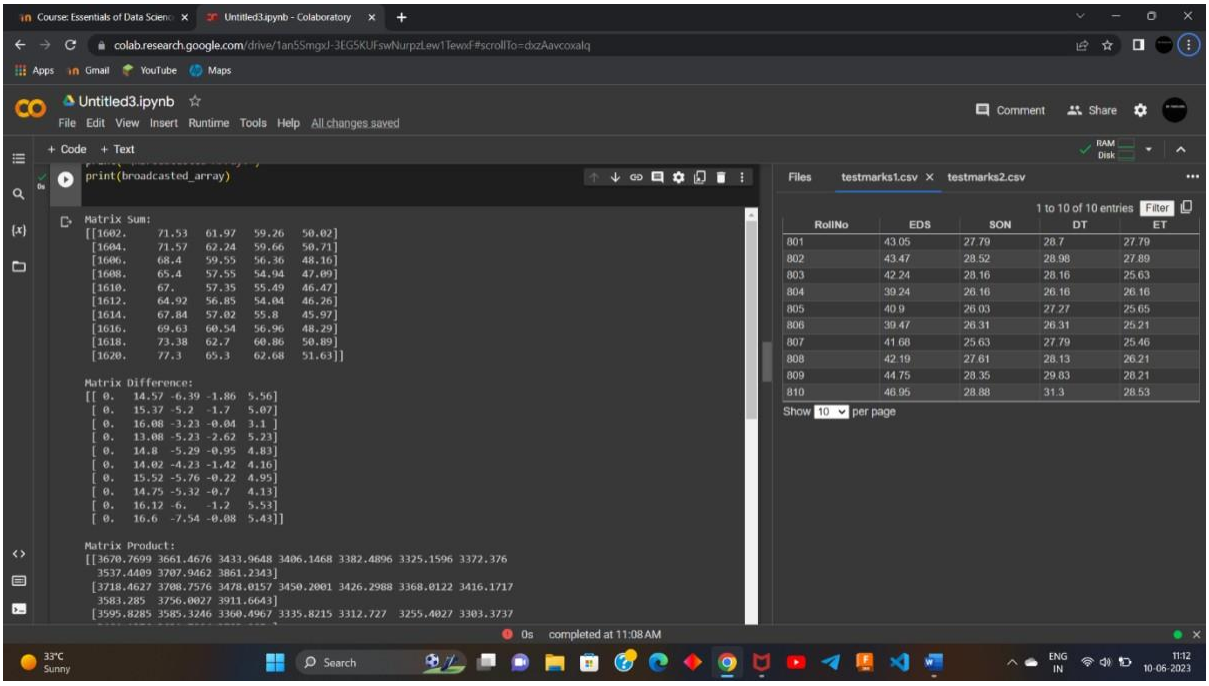
```
Bitwise OR:
[[801  63  59  30  31]
 [802  63  61  30  31]
 [803  58  31  28  31]
 [804  63  31  30  30]
 [805  58  31  31  29]
 [806  63  30  27  29]
 [807  59  31  31  29]
 [808  59  59  28  30]
 [809  60  62  31  30]
 [810  62  60  31  31]]
```

```
Copied Array:
[[801.    43.05  27.79  28.7   27.79]
 [802.    43.47  28.52  28.98  27.89]
 [803.    42.24  28.16  28.16  25.63]
```

```
 [804.    39.24  26.16  26.16  26.16]
 [805.    40.9   26.03  27.27  25.65]
 [806.    39.47  26.31  26.31  25.21]
 [807.    41.68  25.63  27.79  25.46]
 [808.    42.19  27.61  28.13  26.21]
 [809.    44.75  28.35  29.83  28.21]
 [810.    46.95  28.88  31.3   28.53]]

View Array:
[[801.    43.05  27.79  28.7   27.79]
 [802.    43.47  28.52  28.98  27.89]
 [803.    42.24  28.16  28.16  25.63]
 [804.    39.24  26.16  26.16  26.16]
 [805.    40.9   26.03  27.27  25.65]
 [806.    39.47  26.31  26.31  25.21]
 [807.    41.68  25.63  27.79  25.46]
 [808.    42.19  27.61  28.13  26.21]
 [809.    44.75  28.35  29.83  28.21]
 [810.    46.95  28.88  31.3   28.53]]

Data Stack:
[[801.    43.05  27.79  28.7   27.79 801.    28.48  34.18  30.56
22.23]
 [802.    43.47  28.52  28.98  27.89 802.    28.1   33.72  30.68
22.82]
 [803.    42.24  28.16  28.16  25.63 803.    26.16  31.39  28.2
22.53]
 [804.    39.24  26.16  26.16  26.16 804.    26.16  31.39  28.78
20.93]
 [805.    40.9   26.03  27.27  25.65 805.    26.1   31.32  28.22
20.82]
 [806.    39.47  26.31  26.31  25.21 806.    25.45  30.54  27.73
21.05]
 [807.    41.68  25.63  27.79  25.46 807.    26.16  31.39  28.01
20.51]
 [808.    42.19  27.61  28.13  26.21 808.    27.44  32.93  28.83
22.08]
 [809.    44.75  28.35  29.83  28.21 809.    28.63  34.35  31.03
22.68]
 [810.    46.95  28.88  31.3   28.53 810.    30.35  36.42  31.38  23.1
]]

Index of 40.9 in data1:
(array([4]), array([1]))

Sorted Data:
[[801.    39.24  25.63  26.16  25.21]
 [802.    39.47  26.03  26.31  25.46]
 [803.    40.9   26.16  27.27  25.63]
 [804.    41.68  26.31  27.79  25.65]
 [805.    42.19  27.61  28.13  26.16]
 [806.    42.24  27.79  28.16  26.21]
 [807.    43.05  28.16  28.7   27.79]
 [808.    43.47  28.35  28.98  27.89]
 [809.    44.75  28.52  29.83  28.21]
 [810.    46.95  28.88  31.3   28.53]]

Unique Values and Counts:
```

```
[39.24 39.47 40.9  41.68 42.19 42.24 43.05 43.47 44.75 46.95] [1 1 1 1
1 1 1 1 1 1]
```

```
Broadcasted Array:
[[811.     53.05  37.79  38.7   37.79]
 [812.     53.47  38.52  38.98  37.89]
 [813.     52.24  38.16  38.16  35.63]
 [814.     49.24  36.16  36.16  36.16]
 [815.     50.9   36.03  37.27  35.65]
 [816.     49.47  36.31  36.31  35.21]
 [817.     51.68  35.63  37.79  35.46]
 [818.     52.19  37.61  38.13  36.21]
 [819.     54.75  38.35  39.83  38.21]
 [820.     56.95  38.88  41.3   38.53]]
```



Name : Aparna Raj

Roll No : 605

Batch : F1

PRN : 202201070142

Assignment 4

In [8]:
```
import numpy as np
import pandas as pd
all_data=pd.read_csv("/content/sample_data/all_data.csv")
all_data.head()
```

Out [8]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176559.0	Bose SoundSport Headphones	1.0	99.99	04-07-2019 22:30	682 Chestnut St, Boston, MA 02215
1	176560.0	Google Phone	1.0	600.00	04-12-2019 14:38	669 Spruce St, Los Angeles, CA 90001
2	176560.0	Wired Headphones	1.0	11.99	04-12-2019 14:38	669 Spruce St, Los Angeles, CA 90001
3	176561.0	Wired Headphones	1.0	11.99	05/30/19 9:27	333 8th St, Los Angeles, CA 90001
4	176562.0	USB-C Charging Cable	1.0	11.95	04/29/19 13:03	381 Wilson St, San Francisco, CA 94016

Clean up the data

In [9]:
```
all_data.shape
```

Out [9]: (69, 6)

Drop rows of NAN

In [10]:
```
#Find NAN
nan_df=all_data[all_data.isna().any(axis=1)]
display(nan_df.head())

all_data=all_data.dropna(how='all')
all_data.head()
```

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
36	NaN	NaN	NaN	NaN	NaN	NaN
51	NaN	NaN	NaN	NaN	NaN	NaN

Out [10]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176559.0	Bose SoundSport Headphones	1.0	99.99	04-07-2019 22:30	682 Chestnut St, Boston, MA 02215
1	176560.0	Google Phone	1.0	600.00	04-12-2019 14:38	669 Spruce St, Los Angeles, CA 90001
2	176560.0	Wired Headphones	1.0	11.99	04-12-2019 14:38	669 Spruce St, Los Angeles, CA 90001
3	176561.0	Wired Headphones	1.0	11.99	05/30/19 9:27	333 8th St, Los Angeles, CA 90001
4	176562.0	USB-C Charging Cable	1.0	11.95	04/29/19 13:03	381 Wilson St, San Francisco, CA 94016

Get rid of text in order date column

In [11]:
```
all_data=all_data[all_data['Order Date'].str[0:2]!='Or']
print(all_data)
```

```
    Order ID                    Product  Quantity Ordered  Price Each  \
0   176559.0  Bose SoundSport Headphones               1.0       99.99
1   176560.0                Google Phone               1.0      600.00
2   176560.0            Wired Headphones               1.0       11.99
3   176561.0            Wired Headphones               1.0       11.99
4   176562.0        USB-C Charging Cable               1.0       11.95
..       ...                         ...               ...         ...
64  259329.0    Lightning Charging Cable               1.0       14.95
65  259330.0       AA Batteries (4-pack)               2.0        3.84
66  259331.0    Apple Airpods Headphones               1.0      150.00
67  259332.0    Apple Airpods Headphones               1.0      150.00
68  259333.0  Bose SoundSport Headphones               1.0       99.99

          Order Date                      Purchase Address
0   04-07-2019 22:30       682 Chestnut St, Boston, MA 02215
1   04-12-2019 14:38    669 Spruce St, Los Angeles, CA 90001
2   04-12-2019 14:38    669 Spruce St, Los Angeles, CA 90001
3      05/30/19 9:27       333 8th St, Los Angeles, CA 90001
4     04/29/19 13:03  381 Wilson St, San Francisco, CA 94016
..               ...                                    ...
64  09-05-2019 19:00       480 Lincoln St, Atlanta, GA 30301
65    09/25/19 22:01    763 Washington St, Seattle, WA 98101
66     09/29/19 7:00     770 4th St, New York City, NY 10001
67    09/16/19 19:21          782 Lake St, Atlanta, GA 30301
68    09/19/19 18:03   347 Ridge St, San Francisco, CA 94016

[67 rows x 6 columns]
```

Make columns correct type

In [12]:
```
all_data['Quantity Ordered']=pd.to_numeric(all_data['Quantity Ordered'])
all_data['Price each']=pd.to_numeric(all_data['Price Each'])
```

Argument data with additional columns

Add month column

In [13]:
```
all_data['Month']=all_data['Order Date'].str[0:2]
all_data['Month']=all_data['Month'].astype('int32')
all_data.head()
```

Out [13]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Price each	Month
0	176559.0	Bose SoundSport Headphones	1.0	99.99	04-07-2019 22:30	682 Chestnut St, Boston, MA 02215	99.99	4
1	176560.0	Google Phone	1.0	600.00	04-12-2019 14:38	669 Spruce St, Los Angeles, CA 90001	600.00	4
2	176560.0	Wired Headphones	1.0	11.99	04-12-2019 14:38	669 Spruce St, Los Angeles, CA 90001	11.99	4
3	176561.0	Wired Headphones	1.0	11.99	05/30/19 9:27	333 8th St, Los Angeles, CA 90001	11.99	5

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Price each	Month
4	176562.0	USB-C Charging Cable	1.0	11.95	04/29/19 13:03	381 Wilson St, San Francisco, CA 94016	11.95	4

Add month column (Alternative method)

In [14]:
```
all_data['Month 2']=pd.to_datetime(all_data['Order Date']).dt.month
all_data.head()
```

Out [14]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Price each	Month	Month 2
0	176559.0	Bose SoundSport Headphones	1.0	99.99	04-07-2019 22:30	682 Chestnut St, Boston, MA 02215	99.99	4	4
1	176560.0	Google Phone	1.0	600.00	04-12-2019 14:38	669 Spruce St, Los Angeles, CA 90001	600.00	4	4
2	176560.0	Wired Headphones	1.0	11.99	04-12-2019 14:38	669 Spruce St, Los Angeles, CA 90001	11.99	4	4
3	176561.0	Wired Headphones	1.0	11.99	05/30/19 9:27	333 8th St, Los Angeles, CA 90001	11.99	5	5
4	176562.0	USB-C Charging Cable	1.0	11.95	04/29/19 13:03	381 Wilson St, San Francisco, CA 94016	11.95	4	4

Add City Town

In [15]:
```
def get_city(address):
  return address.split(",")[1].strip(" ")

def get_state(address):
  return address.split(",")[2].split(" ")[1]

all_data['city']=all_data['Purchase Address'].apply(lambda x:f"{get_city(x)} ({
all_data.head()
```

Out [15]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Price each	Month	Month 2	city
0	176559.0	Bose SoundSport Headphones	1.0	99.99	04-07-2019 22:30	682 Chestnut St, Boston, MA 02215	99.99	4	4	Boston (MA)
1	176560.0	Google Phone	1.0	600.00	04-12-2019 14:38	669 Spruce St, Los Angeles, CA 90001	600.00	4	4	Los Angeles (CA)

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Price each	Month	Month 2	city
2	176560.0	Wired Headphones	1.0	11.99	04-12-2019 14:38	669 Spruce St, Los Angeles, CA 90001	11.99	4	4	Los Angeles (CA)
3	176561.0	Wired Headphones	1.0	11.99	05/30/19 9:27	333 8th St, Los Angeles, CA 90001	11.99	5	5	Los Angeles (CA)
4	176562.0	USB-C Charging Cable	1.0	11.95	04/29/19 13:03	381 Wilson St, San Francisco, CA 94016	11.95	4	4	San Francisco (CA)

Questions

Question 1:

What was the best month for sales? How much was earned that month?

In [16]:
```
all_data['Sales']=all_data['Quantity Ordered'].astype('int')*all_data['Price Ea
all_data.groupby(['Month']).sum()
```

```
<ipython-input-16-8ba29a3e5d2a>:2: FutureWarning: The default value of numeric_only in
DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False.
Either specify numeric_only or select only columns which should be valid for the function.
  all_data.groupby(['Month']).sum()
```

Out [16]:

Month	Order ID	Quantity Ordered	Price Each	Price each	Month 2	Sales
4	7335546.0	123.0	885.80	885.80	160	1210.76
5	353124.0	2.0	111.98	111.98	10	111.98
6	184076.0	1.0	14.95	14.95	6	14.95
8	726962.0	9.0	23.92	23.92	32	50.83
9	2378802.0	17.0	591.44	591.44	90	616.62
10	550924.0	11.0	10.67	10.67	30	39.69
11	740314.0	19.0	13.66	13.66	44	65.31
12	550635.0	17.0	8.97	8.97	36	50.83

Question 2:

What city sold the most product?

In [22]:
```
Dummycity=all_data.groupby(['city'])
print(Dummycity)
#city_max=all_data.groupby(['city']).sum()
#print(max(city_max))
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fcc83d92b00>
```

In [34]:
```
from itertools import combinations
from collections import Counter

count = Counter()

for row in df2['Grouped']:
  row_list=row.split(',')
  count.update(Counter(combinations(row_list, 2)))

for key,value in count.most_common(10):
  print(key, value)
```

```
('Google Phone', 'Wired Headphones') 1
```

###Question 3:

What product sold the most?Wghy do you think it sold the most?

In [40]:
```
product_group=all_data.groupby('Product')
quantity_ordered=product_group.sum()['Quantity Ordered']
```

```
<ipython-input-40-11142b314e0e>:2: FutureWarning: The default value of numeric_only in
DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False.
Either specify numeric_only or select only columns which should be valid for the function.
  quantity_ordered=product_group.sum()['Quantity Ordered']
```

In [41]:
```
print(quantity_ordered)
```

```
Product
AA Batteries (4-pack)          64.0
AAA Batteries (4-pack)        109.0
Apple Airpods Headphones        3.0
Bose SoundSport Headphones      3.0
Google Phone                    1.0
Lightning Charging Cable        4.0
USB-C Charging Cable            8.0
Wired Headphones                7.0
Name: Quantity Ordered, dtype: float64
```

In [42]:
```
prices=all_data.groupby('Product').mean()['Price Each']
```

```
<ipython-input-42-1f4f73bca841>:1: FutureWarning: The default value of numeric_only in
DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False.
Either specify numeric_only or select only columns which should be valid for the function.
  prices=all_data.groupby('Product').mean()['Price Each']
```

In [43]:
```
print(prices)
```

```
Product
AA Batteries (4-pack)           3.84
AAA Batteries (4-pack)          2.99
Apple Airpods Headphones      150.00
Bose SoundSport Headphones     99.99
Google Phone                  600.00
Lightning Charging Cable       14.95
USB-C Charging Cable           11.95
Wired Headphones               11.99
Name: Price Each, dtype: float64
```

Question 4:

What city sold the most product?

In [28]:

```
df=all_data[all_data['Order ID'].duplicated(keep=False)]
df['Grouped']=df.groupby('Order ID')['Product'].transform(lambda x: ','.join(x))
df2=df[['Order ID', 'Grouped']].drop_duplicates()
print(df['Grouped'])
```

```
1    Google Phone,Wired Headphones
2    Google Phone,Wired Headphones
Name: Grouped, dtype: object
```

```
<ipython-input-28-387d448b896d>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Grouped']=df.groupby('Order ID')['Product'].transform(lambda x: ','.join(x))
```

Name -Aparna Raj

Roll no-605

Batch-F1

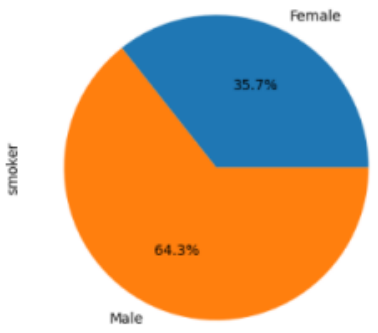
Prn no-202201070142

ASSIGNMENT -5

```
[ ] import pandas as pd
    import matplotlib.pyplot as plt
    d = pd.read_csv('/content/tips.csv')
    print(d)
```

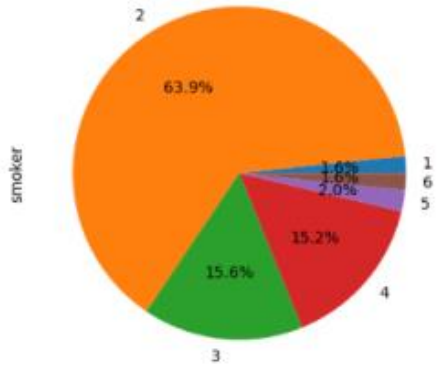
```
     total_bill   tip     sex smoker   day    time  size
0         16.99  1.01  Female     No   Sun  Dinner     2
1         10.34  1.66    Male     No   Sun  Dinner     3
2         21.01  3.50    Male     No   Sun  Dinner     3
3         23.68  3.31    Male     No   Sun  Dinner     2
4         24.59  3.61  Female     No   Sun  Dinner     4
..          ...   ...     ...    ...   ...     ...   ...
239       29.03  5.92    Male     No   Sat  Dinner     3
240       27.18  2.00  Female    Yes   Sat  Dinner     2
241       22.67  2.00    Male    Yes   Sat  Dinner     2
242       17.82  1.75    Male     No   Sat  Dinner     2
243       18.78  3.00  Female     No  Thur  Dinner     2

[244 rows x 7 columns]
```

```
import pandas as pd
import matplotlib.pyplot as plt
d = pd.read_csv('/content/tips.csv')
#print(d)
t1 = d.groupby("sex").count()
t1["smoker"].plot(kind = "pie",autopct = '%1.1f%%')
plt.show()
print(t1)
```



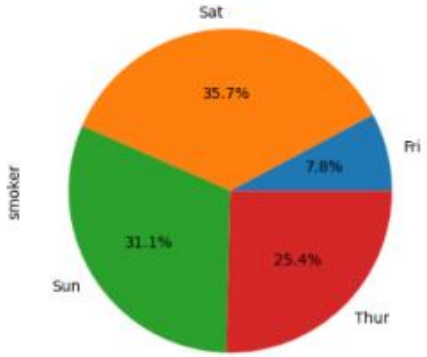

```
        total_bill  tip  smoker  day  time  size
sex
Female          87   87      87   87    87    87
Male           157  157     157  157   157   157
```

```
import pandas as pd
import matplotlib.pyplot as plt
d = pd.read_csv('/content/tips.csv')
#print(d)
t1 = d.groupby("size").count()
t1["smoker"].plot(kind = "pie",autopct = '%1.1f%%')
plt.show()
print(t1)
```



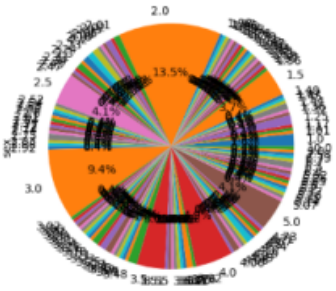

```
      total_bill  tip  sex  smoker  day  time
size
1              4    4    4       4    4     4
2            156  156  156     156  156   156
3             38   38   38      38   38    38
4             37   37   37      37   37    37
5              5    5    5       5    5     5
6              4    4    4       4    4     4
```

```
import pandas as pd
import matplotlib.pyplot as plt
d = pd.read_csv('/content/tips.csv')
#print(d)
t1 = d.groupby("day").count()
t1["smoker"].plot(kind = "pie",autopct = '%1.1f%%')
plt.show()
print(t1)
```




```
      total_bill  tip  sex  smoker  time  size
day
Fri           19   19   19      19    19    19
Sat           87   87   87      87    87    87
Sun           76   76   76      76    76    76
Thur          62   62   62      62    62    62
```

```
import pandas as pd
import matplotlib.pyplot as plt
d = pd.read_csv("/content/tips.csv")
#print(d)
t1 = d.groupby("tip").count()
t1["sex"].plot(kind = "pie",autopct = '%1.1f%%')
plt.show()
print(t1)
```

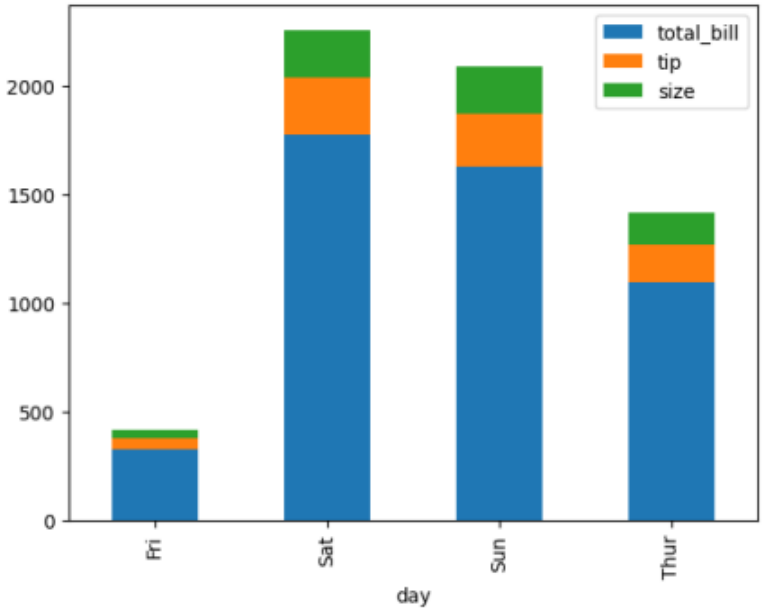


tip	total_bill	sex	smoker	day	time	size
1.00	4	4	4	4	4	4
1.01	1	1	1	1	1	1
1.10	1	1	1	1	1	1
1.17	1	1	1	1	1	1
1.25	3	3	3	3	3	3
...	...	...	...	...	...	...
6.70	1	1	1	1	1	1
6.73	1	1	1	1	1	1
7.58	1	1	1	1	1	1
9.00	1	1	1	1	1	1
10.00	1	1	1	1	1	1

[123 rows x 6 columns]

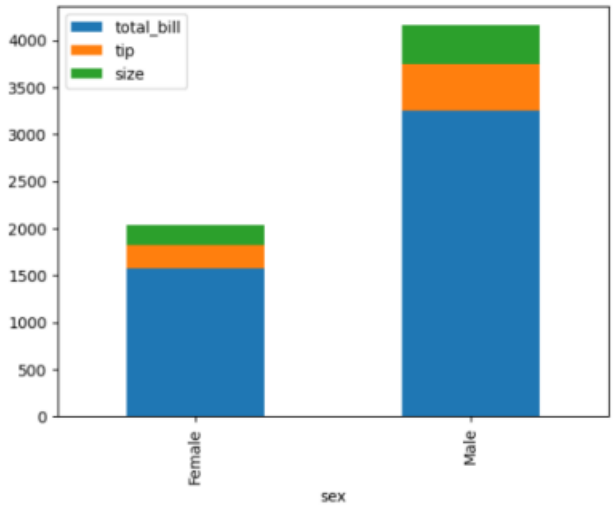
```
t2 = d.groupby("day").sum("total_bill")
t2.plot(kind = "bar",stacked = True)
```

```
<Axes: xlabel='day'>
```



```
t2 = d.groupby("sex").sum("smoker")
t2.plot(kind = "bar",stacked = True)
```

```
<Axes: xlabel='sex'>
```

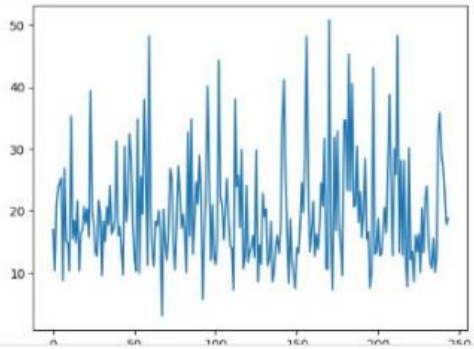



```
import matplotlib.pyplot as plt
d = pd.read_csv('/content/tips.csv')
#print(d)
d1 = d.groupby("total_bill").count()
print(d1)
```

```
             tip  sex  smoker  day  time  size
total_bill
3.07           1    1       1    1     1     1
5.75           1    1       1    1     1     1
7.25           2    2       2    2     2     2
7.51           1    1       1    1     1     1
7.56           1    1       1    1     1     1
...          ...  ...     ...  ...   ...   ...
45.35          1    1       1    1     1     1
48.17          1    1       1    1     1     1
48.27          1    1       1    1     1     1
48.33          1    1       1    1     1     1
50.81          1    1       1    1     1     1

[229 rows x 6 columns]
```

```
d["total_bill"].plot()
```

```
<Axes: >
```

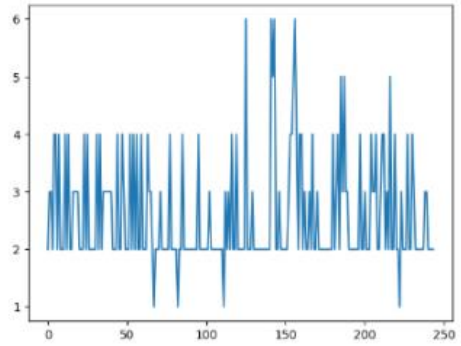


```
import pandas as pd
import matplotlib.pyplot as plt
d = pd.read_csv('/content/tips.csv')
#print(d)
d1 = d.groupby("size").count()
print(d1)
```

```
      total_bill  tip  sex  smoker  day  time
size
1              4    4    4       4    4     4
2            156  156  156     156  156   156
3             38   38   38      38   38    38
4             37   37   37      37   37    37
5              5    5    5       5    5     5
6              4    4    4       4    4     4
```

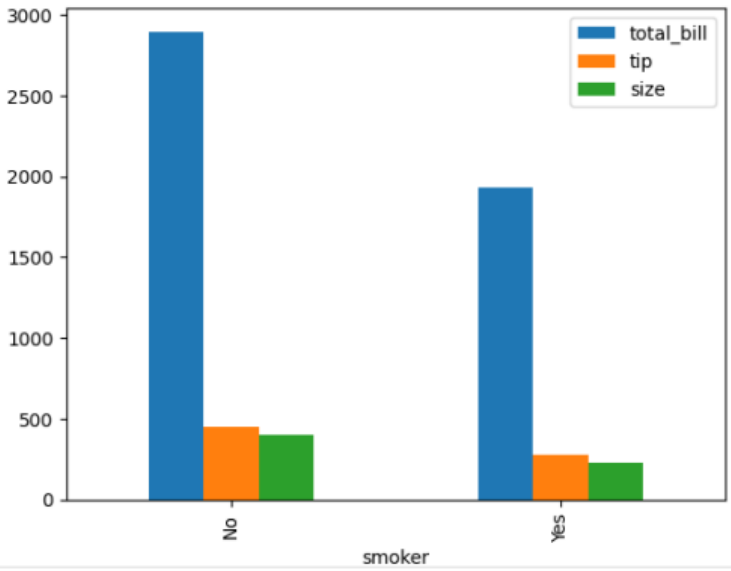
```
d["size"].plot()
```

```
<Axes: >
```



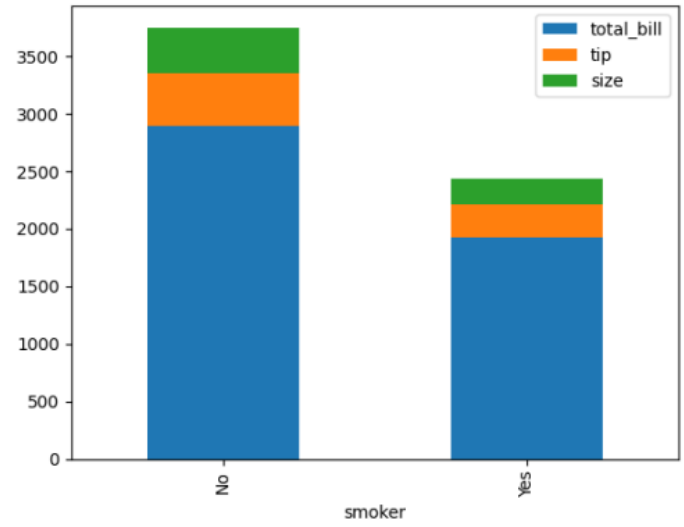

```
t2 = d.groupby("smoker").sum("tip")
t2.plot(kind = "bar",stacked = False)
```

```
<Axes: xlabel='smoker'>
```

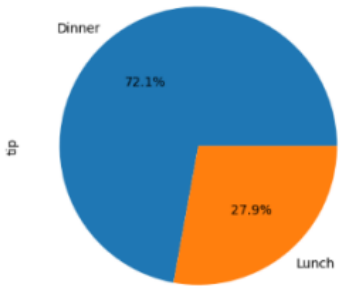


```
t2 = d.groupby("smoker").sum("time")
t2.plot(kind = "bar",stacked = True)
```

```
<Axes: xlabel='smoker'>
```




```
[ ] import pandas as pd
    import matplotlib.pyplot as plt
    d = pd.read_csv('/content/tips.csv')
    #print(d)
    t1 = d.groupby("time").count()
    t1["tip"].plot(kind = "pie",autopct = '%1.1f%%')
    plt.show()
    print(t1)
```




```
        total_bill  tip  sex  smoker  day  size
time
Dinner         176  176  176     176  176   176
Lunch           68   68   68      68   68    68
```