**1. Write a Python program which accepts a sequence of comma-separated numbers from user and generate a list and a tuple with those numbers.**

|  |
| --- |
| **Sample data : 3, 5, 7, 23** |
| **Output :** |
| **List : ['3', ' 5', ' 7', ' 23']** |
| **Tuple : ('3', ' 5', ' 7', ' 23')** |

x=input('enter the values:').split(',')

print(list(x))

​enter the values:1,2,3

['1', '2', '3']

print(tuple(x))

('1', '2', '3')

Out[2]:

**2. Write a Python program to display the first and last colors from the following list.**

**color\_list = ["Red","Green","White" ,"Black"]**

color\_list**=**["Red","Green","White","Black"]

res**=**color\_list[0]

print(res)

Red

res1**=**color\_list[3]

print(res1)

Black

**3. Write a Python program to print the even numbers from a given list.**

|  |
| --- |
| ***Sample List : [1, 2, 3, 4, 5, 6, 7, 8, 9]*** |
| **Expected Result : [2, 4, 6, 8]** |

list=[1,2,3,4,5,6,7,8,9]

for num in list:

if num % 2==0:

print(num)

2

4

6

8

**4. Write a Python program to calculate number of days between two dates. Hint: use Datetime package/module.**

|  |
| --- |
| **Sample dates : (2014, 7, 2), (2014, 7, 11)** |
| **Expected output : 9 days** |

from datetime import date

f\_date=date(2014, 7, 2)

l\_date=date(2014, 7, 11)

delta=l\_date - f\_date

print(delta.days)

9

**5. Write a Python program to get the volume of a sphere with radius 6.**

pi=3.1415

r=6.0

v=4.0/3.0\*pi\*r\*\*3

print('The volume of Sphere is:',v)

The volume of Sphere is: 904.752

**6.Write a Python program to calculate the sum of three given numbers, if the values are equal then return three times of their sum hint: write User defined functions**

def sum(a,b,c):

if(a==b and b==c and c==a):

return a\*\*3

a = int(input("read first number:"))

b = int(input("read second number:"))

c = int(input("read third number:"))

d = sum(a,b,c)

if(d==None):

print("three numbers are not equal")

else:

print(d)

read first number:3

read second number:3

read third number:3

27

**7. Write a Python program to count the number 4 in a given list.**

**List = [1,4,6,8,4,9,4]**

list=[1,4,6,8,4,9,4]

x=4

print('{} has occurred {} times'.format(x,countX(list,x)))

4 has occurred 3 times

1. **Write a Python program to print all even numbers from a given numbers list in the same order and stop the printing if any numbers that come after 237 in the sequence. Go to the editorSample numbers list :**

list1 = [399, 162, 758, 219, 918, 237, 412, 566, 826, 248, 866, 950, 626, 949, 687, 217,

815, 67, 104, 58, 512, 24, 892, 894, 767, 553, 81, 379, 843, 831, 445, 742, 717,

958,743, 527]

for i in list1:

if i== 237:

break;

elif (i%2 == 0):

print(i)

162

758

918

**9. Write a Python program to find those numbers which are divisible by 7 and multiple of 5, between 1500 and 2700 (both included)**

for i in range(1500,2700,1):

if(i%7 == 0 and i%5 == 0):

print(i)

1505

1540

1575

1610

1645

1680

1715

1750

1785

1820

1855

1890

1925

1960

1995

2030

2065

2100

2135

2170

2205

2240

2275

2310

2345

2380

2415

2450

2485

2520

2555

2590

2625

2660

2695

**10.Write a Python program that prints all the numbers from 0 to 6 except 3 and 6.**

for i in range(0,6,1):

if(i==3):

continue;

print(i)

0

1

2

4

5

**11. Write a Python program to get the Fibonacci series between 0 to 50.**

**Note : The Fibonacci Sequence is the series of numbers :**

**0, 1, 1, 2, 3, 5, 8, 13, 21, ....**

**Every next number is found by adding up the two numbers before it.**

**Expected Output : 1 1 2 3 5 8 13 21 34**

nterms = int(input("How many terms? "))

n1, n2 = 0, 1

count = 0

if nterms <= 0:

print("Please enter a positive integer")

elif nterms == 1:

print("Fibonacci sequence upto",nterms,":")

print(n1)

else:

print("Fibonacci sequence:")

while count < nterms:

print(n1)

nth = n1 + n2

n1 = n2

n2 = nth

count += 1

How many terms? 10

Fibonacci sequence:

0

1

1

2

3

5

8

13

21

34

**12. Write a Python function that takes a list and returns a new list with unique elements of the first list.**

l1 = [1,2,3,3,3,3,4,5]

l2 = []

for i in l1:

if i not in l2:

l2.append(i)

print(l2)

[1, 2, 3, 4, 5]

**13. Write a Python program to concatenate all elements in a list into a string and return it.**

def stry(l1):

string = ""

for i in l1:

string += str(i)

return string

l1 = [1,2,3,4,5,6]

stryy = stry(l1)

print(stryy)

123456

**14. Write a Python script to concatenate following dictionaries to create a new one.**

dic1={1:10, 2:20}

dic2={3:30, 4:40}

dic3={5:50, 6:60}

dic4 = { \*\*dic1 ,\*\*dic2, \*\*dic3}

print(dic5)

{1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

**15. Write a Python program to add, subtract, multiple and divide two Pandas Series. Sample Series: [2, 4, 6, 8, 10], [1, 3, 5, 7, 9]**

import pandas as pd

s1 = pd.Series([2, 4, 6, 8, 10])

s2 = pd.Series([1,3,5,7,9])

print(s1+s2)

print(s1-s2)

print(s1\*s2)

print(s1/s2)

0 3

1 7

2 11

3 15

4 19

dtype: int64

0 1

1 1

2 1

3 1

4 1

dtype: int64

0 2

1 12

2 30

3 56

4 90

dtype: int64

0 2.000000

1 1.333333

2 1.200000

3 1.142857

4 1.111111

dtype: float64

**16. Write a Pandas program to select the specified columns and rows from a given data frame.**[**Go to the editor**](https://www.w3resource.com/python-exercises/pandas/index-dataframe.php#EDITOR)**Sample Python dictionary data and list labels:**

**Select 'name' and 'score' columns in rows 1, 3, 5, 6 from the following data frame.**

**exam\_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],**

**score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],**

**attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],**

**qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}**

**labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']**

**Expected Output:**

**Select specific columns and rows:**

**name score**

**b Dima 9.0**

**d James NaN**

**f Michael 20.0**

**g Matthew 14.5**

import pandas as pd

import numpy as np

df1 = pd.DataFrame({'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],\

'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],\

'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],\

'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']})

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

df1

Out[16]:

|  | **name** | **score** | **attempts** | **qualify** |
| --- | --- | --- | --- | --- |
| **0** | Anastasia | 12.5 | 1 | yes |
| **1** | Dima | 9.0 | 3 | no |
| **2** | Katherine | 16.5 | 2 | yes |
| **3** | James | NaN | 3 | no |
| **4** | Emily | 9.0 | 2 | no |
| **5** | Michael | 20.0 | 3 | yes |
| **6** | Matthew | 14.5 | 1 | yes |
| **7** | Laura | NaN | 1 | no |
| **8** | Kevin | 8.0 | 2 | no |
| **9** | Jonas | 19.0 | 1 | yes |

#Select 'name' and 'score' columns in rows 1, 3, 5, 6 from the following data frame.

print(df1.iloc[[1,3,5,6],[0,1]])

name score

1 Dima 9.0

3 James NaN

5 Michael 20.0

6 Matthew 14.5

**21. Write Python Programs to use various operators in Python**

**Arithmetic Operators**

1. Addition Operator

x=10

y=20

print('x+y is',x+y)

x+y is 30

2. Subtraction Operator

x=10

y=20

print('x-y is',x-y)

x-y is -10

3. Division Operator

x=10

y=20

print('x/y is',x/y)

x/y is 0.5

4. Multiplication Operator

x=10

y=20

print('x\*y is',x\*y)

x\*y is 200

5. Percentile Operator

x=20

y=10

print('x%y is',x%y)

x%y is 0

6. Floor Division Operator

x=50

y=20

print('x//y is',x//y)

x//y is 2

7. Exponent Operator

x=50

y=4

print('x\*\*y is',x\*\*y)

x\*\*y is 6250000

**Comparison Operators**

1. EQUAL comparison Operator

x=50

y=4

print('x==y is',x==y)

x==y is False

2. NOT EQUAL comparison operator

x=50

y=4

print('x!=y is',x!=y)

x!=y is True

3. GREATER THAN operator

x=50

y=4

print('x>y is',x>y)

x>y is True

4. LESS THAN operator

x=50

y=4

print('x<y is',x<y)

x<y is False

5. GREATER THAN OR EQUAL TO operator

x=50

y=4

print('x>=y is',x>=y)

x>=y is True

6. LESS THAN OR EQUAL TO operator

x=50

y=4

print('x<=y is',x<=y)

x<=y is False

**Assignment operators**

1. Add AND operator

x=10

x+=2

print(x)

12

2. Subtract AND operator

x=10

x-=2

print(x)

8

3. Mutiply AND operator

x=10

x\*=2

print(x)

20

4. Divide AND operator

x=10

x/=2

print(x)

5.0

5. Modulus AND operator

x=10

x%=2

print(x)

0

6. Exponent AND operator

x=10

x\*\*=10

print(x)

10000000000

7. Integer Division AND operator

x=10

x//=3

print(x)

3

**Logical operators**

1. Logical AND operator

x=10

y=20

print('x and y is',x and y)

x and y is 20

2. Logical OR operator

x=10

y=20

print('x or y is',x or y)

x or y is 10

3. Logical NOT operator

x=10

y=False

print(not x)

False

**Bitwise operators**

1. Bitwise AND

a=10

b=4

print(a & b)

0

2. Bitwise OR

a=10

b=4

print(a | b)

14

3. Bitwise XOR

a=100

b=40

print(a ^ b)

76

4. One’s Complement operator

a=100

b=40

print(~a)

-101

5. Left Shift operator

a=10

print(a<<2)

40

6. Right Shift operator

a=10

print(a>>2)

2

**MEMBERSHIP OPERATORS**

1. IN / NOT IN Operator

s='hello python'

print('e' in s)

True

s='hello python'

print('elo' not in s)

True

**IDENTITY OPERATORS**

1. IS operator

x=6

if(type(x) is int):

print('True')

else:

print('False')

True

2. IS NOT operator

x=4.5

if(type(x) is not int):

print('True')

else:

print('False')

True

**22. Create list of elements and slice and dice it**

l=[10,'hello',2.85,56.5,'python']

l[0:3]

[10, 'hello', 2.85]

l=[10,'hello',2.85,56.5,'python']

l[3:3]

[]

l=[10,'hello',2.85,56.5,'python']

l[3:5]

[56.5, 'python']

l=[10,'hello',2.85,56.5,'python']

l[:5]

[10, 'hello', 2.85, 56.5, 'python']

l=[10,'hello',2.85,56.5,'python']

l[1:]

['hello', 2.85, 56.5, 'python']

l=[10,'hello',2.85,56.5,'python']

l[-1:]

['python']

l=[10,'hello',2.85,56.5,'python']

l[-3:-1]

[2.85, 56.5]

**23. Using while loop accept numbers until sum of numbers is less than 100**

sum=0

num=[]

while True:

inp=int(input('enter values'))

sum=sum+inp

if(sum>=100):

break

num.append(inp)

print(sum)

print(num)

enter values20

enter values40

enter values30

enter values40

130

[20, 40, 30]

**24. Write a python program Read & write Excel files**

import pandas as pd

df=pd.read\_excel("D://samplesuperstore.xls","People")

df

Person Region

0 Anna Andreadi West

1 Chuck Magee East

2 Kelly Williams Central

3 Cassandra Brandow South

Writing excel files

df.to\_excel("D://hima.xls",sheet\_name="example.xls")

df

**25. Write a python program to scrape reviews from a commercial web site**

from bs4 import BeautifulSoup as bs

import requests

link='https://www.amazon.com/Logitech-C920x-Pro-HD-Webcam/dp/B085TFF7M1/ref=sr\_1\_2?dchild=1&fst=as%3Aoff&pf\_rd\_i=16225007011&pf\_rd\_m=ATVPDKIKX0DER&pf\_rd\_p=74069509-93ef-4a3c-8dca-a9e3fa773a64&pf\_rd\_r=5FT7CMNENHG4W49BP8ZK&pf\_rd\_s=merchandised-search-4&pf\_rd\_t=101&qid=1487012920&rnid=16225007011&s=computers-intl-ship&sr=1-2'

page=requests.get(link)

page

<Response [503]>

page.content

b'<!--\n To discuss automated access to Amazon data please contact api-services-support@amazon.com.\n For information about migrating to our APIs refer to our Marketplace APIs at <https://developer.amazonservices.com/ref=rm_5_sv,> or our Product Advertising API at <https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html/ref=rm_5_ac> for advertising use cases.\n-->\n<!doctype html>\n<html>\n<head>\n <meta charset="utf-8">\n <meta http-equiv="x-ua-compatible" content="ie=edge">\n <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">\n <title>Sorry! Something went wrong!</title>\n <style>\n html, body {\n padding: 0;\n margin: 0\n }\n\n img {\n border: 0\n }\n\n #a {\n background: #232f3e;\n padding: 11px 11px 11px 192px\n }\n\n #b {\n position: absolute;\n left: 22px;\n top: 12px\n }\n\n #c {\n position: relative;\n max-width: 800px;\n padding: 0 40px 0 0\n }\n\n #e, #f {\n height: 35px;\n border: 0;\n font-size: 1em\n }\n\n #e {\n width: 100%;\n margin: 0;\n padding: 0 10px;\n border-radius: 4px 0 0 4px\n }\n\n #f {\n cursor: pointer;\n background: #febd69;\n font-weight: bold;\n border-radius: 0 4px 4px 0;\n -webkit-appearance: none;\n position: absolute;\n top: 0;\n right: 0;\n padding: 0 12px\n }\n\n @media (max-width: 500px) {\n #a {\n padding: 55px 10px 10px\n }\n\n #b {\n left: 6px\n }\n }\n\n #g {\n text-align: center;\n margin: 30px 0\n }\n\n #g img {\n max-width: 90%\n }\n\n #d {\n display: none\n }\n\n #d[src] {\n display: inline\n }\n </style>\n</head>\n<body>\n <a href="/ref=cs\_503\_logo"><img id="b" src="https://images-na.ssl-images-amazon.com/images/G/01/error/logo.\_TTD\_.png" alt="Amazon.com"></a>\n <form id="a" accept-charset="utf-8" action="/s" method="GET" role="search">\n <div id="c">\n <input id="e" name="field-keywords" placeholder="Search">\n <input name="ref" type="hidden" value="cs\_503\_search">\n <input id="f" type="submit" value="Go">\n </div>\n </form>\n<div id="g">\n <div><a href="/ref=cs\_503\_link"><img src="https://images-na.ssl-images-amazon.com/images/G/01/error/500\_503.png"\n alt="Sorry! Something went wrong on our end. Please go back and try again or go to Amazon\'s home page."></a>\n </div>\n <a href="/dogsofamazon/ref=cs\_503\_d" target="\_blank" rel="noopener noreferrer"><img id="d" alt="Dogs of Amazon"></a>\n <script>document.getElementById("d").src = "https://images-na.ssl-images-amazon.com/images/G/01/error/" + (Math.floor(Math.random() \* 43) + 1) + ".\_TTD\_.jpg";</script>\n</div>\n</body>\n</html>\n'

Another way to display the contents of the page

soup=bs(page.content,'html.parser')

print(soup.prettify())

<!--

To discuss automated access to Amazon data please contact api-services-support@amazon.com.

For information about migrating to our APIs refer to our Marketplace APIs at <https://developer.amazonservices.com/ref=rm_5_sv,> or our Product Advertising API at <https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html/ref=rm_5_ac> for advertising use cases.

-->

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8"/>

<meta content="ie=edge" http-equiv="x-ua-compatible"/>

<meta content="width=device-width, initial-scale=1, shrink-to-fit=no" name="viewport"/>

<title>

Sorry! Something went wrong!

</title>

<style>

html, body {

padding: 0;

margin: 0

}

img {

border: 0

}

#a {

background: #232f3e;

padding: 11px 11px 11px 192px

}

#b {

position: absolute;

left: 22px;

top: 12px

}

#c {

position: relative;

max-width: 800px;

padding: 0 40px 0 0

}

#e, #f {

height: 35px;

border: 0;

font-size: 1em

}

#e {

width: 100%;

margin: 0;

padding: 0 10px;

border-radius: 4px 0 0 4px

}

#f {

cursor: pointer;

background: #febd69;

font-weight: bold;

border-radius: 0 4px 4px 0;

-webkit-appearance: none;

position: absolute;

top: 0;

right: 0;

padding: 0 12px

}

@media (max-width: 500px) {

#a {

padding: 55px 10px 10px

}

#b {

left: 6px

}

}

#g {

text-align: center;

margin: 30px 0

}

#g img {

max-width: 90%

}

#d {

display: none

}

#d[src] {

display: inline

}

</style>

</head>

<body>

<a href="/ref=cs\_503\_logo">

<img alt="Amazon.com" id="b" src="https://images-na.ssl-images-amazon.com/images/G/01/error/logo.\_TTD\_.png"/>

</a>

<form accept-charset="utf-8" action="/s" id="a" method="GET" role="search">

<div id="c">

<input id="e" name="field-keywords" placeholder="Search"/>

<input name="ref" type="hidden" value="cs\_503\_search"/>

<input id="f" type="submit" value="Go"/>

</div>

</form>

<div id="g">

<div>

<a href="/ref=cs\_503\_link">

<img alt="Sorry! Something went wrong on our end. Please go back and try again or go to Amazon's home page." src="https://images-na.ssl-images-amazon.com/images/G/01/error/500\_503.png"/>

</a>

</div>

<a href="/dogsofamazon/ref=cs\_503\_d" rel="noopener noreferrer" target="\_blank">

<img alt="Dogs of Amazon" id="d"/>

</a>

<script>

document.getElementById("d").src = "https://images-na.ssl-images-amazon.com/images/G/01/error/" + (Math.floor(Math.random() \* 43) + 1) + ".\_TTD\_.jpg";

</script>

</div>

</body>

</html>

**To find out names of the reviewers of that particular item**

names=soup.find\_all('span',class\_='a-profile-name')

names

[]

#As there are no reviewers for the selected item we got empty list symbol

**To print only the names of the reviewers of the selected item**

cust\_name=[]

for i in range(0,len(names)):

cust\_name.append(names[i].get\_text())

cust\_name

[]

#As there are no reviewers for the selected item we got empty list symbol

**To print the title name of the product given by the reviewer**

title=soup.find\_all('span',class\_='review\_title')

title

[]

#As there is no title for the selected product we got an empty list symbol

**Another way To print the title name of the product given by the reviewer**

title=soup.find\_all('a',class\_='review-title-content')

title

[]

#As there is no title for the selected product we got an empty list symbol

**To print the review of the product**

review\_title=[]

for i in range(0,len(title)):

review\_title.append(title[i].get\_text())

review\_title

**To print the rating of the product**

rating=soup.find\_all('i',class\_='review\_rating')

rating

**Another way To print the rating of the product**

rate=[]

for i in range(0,len(rating)):

rate.append(rating[i].get\_text())

rate

**To print the review of the product**

review=soup.find\_all("span",{"data-hook":"review-body"})

review

**Another way To print the review of the product**

review\_content=[]

for i in range(0,len(review)):

review\_content.append(review[i].get\_text())

review\_content

**To print the obtained results as a Data Frame**

import pandas as pd

df=pd.DataFrame()

df['Customer Name']=cust\_name

df['Review Title']=review\_title

df['Ratings']=rate

df['Reviews']=review\_content

df

**Inorder to convert data frame to CSV file**

df.to\_csv(r'D:\\reviews.csv',index=True)

**26. Create a 3x3 matrix with values ranging from 2 to 10 using numpy**

from numpy import \*

arr1=array([[2,3,4],[5,6,7],[8,9,10]])

print(arr1)

[[ 2 3 4]

[ 5 6 7]

[ 8 9 10]]

**27. Write a Python program to convert a list of numeric value into a one-dimensional NumPy array**

import numpy as np

list=[8,9,10,1,2,3,4]

x=np.array(list)

print(x)

[ 8 9 10 1 2 3 4]

**17. Use Crime dataset from LMS**

**do the plottings like plottings like histogram, boxplot, scatterplot, barplot, piechart,dot chart.**

Loading dataset using pandas

import pandas as pb

pb.read\_csv('crime\_data.csv')

Unnamed: 0 Murder Assault UrbanPop Rape

0 Alabama 13.2 236 58 21.2

1 Alaska 10.0 263 48 44.5

2 Arizona 8.1 294 80 31.0

3 Arkansas 8.8 190 50 19.5

4 California 9.0 276 91 40.6

5 Colorado 7.9 204 78 38.7

6 Connecticut 3.3 110 77 11.1

7 Delaware 5.9 238 72 15.8

8 Florida 15.4 335 80 31.9

9 Georgia 17.4 211 60 25.8

10 Hawaii 5.3 46 83 20.2

11 Idaho 2.6 120 54 14.2

12 Illinois 10.4 249 83 24.0

13 Indiana 7.2 113 65 21.0

14 Iowa 2.2 56 57 11.3

15 Kansas 6.0 115 66 18.0

16 Kentucky 9.7 109 52 16.3

17 Louisiana 15.4 249 66 22.2

18 Maine 2.1 83 51 7.8

19 Maryland 11.3 300 67 27.8

20 Massachusetts 4.4 149 85 16.3

21 Michigan 12.1 255 74 35.1

22 Minnesota 2.7 72 66 14.9

23 Mississippi 16.1 259 44 17.1

24 Missouri 9.0 178 70 28.2

25 Montana 6.0 109 53 16.4

26 Nebraska 4.3 102 62 16.5

27 Nevada 12.2 252 81 46.0

28 New Hampshire 2.1 57 56 9.5

29 New Jersey 7.4 159 89 18.8

30 New Mexico 11.4 285 70 32.1

31 New York 11.1 254 86 26.1

32 North Carolina13.0 337 45 16.1

33 North Dakota 0.8 45 44 7.3

34 Ohio 7.3 120 75 21.4

35 Oklahoma 6.6 151 68 20.0

36 Oregon 4.9 159 67 29.3

37 Pennsylvania 6.3 106 72 14.9

38 Rhode Island 3.4 174 87 8.3

39 South Carolina14.4 279 48 22.5

40 South Dakota 3.8 86 45 12.8

41 Tennessee 13.2 188 59 26.9

42 Texas 12.7 201 80 25.5

43 Utah 3.2 120 80 22.9

44 Vermont 2.2 48 32 11.2

45 Virginia 8.5 156 63 20.7

46 Washington 4.0 145 73 26.2

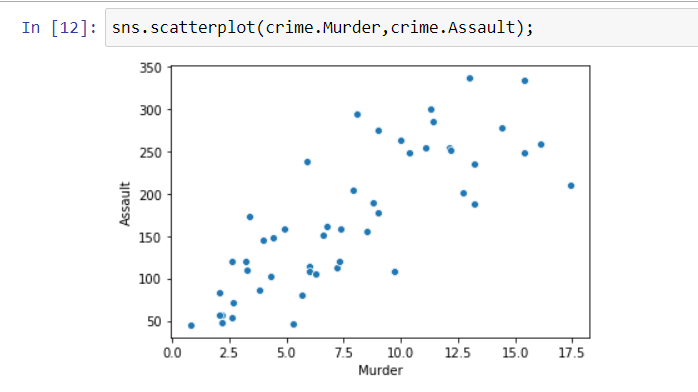
47 West Virginia 5.7 81 39 9.3

48 Wisconsin 2.6 53 66 10.8

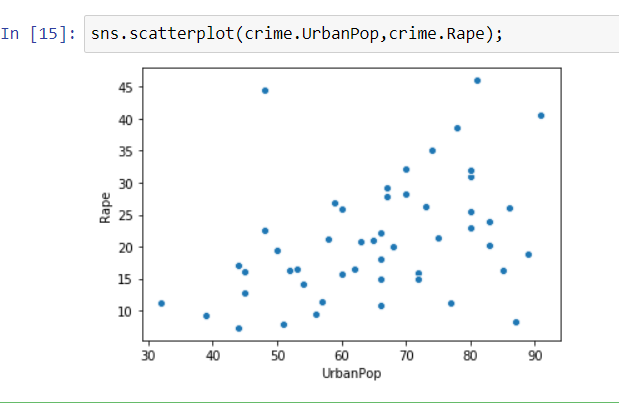
49 Wyoming 6.8 161 60 15.6

**Plotting scatterplot for the columns Murder and Assault**

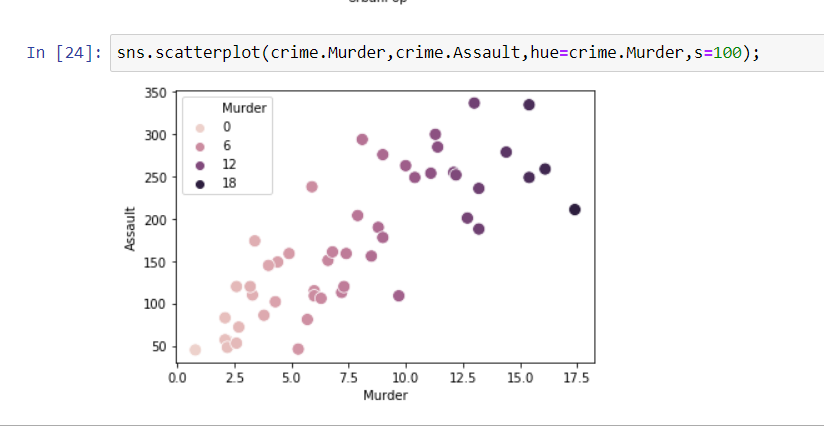
import seaborn as sns

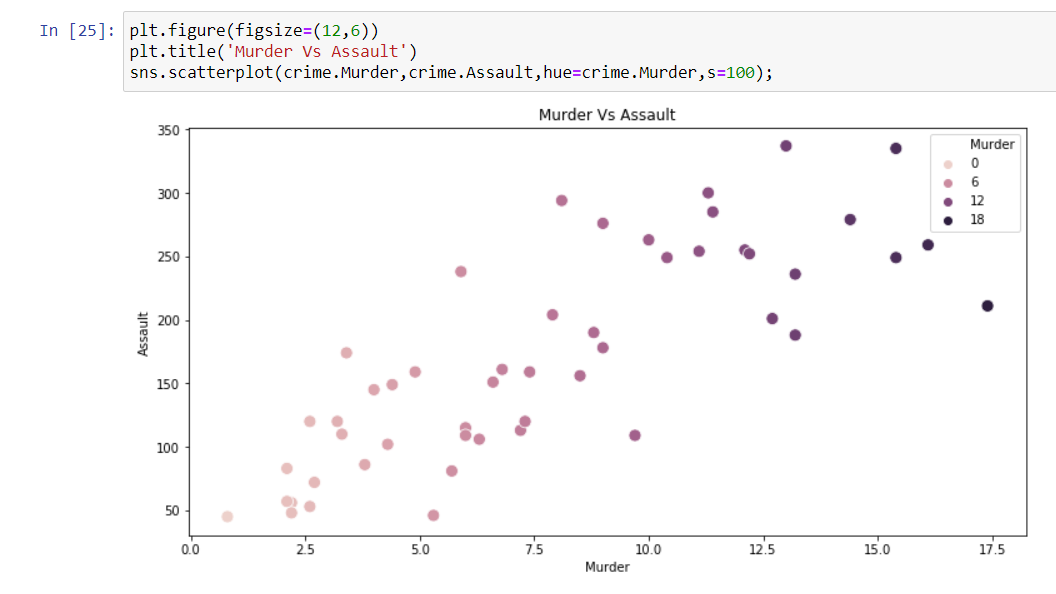


**Plotting scatterplot for the columns UrbanPop and Rape**

****

**Adding color and hue based on Murder**

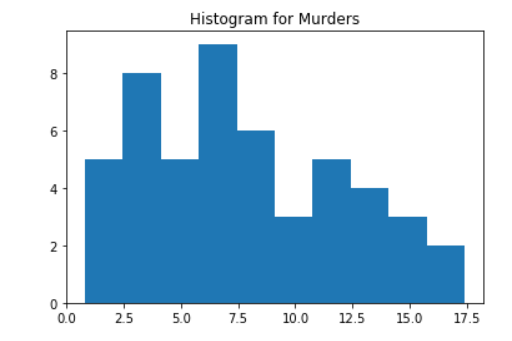
****

****

**Ploting Histogram for Murders**

plt.title('Histogram for Murders')

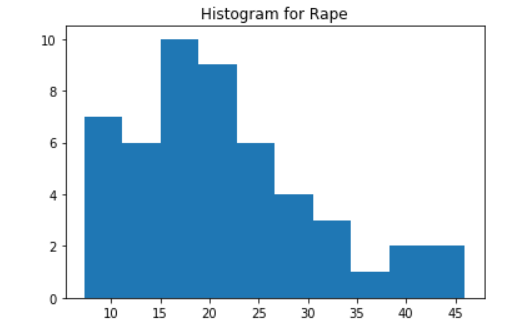
plt.hist(crime.Murder);



**Ploting Histogram for Rape**

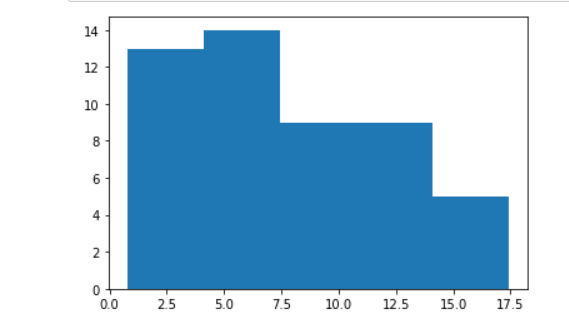
plt.title('Histogram for Rape')

plt.hist(crime.Rape);



**Specifying the number of bins**

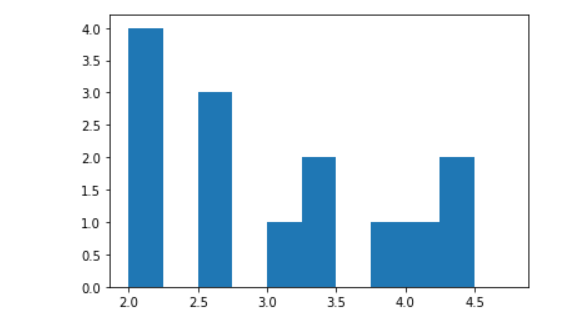
plt.hist(crime.Murder,bins=5);



**#specifying the boundaries of each bin**

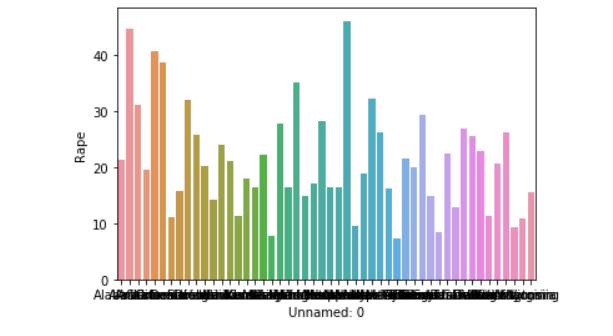
import numpy as np

plt.hist(crime.Murder,bins=np.arange(2,5,0.25));



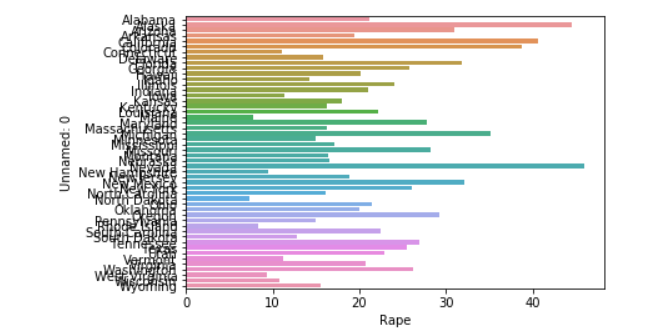
**Barplot between 2 columns**

sns.barplot('Unnamed: 0','Rape',data=crime);



**We make bars horizontal by switching the axes**

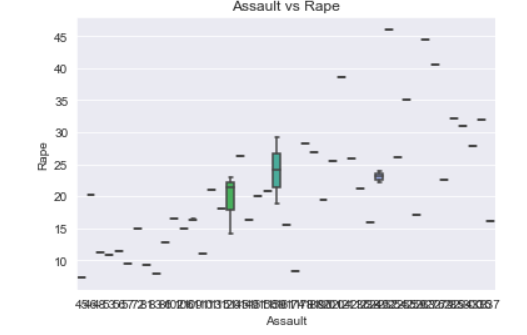
sns.barplot('Rape','Unnamed: 0',data=crime);



**Boxplot for Assault Vs Rape**

plt.title('Assault vs Rape')

sns.boxplot(crime.Assault,crime.Rape);



**Pie chart between Location and number of rapes**

import matplotlib.pyplot as plt

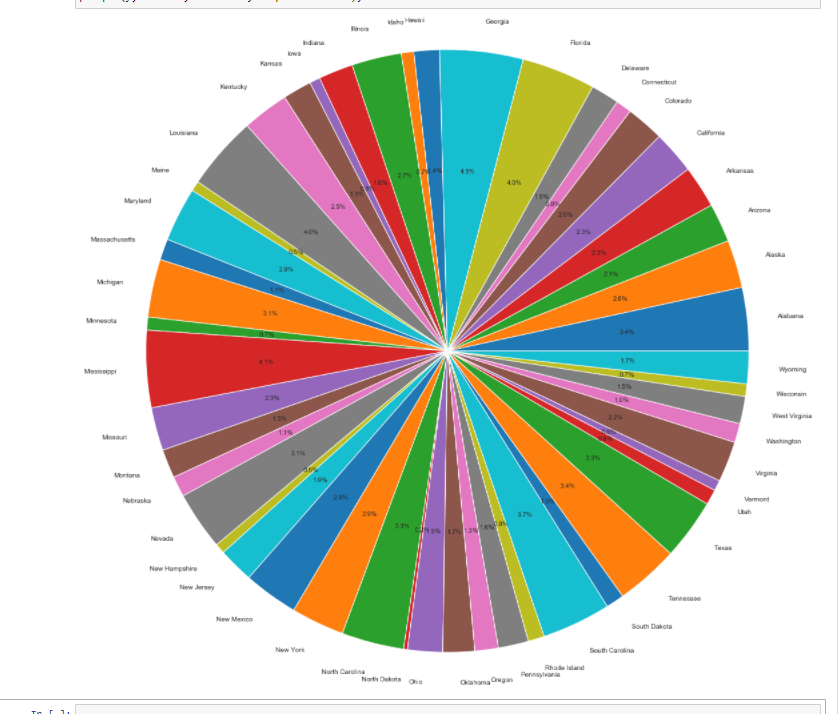
import pandas as pd

df=pd.read\_csv('crime\_data.csv')

x=df['Unnamed: 0']

y=df['Murder']

plt.pie(y,labels=x,radius=5.2,autopct='%2.1f%%');



**Dot plot between Population Vs Rapes**

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

data=pd.read\_csv('crime\_data.csv')

x=data.UrbanPop

y=data.Rape

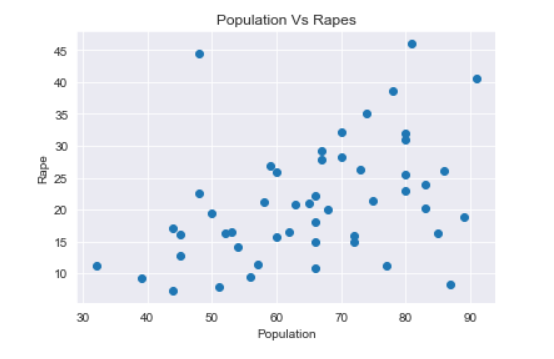
plt.scatter(x,y)

plt.xlabel('Population')

plt.ylabel('Rape')

plt.title('Population Vs Rapes')

plt.show();



**18. use mtcars dataset from LMS**

**A) delete/ drop rows-10 to 15 of all columns**

mtcars=pd.read\_csv('mtcars.csv',index\_col='mpg')

mtcars.drop([21,21,22.8,21.4,18.7,18.1,14.3,24.4,22.8,19.2,17.8,16.4,17.3,15.2],inplace=True)

mtcars

cyl disp hp drat wt qsec vs am gear carb

mpg

10.4 8 472.0 205 2.93 5.250 17.98 0 0 3 4

10.4 8 460.0 215 3.00 5.424 17.82 0 0 3 4

14.7 8 440.0 230 3.23 5.345 17.42 0 0 3 4

32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1

30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2

33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1

21.5 4 120.1 97 3.70 2.465 20.01 1 0 3 1

15.5 8 318.0 150 2.76 3.520 16.87 0 0 3 2

13.3 8 350.0 245 3.73 3.840 15.41 0 0 3 4

27.3 4 79.0 66 4.08 1.935 18.90 1 1 4 1

26.0 4 120.3 91 4.43 2.140 16.70 0 1 5 2

30.4 4 95.1 113 3.77 1.513 16.90 1 1 5 2

15.8 8 351.0 264 4.22 3.170 14.50 0 1 5 4

19.7 6 145.0 175 3.62 2.770 15.50 0 1 5 6

15.0 8 301.0 335 3.54 3.570 14.60 0 1 5 8

**B)drop the qsec column**

del mtcars['qsec']

mtcars

cyl disp hp drat wt vs am gear carb

mpg

10.4 8 472.0 205 2.93 5.250 0 0 3 4

10.4 8 460.0 215 3.00 5.424 0 0 3 4

14.7 8 440.0 230 3.23 5.345 0 0 3 4

32.4 4 78.7 66 4.08 2.200 1 1 4 1

30.4 4 75.7 52 4.93 1.615 1 1 4 2

33.9 4 71.1 65 4.22 1.835 1 1 4 1

21.5 4 120.1 97 3.70 2.465 1 0 3 1

15.5 8 318.0 150 2.76 3.520 0 0 3 2

13.3 8 350.0 245 3.73 3.840 0 0 3 4

27.3 4 79.0 66 4.08 1.935 1 1 4 1

26.0 4 120.3 91 4.43 2.140 0 1 5 2

30.4 4 95.1 113 3.77 1.513 1 1 5 2

15.8 8 351.0 264 4.22 3.170 0 1 5 4

19.7 6 145.0 175 3.62 2.770 0 1 5 6

15.0 8 301.0 335 3.54 3.570 0 1 5 8

**19. Use Bank Dataset from LMS**

bankfull=pd.read\_csv("bank-full.csv")

bankfull

|  | **age;"job";"marital";"education";"default";"balance";"housing";"loan";"contact";"day";"month";"duration";"campaign";"pdays";"previous";"poutcome";"y"** |
| --- | --- |
| **0** | 58;"management";"married";"tertiary";"no";2143... |
| **1** | 44;"technician";"single";"secondary";"no";29;"... |
| **2** | 33;"entrepreneur";"married";"secondary";"no";2... |
| **3** | 47;"blue-collar";"married";"unknown";"no";1506... |
| **4** | 33;"unknown";"single";"unknown";"no";1;"no";"n... |
| **...** | ... |
| **45206** | 51;"technician";"married";"tertiary";"no";825;... |
| **45207** | 71;"retired";"divorced";"primary";"no";1729;"n... |
| **45208** | 72;"retired";"married";"secondary";"no";5715;"... |
| **45209** | 57;"blue-collar";"married";"secondary";"no";66... |
| **45210** | 37;"entrepreneur";"married";"secondary";"no";2... |

bankfull=pd.read\_csv('bank-full.csv',sep=";")

bankfull

Out[123]:

|  | **age** | **job** | **marital** | **education** | **default** | **balance** | **housing** | **loan** | **contact** | **day** | **month** | **duration** | **campaign** | **pdays** | **previous** | **poutcome** | **y** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 58 | management | married | tertiary | no | 2143 | yes | no | unknown | 5 | may | 261 | 1 | -1 | 0 | unknown | no |
| **1** | 44 | technician | single | secondary | no | 29 | yes | no | unknown | 5 | may | 151 | 1 | -1 | 0 | unknown | no |
| **2** | 33 | entrepreneur | married | secondary | no | 2 | yes | yes | unknown | 5 | may | 76 | 1 | -1 | 0 | unknown | no |
| **3** | 47 | blue-collar | married | unknown | no | 1506 | yes | no | unknown | 5 | may | 92 | 1 | -1 | 0 | unknown | no |
| **4** | 33 | unknown | single | unknown | no | 1 | no | no | unknown | 5 | may | 198 | 1 | -1 | 0 | unknown | no |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **45206** | 51 | technician | married | tertiary | no | 825 | no | no | cellular | 17 | nov | 977 | 3 | -1 | 0 | unknown | yes |
| **45207** | 71 | retired | divorced | primary | no | 1729 | no | no | cellular | 17 | nov | 456 | 2 | -1 | 0 | unknown | yes |
| **45208** | 72 | retired | married | secondary | no | 5715 | no | no | cellular | 17 | nov | 1127 | 5 | 184 | 3 | success | yes |
| **45209** | 57 | blue-collar | married | secondary | no | 668 | no | no | telephone | 17 | nov | 508 | 4 | -1 | 0 | unknown | no |
| **45210** | 37 | entrepreneur | married | secondary | no | 2971 | no | no | cellular | 17 | nov | 361 | 2 | 188 | 11 | other | no |

45211 rows × 17 columns

**B) rename all the column names DF**

bankfull.rename(columns={'age':'aaa','job':'bbb','marital':'ccc','education':'ddd','default':'eee','balance':'fff','housing':'ggg','loan':'hhh','contact':'iii','day':'jjj','month':'kkk','duration':'lll', 'campaign':'mmm','pdays':'nnn','previous':'ooo','poutcome':'ppp','y':'qqq'},inplace=True)

bankfull

| **aaa** | **bbb** | **ccc** | **ddd** | **eee** | **fff** | **ggg** | **hhh** | **iii** | **jjj** | **kkk** | **lll** | **mmm** | **nnn** | **ooo** | **ppp** | **qqq** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 58 | management | married | tertiary | no | 2143 | yes | no | unknown | 5 | may | 261 | 1 | -1 | 0 | unknown | no |
| **1** | 44 | technician | single | secondary | no | 29 | yes | no | unknown | 5 | may | 151 | 1 | -1 | 0 | unknown | no |
| **2** | 33 | entrepreneur | married | secondary | no | 2 | yes | yes | unknown | 5 | may | 76 | 1 | -1 | 0 | unknown | no |
| **3** | 47 | blue-collar | married | unknown | no | 1506 | yes | no | unknown | 5 | may | 92 | 1 | -1 | 0 | unknown | no |
| **4** | 33 | unknown | single | unknown | no | 1 | no | no | unknown | 5 | may | 198 | 1 | -1 | 0 | unknown | no |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **45206** | 51 | technician | married | tertiary | no | 825 | no | no | cellular | 17 | nov | 977 | 3 | -1 | 0 | unknown | yes |
| **45207** | 71 | retired | divorced | primary | no | 1729 | no | no | cellular | 17 | nov | 456 | 2 | -1 | 0 | unknown | yes |
| **45208** | 72 | retired | married | secondary | no | 5715 | no | no | cellular | 17 | nov | 1127 | 5 | 184 | 3 | success | yes |
| **45209** | 57 | blue-collar | married | secondary | no | 668 | no | no | telephone | 17 | nov | 508 | 4 | -1 | 0 | unknown | no |
| **45210** | 37 | entrepreneur | married | secondary | no | 2971 | no | no | cellular | 17 | nov | 361 | 2 | 188 | 11 | other | no |

45211 rows × 17 columns

**C) Rename only one specific column in DF**

bankfull.rename(columns={'aaa':'age','bbb':'job'},inplace=True)

bankfull

|  | **age** | **job** | **ccc** | **ddd** | **eee** | **fff** | **ggg** | **hhh** | **iii** | **jjj** | **kkk** | **lll** | **mmm** | **nnn** | **ooo** | **ppp** | **qqq** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 58 | management | married | tertiary | no | 2143 | yes | no | unknown | 5 | may | 261 | 1 | -1 | 0 | unknown | no |
| **1** | 44 | technician | single | secondary | no | 29 | yes | no | unknown | 5 | may | 151 | 1 | -1 | 0 | unknown | no |
| **2** | 33 | entrepreneur | married | secondary | no | 2 | yes | yes | unknown | 5 | may | 76 | 1 | -1 | 0 | unknown | no |
| **3** | 47 | blue-collar | married | unknown | no | 1506 | yes | no | unknown | 5 | may | 92 | 1 | -1 | 0 | unknown | no |
| **4** | 33 | unknown | single | unknown | no | 1 | no | no | unknown | 5 | may | 198 | 1 | -1 | 0 | unknown | no |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **45206** | 51 | technician | married | tertiary | no | 825 | no | no | cellular | 17 | nov | 977 | 3 | -1 | 0 | unknown | yes |
| **45207** | 71 | retired | divorced | primary | no | 1729 | no | no | cellular | 17 | nov | 456 | 2 | -1 | 0 | unknown | yes |
| **45208** | 72 | retired | married | secondary | no | 5715 | no | no | cellular | 17 | nov | 1127 | 5 | 184 | 3 | success | yes |
| **45209** | 57 | blue-collar | married | secondary | no | 668 | no | no | telephone | 17 | nov | 508 | 4 | -1 | 0 | unknown | no |
| **45210** | 37 | entrepreneur | married | secondary | no | 2971 | no | no | cellular | 17 | nov | 361 | 2 | 188 | 11 | other | no |

45211 rows × 17 columns

**18. use mtcars dataset from LMS**

**C)write the forloop to get value\_counts of all columns**

pd.value\_counts(mtcars['mpg'])

10.4 2

21.4 2

30.4 2

22.8 2

15.2 2

19.2 2

21.0 2

18.1 1

32.4 1

14.7 1

21.5 1

15.5 1

15.8 1

17.8 1

27.3 1

26.0 1

19.7 1

15.0 1

18.7 1

16.4 1

33.9 1

24.4 1

14.3 1

17.3 1

13.3 1

Name: mpg, dtype: int64

**Using for loop to find the value\_counts of each column**

for col in crime.columns:

display(crime[col].value\_counts())

Indiana 1

Minnesota 1

Arizona 1

Utah 1

Colorado 1

Maryland 1

Delaware 1

Tennessee 1

Texas 1

Oklahoma 1

South Dakota 1

Oregon 1

New Mexico 1

Ohio 1

South Carolina 1

Kansas 1

Missouri 1

Mississippi 1

Maine 1

Idaho 1

New Jersey 1

West Virginia 1

Connecticut 1

Wyoming 1

Wisconsin 1

Montana 1

Arkansas 1

Georgia 1

Kentucky 1

Pennsylvania 1

Alabama 1

Nevada 1

North Dakota 1

New York 1

Rhode Island 1

Illinois 1

Massachusetts 1

Louisiana 1

Virginia 1

New Hampshire 1

Hawaii 1

Vermont 1

Florida 1

North Carolina 1

Michigan 1

Nebraska 1

Washington 1

Iowa 1

California 1

Alaska 1

Name: Unnamed: 0, dtype: int64

15.4 2

9.0 2

2.6 2

6.0 2

2.1 2

13.2 2

2.2 2

0.8 1

3.4 1

8.8 1

4.4 1

7.2 1

6.3 1

16.1 1

5.9 1

4.9 1

7.3 1

4.0 1

14.4 1

10.4 1

8.5 1

13.0 1

3.3 1

11.3 1

17.4 1

2.7 1

4.3 1

3.2 1

12.1 1

12.7 1

7.9 1

3.8 1

5.3 1

6.6 1

6.8 1

11.4 1

8.1 1

7.4 1

12.2 1

5.7 1

9.7 1

11.1 1

10.0 1

Name: Murder, dtype: int64

120 3

249 2

159 2

109 2

56 1

211 1

285 1

254 1

238 1

45 1

57 1

86 1

149 1

276 1

337 1

102 1

335 1

255 1

204 1

201 1

72 1

263 1

259 1

83 1

145 1

188 1

190 1

161 1

252 1

300 1

156 1

115 1

178 1

113 1

48 1

110 1

236 1

106 1

81 1

279 1

174 1

294 1

53 1

46 1

151 1

Name: Assault, dtype: int64

66 4

80 4

45 2

60 2

67 2

83 2

70 2

72 2

44 2

48 2

63 1

81 1

78 1

75 1

77 1

86 1

74 1

73 1

68 1

85 1

89 1

87 1

62 1

91 1

32 1

39 1

50 1

51 1

52 1

53 1

54 1

56 1

57 1

58 1

59 1

65 1

Name: UrbanPop, dtype: int64

14.9 2

16.3 2

12.8 1

11.3 1

14.2 1

20.7 1

38.7 1

31.9 1

18.8 1

25.5 1

22.5 1

7.3 1

20.0 1

7.8 1

9.5 1

46.0 1

28.2 1

16.5 1

18.0 1

21.0 1

24.0 1

19.5 1

31.0 1

9.3 1

16.4 1

26.2 1

29.3 1

22.2 1

25.8 1

10.8 1

20.2 1

11.2 1

15.8 1

22.9 1

11.1 1

16.1 1

35.1 1

26.1 1

40.6 1

21.2 1

17.1 1

15.6 1

8.3 1

26.9 1

32.1 1

27.8 1

21.4 1

44.5 1

Name: Rape, dtype: int64

**Count for the entire dataframe**

mtcars.apply(pd.value\_counts)

|  | **mpg** | **cyl** | **disp** | **hp** | **drat** | **wt** | **qsec** | **vs** | **am** | **gear** | **carb** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0.000** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 18.0 | 19.0 | NaN | NaN |
| **1.000** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 14.0 | 13.0 | NaN | 7.0 |
| **1.513** | NaN | NaN | NaN | NaN | NaN | 1.0 | NaN | NaN | NaN | NaN | NaN |
| **1.615** | NaN | NaN | NaN | NaN | NaN | 1.0 | NaN | NaN | NaN | NaN | NaN |
| **1.835** | NaN | NaN | NaN | NaN | NaN | 1.0 | NaN | NaN | NaN | NaN | NaN |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **360.000** | NaN | NaN | 2.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **400.000** | NaN | NaN | 1.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **440.000** | NaN | NaN | 1.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **460.000** | NaN | NaN | 1.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **472.000** | NaN | NaN | 1.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

158 rows × 11 columns

**19. Use Bank Dataset from LMS**

**A)change all the categorical columns into numerical by creating Dummies and using label encoder.**

import pandas as pd

import numpy as np

df=pd.read\_csv('bank-full.csv', sep = ';')

df

|  | **age** | **job** | **marital** | **education** | **default** | **balance** | **housing** | **loan** | **contact** | **day** | **month** | **duration** | **campaign** | **pdays** | **previous** | **poutcome** | **y** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 58 | management | married | tertiary | no | 2143 | yes | no | unknown | 5 | may | 261 | 1 | -1 | 0 | unknown | no |
| **1** | 44 | technician | single | secondary | no | 29 | yes | no | unknown | 5 | may | 151 | 1 | -1 | 0 | unknown | no |
| **2** | 33 | entrepreneur | married | secondary | no | 2 | yes | yes | unknown | 5 | may | 76 | 1 | -1 | 0 | unknown | no |
| **3** | 47 | blue-collar | married | unknown | no | 1506 | yes | no | unknown | 5 | may | 92 | 1 | -1 | 0 | unknown | no |
| **4** | 33 | unknown | single | unknown | no | 1 | no | no | unknown | 5 | may | 198 | 1 | -1 | 0 | unknown | no |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **45206** | 51 | technician | married | tertiary | no | 825 | no | no | cellular | 17 | nov | 977 | 3 | -1 | 0 | unknown | yes |
| **45207** | 71 | retired | divorced | primary | no | 1729 | no | no | cellular | 17 | nov | 456 | 2 | -1 | 0 | unknown | yes |
| **45208** | 72 | retired | married | secondary | no | 5715 | no | no | cellular | 17 | nov | 1127 | 5 | 184 | 3 | success | yes |
| **45209** | 57 | blue-collar | married | secondary | no | 668 | no | no | telephone | 17 | nov | 508 | 4 | -1 | 0 | unknown | no |
| **45210** | 37 | entrepreneur | married | secondary | no | 2971 | no | no | cellular | 17 | nov | 361 | 2 | 188 | 11 | other | no |

45211 rows × 17 columns

df.head()

Out[27]:

|  | **age** | **job** | **marital** | **education** | **default** | **balance** | **housing** | **loan** | **contact** | **day** | **month** | **duration** | **campaign** | **pdays** | **previous** | **poutcome** | **y** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 58 | management | married | tertiary | no | 2143 | yes | no | unknown | 5 | may | 261 | 1 | -1 | 0 | unknown | no |
| **1** | 44 | technician | single | secondary | no | 29 | yes | no | unknown | 5 | may | 151 | 1 | -1 | 0 | unknown | no |
| **2** | 33 | entrepreneur | married | secondary | no | 2 | yes | yes | unknown | 5 | may | 76 | 1 | -1 | 0 | unknown | no |
| **3** | 47 | blue-collar | married | unknown | no | 1506 | yes | no | unknown | 5 | may | 92 | 1 | -1 | 0 | unknown | no |
| **4** | 33 | unknown | single | unknown | no | 1 | no | no | unknown | 5 | may | 198 | 1 | -1 | 0 | unknown | no |

from sklearn import preprocessing

le = preprocessing.LabelEncoder()

df['job']=le.fit\_transform(df['job'])

df['job'].head()

0 4

1 9

2 2

3 1

4 11

Name: job, dtype: int32

df['job'].unique()

array([ 4, 9, 2, 1, 11, 5, 0, 7, 6, 10, 3, 8])

df['marital'].unique()

array(['married', 'single', 'divorced'], dtype=object)

df['education'].unique()

array(['tertiary', 'secondary', 'unknown', 'primary'], dtype=object)

df[['marital','education']].head()

marital education

0 married tertiary

1 single secondary

2 married secondary

3 married unknown

4 single unknown

df['marital']=le.fit\_transform(df['marital'])

df['education']=le.fit\_transform(df['education'])

df['default']=le.fit\_transform(df['default'])

df['housing']=le.fit\_transform(df['housing'])

df['contact']=le.fit\_transform(df['contact'])

df['loan']=le.fit\_transform(df['loan'])

df['month']=le.fit\_transform(df['month'])

df['poutcome']=le.fit\_transform(df['poutcome'])

df['y']=le.fit\_transform(df['y'])

df.dtypes

age int64

job int32

marital int32

education int32

default int32

balance int64

housing int32

loan int32

contact int32

day int64

month int32

duration int64

campaign int64

pdays int64

previous int64

poutcome int32

y int32

dtype: object

df['y'].unique()

array([0, 1])

df['marital'].unique()

array([1, 2, 0])

df.head()

|  | **age** | **job** | **marital** | **education** | **default** | **balance** | **housing** | **loan** | **contact** | **day** | **month** | **duration** | **campaign** | **pdays** | **previous** | **poutcome** | **y** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 58 | 4 | 1 | 2 | 0 | 2143 | 1 | 0 | 2 | 5 | 8 | 261 | 1 | -1 | 0 | 3 | 0 |
| **1** | 44 | 9 | 2 | 1 | 0 | 29 | 1 | 0 | 2 | 5 | 8 | 151 | 1 | -1 | 0 | 3 | 0 |
| **2** | 33 | 2 | 1 | 1 | 0 | 2 | 1 | 1 | 2 | 5 | 8 | 76 | 1 | -1 | 0 | 3 | 0 |
| **3** | 47 | 1 | 1 | 3 | 0 | 1506 | 1 | 0 | 2 | 5 | 8 | 92 | 1 | -1 | 0 | 3 | 0 |
| **4** | 33 | 11 | 2 | 3 | 0 | 1 | 0 | 0 | 2 | 5 | 8 | 198 | 1 | -1 | 0 | 3 | 0 |

Df

| **age** | **job** | **marital** | **education** | **default** | **balance** | **housing** | **loan** | **contact** | **day** | **month** | **duration** | **campaign** | **pdays** | **previous** | **poutcome** | **y** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 58 | 4 | 1 | 2 | 0 | 2143 | 1 | 0 | 2 | 5 | 8 | 261 | 1 | -1 | 0 | 3 | 0 |
| **1** | 44 | 9 | 2 | 1 | 0 | 29 | 1 | 0 | 2 | 5 | 8 | 151 | 1 | -1 | 0 | 3 | 0 |
| **2** | 33 | 2 | 1 | 1 | 0 | 2 | 1 | 1 | 2 | 5 | 8 | 76 | 1 | -1 | 0 | 3 | 0 |
| **3** | 47 | 1 | 1 | 3 | 0 | 1506 | 1 | 0 | 2 | 5 | 8 | 92 | 1 | -1 | 0 | 3 | 0 |
| **4** | 33 | 11 | 2 | 3 | 0 | 1 | 0 | 0 | 2 | 5 | 8 | 198 | 1 | -1 | 0 | 3 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **45206** | 51 | 9 | 1 | 2 | 0 | 825 | 0 | 0 | 0 | 17 | 9 | 977 | 3 | -1 | 0 | 3 | 1 |
| **45207** | 71 | 5 | 0 | 0 | 0 | 1729 | 0 | 0 | 0 | 17 | 9 | 456 | 2 | -1 | 0 | 3 | 1 |
| **45208** | 72 | 5 | 1 | 1 | 0 | 5715 | 0 | 0 | 0 | 17 | 9 | 1127 | 5 | 184 | 3 | 2 | 1 |
| **45209** | 57 | 1 | 1 | 1 | 0 | 668 | 0 | 0 | 1 | 17 | 9 | 508 | 4 | -1 | 0 | 3 | 0 |
| **45210** | 37 | 2 | 1 | 1 | 0 | 2971 | 0 | 0 | 0 | 17 | 9 | 361 | 2 | 188 | 11 | 1 | 0 |

45211 rows × 17 columns

| **age** | **job** | **marital** | **education** | **default** | **balance** | **housing** | **loan** | **contact** | **day** | **month** | **duration** | **campaign** | **pdays** | **previous** | **poutcome** | **y** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 58 | 4 | 1 | 2 | 0 | 2143 | 1 | 0 | 2 | 5 | 8 | 261 | 1 | -1 | 0 | 3 | 0 |
| **1** | 44 | 9 | 2 | 1 | 0 | 29 | 1 | 0 | 2 | 5 | 8 | 151 | 1 | -1 | 0 | 3 | 0 |
| **2** | 33 | 2 | 1 | 1 | 0 | 2 | 1 | 1 | 2 | 5 | 8 | 76 | 1 | -1 | 0 | 3 | 0 |
| **3** | 47 | 1 | 1 | 3 | 0 | 1506 | 1 | 0 | 2 | 5 | 8 | 92 | 1 | -1 | 0 | 3 | 0 |
| **4** | 33 | 11 | 2 | 3 | 0 | 1 | 0 | 0 | 2 | 5 | 8 | 198 | 1 | -1 | 0 | 3 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **45206** | 51 | 9 | 1 | 2 | 0 | 825 | 0 | 0 | 0 | 17 | 9 | 977 | 3 | -1 | 0 | 3 | 1 |
| **45207** | 71 | 5 | 0 | 0 | 0 | 1729 | 0 | 0 | 0 | 17 | 9 | 456 | 2 | -1 | 0 | 3 | 1 |
| **45208** | 72 | 5 | 1 | 1 | 0 | 5715 | 0 | 0 | 0 | 17 | 9 | 1127 | 5 | 184 | 3 | 2 | 1 |
| **45209** | 57 | 1 | 1 | 1 | 0 | 668 | 0 | 0 | 1 | 17 | 9 | 508 | 4 | -1 | 0 | 3 | 0 |
| **45210** | 37 | 2 | 1 | 1 | 0 | 2971 | 0 | 0 | 0 | 17 | 9 | 361 | 2 | 188 | 11 | 1 | 0 |

45211 rows × 17 columns

**20. After doing all the changes in bank data(Q19). save the file in your directory in Csv Format.**

**#converting dataframe to csv**

df=df.to\_csv()

print(type(df))

<class 'str'>

**#saving in the working directory**

df.to\_csv("C:\\Users\\ANDREWS HIMA KIRAN\\bank.csv")

print(df)

**17. Use Crime dataset from LMS**

**I) find the aggregations like all moments of business decisions for all columns,value counts.**

**Describing Murder Column**

crime.Murder.describe()

count 50.00000

mean 7.78800

std 4.35551

min 0.80000

25% 4.07500

50% 7.25000

75% 11.25000

max 17.40000

Name: Murder, dtype: float64

**Describing Assault Column**

crime.Assault.describe()

count 50.000000

mean 170.760000

std 83.337661

min 45.000000

25% 109.000000

50% 159.000000

75% 249.000000

max 337.000000

Name: Assault, dtype: float64

**Describing UrbanPop Column**

crime.UrbanPop.describe()

count 50.000000

mean 65.540000

std 14.474763

min 32.000000

25% 54.500000

50% 66.000000

75% 77.750000

max 91.000000

Name: UrbanPop, dtype: float64

**Describing Rape Column**

crime.Rape.describe()

count 50.000000

mean 21.232000

std 9.366385

min 7.300000

25% 15.075000

50% 20.100000

75% 26.175000

max 46.000000

Name: Rape, dtype: float64

**Finding mean of Murder column**

crime.Murder.mean()

7.787999999999999

**Finding mean of Assault column**

crime.Assault.mean()

170.76

**Finding mean of UrbanPop column**

crime.UrbanPop.mean()

65.54

**Finding mean of Rape column**

crime.Rape.mean()

21.231999999999992

**Finding unique values of Murder column**

crime.Murder.unique()

array([13.2, 10. , 8.1, 8.8, 9. , 7.9, 3.3, 5.9, 15.4, 17.4, 5.3,

2.6, 10.4, 7.2, 2.2, 6. , 9.7, 2.1, 11.3, 4.4, 12.1, 2.7,

16.1, 4.3, 12.2, 7.4, 11.4, 11.1, 13. , 0.8, 7.3, 6.6, 4.9,

6.3, 3.4, 14.4, 3.8, 12.7, 3.2, 8.5, 4. , 5.7, 6.8])

**Finding unique values of Assault column**

crime.Assault.unique()

array([236, 263, 294, 190, 276, 204, 110, 238, 335, 211, 46, 120, 249,

113, 56, 115, 109, 83, 300, 149, 255, 72, 259, 178, 102, 252,

57, 159, 285, 254, 337, 45, 151, 106, 174, 279, 86, 188, 201,

48, 156, 145, 81, 53, 161], dtype=int64)

**Finding unique values of UrbanPop column**

crime.UrbanPop.unique()

array([58, 48, 80, 50, 91, 78, 77, 72, 60, 83, 54, 65, 57, 66, 52, 51, 67,

85, 74, 44, 70, 53, 62, 81, 56, 89, 86, 45, 75, 68, 87, 59, 32, 63,

73, 39], dtype=int64)

**Finding unique values of Rape column**

crime.Rape.unique()

array([21.2, 44.5, 31. , 19.5, 40.6, 38.7, 11.1, 15.8, 31.9, 25.8, 20.2,

14.2, 24. , 21. , 11.3, 18. , 16.3, 22.2, 7.8, 27.8, 35.1, 14.9,

17.1, 28.2, 16.4, 16.5, 46. , 9.5, 18.8, 32.1, 26.1, 16.1, 7.3,

21.4, 20. , 29.3, 8.3, 22.5, 12.8, 26.9, 25.5, 22.9, 11.2, 20.7,

26.2, 9.3, 10.8, 15.6])

**Finding value\_count of Rape column**

crime.Rape.value\_counts()

14.9 2

16.3 2

12.8 1

11.3 1

14.2 1

20.7 1

38.7 1

31.9 1

18.8 1

25.5 1

22.5 1

7.3 1

20.0 1

7.8 1

9.5 1

46.0 1

28.2 1

16.5 1

18.0 1

21.0 1

24.0 1

19.5 1

31.0 1

9.3 1

16.4 1

26.2 1

29.3 1

22.2 1

25.8 1

10.8 1

20.2 1

11.2 1

15.8 1

22.9 1

11.1 1

16.1 1

35.1 1

26.1 1

40.6 1

21.2 1

17.1 1

15.6 1

8.3 1

26.9 1

32.1 1

27.8 1

21.4 1

44.5 1

Name: Rape, dtype: int64

**Finding value\_count of Murder column**

crime.Murder.value\_counts()

15.4 2

9.0 2

2.6 2

6.0 2

2.1 2

13.2 2

2.2 2

0.8 1

3.4 1

8.8 1

4.4 1

7.2 1

6.3 1

16.1 1

5.9 1

4.9 1

7.3 1

4.0 1

14.4 1

10.4 1

8.5 1

13.0 1

3.3 1

11.3 1

17.4 1

2.7 1

4.3 1

3.2 1

12.1 1

12.7 1

7.9 1

3.8 1

5.3 1

6.6 1

6.8 1

11.4 1

8.1 1

7.4 1

12.2 1

5.7 1

9.7 1

11.1 1

10.0 1

Name: Murder, dtype: int64

**Finding value\_count of Assault column**

crime.Assault.value\_counts()

120 3

249 2

159 2

109 2

56 1

211 1

285 1

254 1

238 1

45 1

57 1

86 1

149 1

276 1

337 1

102 1

335 1

255 1

204 1

201 1

72 1

263 1

259 1

83 1

145 1

188 1

190 1

161 1

252 1

300 1

156 1

115 1

178 1

113 1

48 1

110 1

236 1

106 1

81 1

279 1

174 1

294 1

53 1

46 1

151 1

Name: Assault, dtype: int64

**Finding value\_count of UrbanPop column**

crime.UrbanPop.value\_counts()

66 4

80 4

45 2

60 2

67 2

83 2

70 2

72 2

44 2

48 2

63 1

81 1

78 1

75 1

77 1

86 1

74 1

73 1

68 1

85 1

89 1

87 1

62 1

91 1

32 1

39 1

50 1

51 1

52 1

53 1

54 1

56 1

57 1

58 1

59 1

65 1

Name: UrbanPop, dtype: int64

**Finding Median of MURDER column**

murderlist=crime["Murder"].tolist()

print(murderlist)

murderlistmedian=numpy.median(murderlist)

print(murderlistmedian)

[13.2, 10.0, 8.1, 8.8, 9.0, 7.9, 3.3, 5.9, 15.4, 17.4, 5.3, 2.6, 10.4, 7.2, 2.2, 6.0, 9.7, 15.4, 2.1, 11.3, 4.4, 12.1, 2.7, 16.1, 9.0, 6.0, 4.3, 12.2, 2.1, 7.4, 11.4, 11.1, 13.0, 0.8, 7.3, 6.6, 4.9, 6.3, 3.4, 14.4, 3.8, 13.2, 12.7, 3.2, 2.2, 8.5, 4.0, 5.7, 2.6, 6.8]

7.25

**Finding Median of ASSAULT column**

assaultlist=crime["Assault"].tolist()

print(assaultlist)

assaultlistmedian=numpy.median(assaultlist)

print(assaultlistmedian)

[236, 263, 294, 190, 276, 204, 110, 238, 335, 211, 46, 120, 249, 113, 56, 115, 109, 249, 83, 300, 149, 255, 72, 259, 178, 109, 102, 252, 57, 159, 285, 254, 337, 45, 120, 151, 159, 106, 174, 279, 86, 188, 201, 120, 48, 156, 145, 81, 53, 161]

159.0

**Finding Median of URBANPOP column**

urbanpoplist=crime["UrbanPop"].tolist()

print(urbanpoplist)

urbanpoplistmedian=numpy.median(urbanpoplist)

print(urbanpoplistmedian)

[58, 48, 80, 50, 91, 78, 77, 72, 80, 60, 83, 54, 83, 65, 57, 66, 52, 66, 51, 67, 85, 74, 66, 44, 70, 53, 62, 81, 56, 89, 70, 86, 45, 44, 75, 68, 67, 72, 87, 48, 45, 59, 80, 80, 32, 63, 73, 39, 66, 60]

66.0

**Finding Median of RAPE column**

rapelist=crime["Rape"].tolist()

print(rapelist)

rapelistmedian=numpy.median(rapelist)

print(rapelistmedian)

[21.2, 44.5, 31.0, 19.5, 40.6, 38.7, 11.1, 15.8, 31.9, 25.8, 20.2, 14.2, 24.0, 21.0, 11.3, 18.0, 16.3, 22.2, 7.8, 27.8, 16.3, 35.1, 14.9, 17.1, 28.2, 16.4, 16.5, 46.0, 9.5, 18.8, 32.1, 26.1, 16.1, 7.3, 21.4, 20.0, 29.3, 14.9, 8.3, 22.5, 12.8, 26.9, 25.5, 22.9, 11.2, 20.7, 26.2, 9.3, 10.8, 15.6]

20.1

**Finding Mode of RAPE column**

rapemode=scipy.stats.mode(rapelist)

print(rapemode)

ModeResult(mode=array([14.9]), count=array([2]))

**Finding Mode of MURDER column**

murdermode=scipy.stats.mode(murderlist)

print(murdermode)

ModeResult(mode=array([2.1]), count=array([2]))

**Finding Mode of URBANPOP column**

urbanpopmode=scipy.stats.mode(urbanpoplist)

print(urbanpopmode)

ModeResult(mode=array([66]), count=array([4]))

**Finding Mode of ASSAULT column**

assaultmode=scipy.stats.mode(assaultlist)

print(assaultmode)

ModeResult(mode=array([120]), count=array([3]))

**To calculate Variance of Crime dataset**

crime.var()

Murder 18.970465

Assault 6945.165714

UrbanPop 209.518776

Rape 87.729159

dtype: float64

**To calculate Skewness and Kurtosis of Crime dataset**

crime.skew()

Murder 0.393956

Assault 0.234410

UrbanPop -0.226009

Rape 0.801200

dtype: float64

crime.kurtosis()

Murder -0.827488

Assault -1.053848

UrbanPop -0.738360

Rape 0.353964

dtype: float64

**Calculating Range for Murder column**

crimemurder=crime["Murder"]

print(crimemurder)

0 13.2

1 10.0

2 8.1

3 8.8

4 9.0

5 7.9

6 3.3

7 5.9

8 15.4

9 17.4

10 5.3

11 2.6

12 10.4

13 7.2

14 2.2

15 6.0

16 9.7

17 15.4

18 2.1

19 11.3

20 4.4

21 12.1

22 2.7

23 16.1

24 9.0

25 6.0

26 4.3

27 12.2

28 2.1

29 7.4

30 11.4

31 11.1

32 13.0

33 0.8

34 7.3

35 6.6

36 4.9

37 6.3

38 3.4

39 14.4

40 3.8

41 13.2

42 12.7

43 3.2

44 2.2

45 8.5

46 4.0

47 5.7

48 2.6

49 6.8

Name: Murder, dtype: float64

print("range is",crime.Murder.max()-crime.Murder.min())

range is 16.599999999999998

In [ ]:

**Calculating Range for Assault column**

crimeassault=crime["Assault"]

print(crimeassault)

0 236

1 263

2 294

3 190

4 276

5 204

6 110

7 238

8 335

9 211

10 46

11 120

12 249

13 113

14 56

15 115

16 109

17 249

18 83

19 300

20 149

21 255

22 72

23 259

24 178

25 109

26 102

27 252

28 57

29 159

30 285

31 254

32 337

33 45

34 120

35 151

36 159

37 106

38 174

39 279

40 86

41 188

42 201

43 120

44 48

45 156

46 145

47 81

48 53

49 161

Name: Assault, dtype: int64

print("range is",crime.Assault.max()-crime.Assault.min())

range is 292

In [ ]:

**Calculating Range for UrbanPop column**

crimeurbanpop=crime["UrbanPop"]

print(crimeurbanpop)

0 58

1 48

2 80

3 50

4 91

5 78

6 77

7 72

8 80

9 60

10 83

11 54

12 83

13 65

14 57

15 66

16 52

17 66

18 51

19 67

20 85

21 74

22 66

23 44

24 70

25 53

26 62

27 81

28 56

29 89

30 70

31 86

32 45

33 44

34 75

35 68

36 67

37 72

38 87

39 48

40 45

41 59

42 80

43 80

44 32

45 63

46 73

47 39

48 66

49 60

Name: UrbanPop, dtype: int64

print("range is",crime.UrbanPop.max()-crime.UrbanPop.min())

range is 59

In [ ]:

**Calculating Range for Rape column**

crimerape=crime["Rape"]

print(crimerape)

0 21.2

1 44.5

2 31.0

3 19.5

4 40.6

5 38.7

6 11.1

7 15.8

8 31.9

9 25.8

10 20.2

11 14.2

12 24.0

13 21.0

14 11.3

15 18.0

16 16.3

17 22.2

18 7.8

19 27.8

20 16.3

21 35.1

22 14.9

23 17.1

24 28.2

25 16.4

26 16.5

27 46.0

28 9.5

29 18.8

30 32.1

31 26.1

32 16.1

33 7.3

34 21.4

35 20.0

36 29.3

37 14.9

38 8.3

39 22.5

40 12.8

41 26.9

42 25.5

43 22.9

44 11.2

45 20.7

46 26.2

47 9.3

48 10.8

49 15.6

Name: Rape, dtype: float64

print("range is",crime.Rape.max()-crime.Rape.min())

range is 38.7

In [ ]:

**28. Write a Python program to create a null vector of size 10 and update sixth value to 11.**

**Code to create a null vector of size 10**

import numpy as np

vector=np.zeros(10)

print(vector)

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

In [ ]:

**Updating the sixth value of a vector to 11**

vector[5]=11

print(vector)

[ 0. 0. 0. 0. 0. 11. 0. 0. 0. 0.]

In [ ]: