In [3]:

```
tup1 = (1, 2, 3, 4, 5, 7, 10, 11, 30)
```

In [18]:

```
tup1
```

Out[18]:

```
(1, 2, 3, 4, 5, 7, 10, 11, 30)
```

In [19]:

```
type(tup1)
```

Out[19]:

```
tuple
```

In [20]:

```
=(1,2,3,5,7,10,11,30,"pune")
```

In [21]:

```
tup3=("a","b","c","d",20,30,40)
```

In [22]:

```
tup3[2]
```

Out[22]:

```
'c'
```

In [4]:

```
del tup1
```

In [24]:

```
tup3=(tup2,"a","b","c","d",20,30,40)
```

In [27]:

```
tup3[4]=200 #immutable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-27-ea4a9ef8a869> in <module>
----> 1 tup3[4]=200

TypeError: 'tuple' object does not support item assignment
```

In [28]:

```
del(tup3) #delete
```

In [29]:

```
tup3 #check delete
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-29-f484f3ced80d> in <module>
----> 1 tup3

NameError: name 'tup3' is not defined
```

In [30]:

```
tup3=("a","b","c","c","c","d",200,30,40)
```

In [31]:

```
tup3.count("c")
```

Out[31]:

```
3
```

In [32]:

```
tup3.count(200)
```

Out[32]:

```
1
```

In [33]:

```
tup3.index(200)
```

Out[33]:

```
6
```

In [5]:

```
# LISTS

l1=[1,2,3,4,11,12,40,50,70,'data'] #creation
```

In [6]:

```
print(l1)
type(l1)
```

```
[1, 2, 3, 4, 11, 12, 40, 50, 70, 'data']
```

Out[6]:

```
list
```

In [7]:

```
l1[4]
```

Out[7]:

```
11
```

In [8]:

```
l1[4]=111111
```

In [9]:

```
#l1
l1[0:5]
```

Out[9]:

```
[1, 2, 3, 4, 111111]
```

l2= list([1,2,3,4,11,12,40,50,70,'data'])

In [3]:

```
l2= list([1,2,3,4,11,12,40,50,70,'data',"india"])
```

In [4]:

```
type(l2)
```

Out[4]:

```
list
```

In [5]:

```
l2.append("new entry")    #TO ADD AN ELEMENT
```

In [6]:

```
l2
```

Out[6]:

```
[1, 2, 3, 4, 11, 12, 40, 50, 70, 'data', 'india', 'new entry']
```

In [7]:

```
l3=[]   #EMPTY LIST
```

In [8]:

```
l3.append("a")
```

In [9]:

```
l3.append("b")
```

In [10]:

```
l3
```

Out[10]:

```
['a', 'b']
```

In [11]:

```
l3.append(["c","d"])
```

In [12]:

```
l3
```

Out[12]:

```
['a', 'b', ['c', 'd']]
```

In [14]:

```
l2.count()
```

```
  File "<ipython-input-14-430963f7bc66>", line 1
    l2.count(*)
              ^
SyntaxError: invalid syntax
```

In [50]:

```
l2.count('india')
```

Out[50]:

```
1
```

In [51]:

```
l2.pop() #any random data from l2
```

Out[51]:

'new entry'

In [56]:

```
l2.pop() #throws random variable
```

Out[56]:

70

In [53]:

```
l2.pop()
```

Out[53]:

'data'

In [55]:

```
len(l2) #countslength
```

Out[55]:

9

In [57]:

```
#dictionary
```

In [15]:

```
dict1={"name":"Mr.A"   ,   "Age" :30}
```

In [17]:

```
#dict1.keys()
dict1.items()
```

Out[17]:

dict_items([('name', 'Mr.A'), ('Age', 30)])

In [60]:

```
dict1.values()
```

Out[60]:

dict_values(['Mr.A', 30])

In [61]:

```
empy_details = {   "Name": ("Mr.A","Mr.B","Mr.C") ,
                 "Age": (30 ,25 , 34 ) ,
                   "salary":(100,200,300) ,
                   "Department":("Fin", "Account", "reatil")}
```

In [63]:

```
empy_details.keys()
```

Out[63]:

dict_keys(['Name', 'Age', 'salary', 'Department'])

In [64]:

```
empy_details.values()
```

Out[64]:

dict_values([('Mr.A', 'Mr.B', 'Mr.C'), (30, 25, 34), (100, 200, 300), ('Fin', 'Account', 'reatil')])

In [65]:

```
import pandas as pd
```

In [66]:

```
df1 = pd.DataFrame(empy_details) #convert dict to data frames df1=variable
```

In [67]:

```
df1
```

Out[67]:

	Name	Age	salary	Department
0	Mr.A	30	100	Fin
1	Mr.B	25	200	Account
2	Mr.C	34	300	reatil

In [68]:

```
type(df1)
```

Out[68]:

pandas.core.frame.DataFrame

In [12]:

```
#   SETS



s1= set([1,4,4,3,3,3,34,4,6,7,8,9,10])
```

In [13]:

```
type(s1)
```

Out[13]:

set

In [14]:

```
s1
```

Out[14]:

{1, 3, 4, 6, 7, 8, 9, 10, 34}

In [16]:

```
s1[1]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-da05ae654f28> in <module>
----> 1 s1[1]

TypeError: 'set' object is not subscriptable
```

In [18]:

```
l1=[1,2,2,3,3,4,4,4,67]
```

```
type(l1)
```

Out[18]:

```
list
```

In [74]:

```
l1=set(l1)     #list to set conversation
```

In [75]:

```
type(l1)
```

Out[75]:

```
set
```

In [76]:

```
1+2
```

Out[76]:

```
3
```

In [80]:

```
name = "Mr A"
age = 33
```

In [81]:

```
print(name)
```

```
Mr A
```

In [82]:

```
print(age)
```

```
33
```

In [83]:

```
print(name,age)
```

```
Mr A 33
```

In [84]:

```
print("the name of the person is ",name)
```

```
the name of the person is  Mr A
```

In [85]:

```
#print. format
```

In [87]:

```
print("his name is {} and his age is {}".format(name ,age))
```

```
his name is Mr A and his age is 33
```

In [19]:

```
#conditional statements
```

In [92]:

```
i= 11
```

```
if i<10:
    print("value of i is.." ,i)
else:
    print("value of i is greater than 10..." , i)
    print("inside the else part")
```

```
value of i is greater than 10... 11
inside the else part
```

In [93]:

```
city = "Mumbai"

if city == "Mumbai":
    print("it is capital of maharashta" , city)
    print("This city is the financial captial of india" , city)
elif  city == "Chennai":                                       #  if else
    print("its capital of TN" , city)
    print("its automobile hub" , city)
elif city  ==  "Delhi":
    print("its capital of India" , city)
```

```
it is capital of maharashta Mumbai
This city is the financial captial of india Mumbai
```

In [95]:

```
#to take the input from the user

value = input("the number which you want to pass..")
```

```
the number which you want to pass..1000
```

In [96]:

```
value
```

Out[96]:

```
'1000'
```

In [97]:

```
value = eval(input("the number which you want to pass.."))    #used eval in order to conce
rt string value to int value
```

```
the number which you want to pass..1000
```

In [98]:

```
value +1000
```

Out[98]:

```
2000
```

In [101]:

```
if city == "Mumbai":
    print("it is capital of maharashta" , city)
    print("This city is the financial captial of india" , city)
elif  city == "Chennai":                                       #  if else
    print("its capital of TN" , city)
    print("its automobile hub" , city)
elif city  ==  "Delhi":
    print("its capital of India" , city)
else:
    print("this city is not in my data base")
```

```
it is capital of maharashta Mumbai
This city is the financial captial of india Mumbai
```

In [102]:

In [102]:

```
city  = eval(input("please enter the name of the city "))    #to let the user enter tha va
lue

if city == "Mumbai":
    print("it is capital of maharashta" , city)
    print("This city is the financial captial of india" , city)
elif  city == "Chennai":                                   #  if else
    print("its capital of TN" , city)
    print("its automobile hub" , city)
elif city  ==  "Delhi":
    print("its capital of India" , city)
else:
    print("this city is not in my data base")
```

```
please enter the name of the city "Mumbai"
it is capital of maharashta Mumbai
This city is the financial captial of india Mumbai
```

In [103]:

```
#loops
```

In [104]:

```
cars = ["Maruti" , "Audi" , "Merc"]
```

In [106]:

```
for i in cars :
    print(i)
```

```
Maruti
Audi
Merc
```

In [108]:

```
number = [1,2,3,4,5,6,7,8,9,10]

for j in number:
    print(j)
    print(j*j)
    print("--")
```

```
1
1
--
2
4
--
3
9
--
4
16
--
5
25
--
6
36
--
7
49
--
8
64
--
9
81
--
```

```
--
10
100
--
```

In [21]:

```
#create a user defined function
#fun--> add_num

def add_num(a,b,c):
    print(a)
    print(b)
    val=a+b
    print("value of val is..",val)
```

In [22]:

```
add_num(10,20,30)   #calling the function
```

```
10
20
value of val is.. 30
```

In [23]:

```
val           #erroe because it is a local variable
                        #      to access make it global
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-23-546ba5ecc549> in <module>
----> 1 val           #erroe because it is a local variable
      2                         #      to access make it global

NameError: name 'val' is not defined
```

In [26]:

```
def add_num(a,b,c):
    print(a)
    print(b)
    global val
    val=a+b
    print("value of val is..",val)
```

In [27]:

```
add_num(10,20,30)
```

```
10
20
value of val is.. 30
```

In [28]:

```
val
```

Out[28]:

```
30
```

In [122]:

```
# data frames
#to read a file-->
#path > file name > type
```

In [3]:

```
import pandas as pd
```

In [4]:

```
#cr is a dtaframe

cr = pd.read_csv(r"C:\Users\nb291\Desktop\CreditRisk - CreditRisk.csv")

#file has been read pd.read_csv is the function ... we need to use r bfore giving the pat
h it stands for raw
```

In [5]:

```
cr.head() # it will show top 5 records
```

Out[5]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0.0	Graduate	No	6000	0.0	141.0	

In [6]:

```
import os   #operating system
```

In [7]:

```
os.getcwd()    #current working directory
```

Out[7]:

```
'C:\\Users\\nb291'
```

In [8]:

```
#copy paste the loaction/path followed by the name of file and.csv
```

In [9]:

```
cr
```

Out[9]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount
0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	NaN
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.0
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	0.0	66.0
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	2358.0	120.0
4	LP001008	Male	No	0.0	Graduate	No	6000	0.0	141.0
...	...	...	...	...	...	...	...	...	...
976	LP002971	Male	Yes	4.0	Not Graduate	Yes	4009	1777.0	113.0
977	LP002975	Male	Yes	0.0	Graduate	No	4158	709.0	115.0
978	LP002980	Male	No	0.0	Graduate	No	3250	1993.0	126.0
979	LP002986	Male	Yes	0.0	Graduate	No	5000	2393.0	158.0
980	LP002989	Male	No	0.0	Graduate	Yes	9200	0.0	98.0

981 rows × 13 columns

In [10]:

```
cr.tail()    #last 5 records
```

Out[10]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount
976	LP002971	Male	Yes	4.0	Not Graduate	Yes	4009	1777.0	113.0
977	LP002975	Male	Yes	0.0	Graduate	No	4158	709.0	115.0
978	LP002980	Male	No	0.0	Graduate	No	3250	1993.0	126.0
979	LP002986	Male	Yes	0.0	Graduate	No	5000	2393.0	158.0
980	LP002989	Male	No	0.0	Graduate	Yes	9200	0.0	98.0

In [11]:

```
cr.head(10)    #to see the top 10 rec ..just passs the value
```

Out[11]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0.0	Graduate	No	6000	0.0	141.0	
5	LP001011	Male	Yes	2.0	Graduate	Yes	5417	4196.0	267.0	
6	LP001013	Male	Yes	0.0	Not Graduate	No	2333	1516.0	95.0	
7	LP001014	Male	Yes	4.0	Graduate	No	3036	2504.0	158.0	
8	LP001018	Male	Yes	2.0	Graduate	No	4006	1526.0	168.0	
9	LP001020	Male	Yes	1.0	Graduate	No	12841	10968.0	349.0	

In [12]:

```
cr.shape    #gives number of rows and columns
```

Out[12]:

```
(981, 13)
```

In [13]:

```
cr.columns #gives names of columns
```

Out[13]:

```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [14]:

```
cr.describe()    #gives the description of columns
```

Out[14]:

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	956.000000	981.000000	981.000000	954.000000	961.000000	902.000000
mean	0.881799	5179.795107	1601.916330	142.511530	342.201873	0.835920
std	1.255623	5695.104533	2718.772806	77.421743	65.100602	0.370553
min	0.000000	0.000000	0.000000	9.000000	6.000000	0.000000
25%	0.000000	2875.000000	0.000000	100.000000	360.000000	1.000000
50%	0.000000	3800.000000	1110.000000	126.000000	360.000000	1.000000
75%	2.000000	5516.000000	2365.000000	162.000000	360.000000	1.000000
max	4.000000	81000.000000	41667.000000	700.000000	480.000000	1.000000

In [15]:

```
#description of only 6 col is given here ...?
#it is giving the description of only numerical data
#foll all descriptions we need to write the following code
```

In [16]:

```
cr.describe(include = "all")
```

Out[16]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmou
count	981	957	978	956.000000	981	926	981.000000	981.000000	954.00000
unique	981	2	2	NaN	2	2	NaN	NaN	Na
top	LP001908	Male	Yes	NaN	Graduate	No	NaN	NaN	Na
freq	1	775	631	NaN	763	807	NaN	NaN	Na
mean	NaN	NaN	NaN	0.881799	NaN	NaN	5179.795107	1601.916330	142.51153
std	NaN	NaN	NaN	1.255623	NaN	NaN	5695.104533	2718.772806	77.42174
min	NaN	NaN	NaN	0.000000	NaN	NaN	0.000000	0.000000	9.00000
25%	NaN	NaN	NaN	0.000000	NaN	NaN	2875.000000	0.000000	100.00000
50%	NaN	NaN	NaN	0.000000	NaN	NaN	3800.000000	1110.000000	126.00000
75%	NaN	NaN	NaN	2.000000	NaN	NaN	5516.000000	2365.000000	162.00000
max	NaN	NaN	NaN	4.000000	NaN	NaN	81000.000000	41667.000000	700.00000

In [17]:

```
cr.Gender#to acces a particular dat
```

Out[17]:

```
0       Male
1       Male
2       Male
3       Male
4       Male
        ...
976     Male
977     Male
978     Male
979     Male
980     Male
Name: Gender, Length: 981, dtype: object
```

In [18]:

```python
cr['Gender']
```

Out[18]:

```
0      Male
1      Male
2      Male
3      Male
4      Male
       ...
976    Male
977    Male
978    Male
979    Male
980    Male
Name: Gender, Length: 981, dtype: object
```

In [19]:

```python
cr.ApplicantIncome.mean()
```

Out[19]:

```
5179.795107033639
```

In [20]:

```python
cr.ApplicantIncome.median()
```

Out[20]:

```
3800.0
```

In [21]:

```python
cr.ApplicantIncome.sum()
```

Out[21]:

```
5081379
```

In [22]:

```python
cr.ApplicantIncome.max()
```

Out[22]:

```
81000
```

In [23]:

```python
cr.Gender.value_counts()
```

Out[23]:

```
Male      775
Female    182
Name: Gender, dtype: int64
```

In [24]:

```python
cr.Property_Area.value_counts()
```

Out[24]:

```
Semiurban    349
Urban        342
Rural        290
Name: Property_Area, dtype: int64
```

In [25]:

```python
#check the null values and replace thrm with some logical value
```

In [26]:

```
cr.isnull().sum()
```

Out[26]:

```
Loan_ID              0
Gender              24
Married              3
Dependents          25
Education            0
Self_Employed       55
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount          27
Loan_Amount_Term    20
Credit_History      79
Property_Area        0
Loan_Status          0
dtype: int64
```

In [27]:

```
#make an assumptions

#for a numeric columnas can replace null with mean/median

#for non numeric column pass most frequenr value
```

In [28]:

```
cr.Gender = cr.Gender.fillna("Male") #fillna = fill null values
```

In [29]:

```
cr.Married = cr.Married.fillna("Yes")
```

In [30]:

```
cr.Dependents = cr.Dependents.fillna(0)
```

In [31]:

```
cr.Self_Employed = cr.Self_Employed.fillna("No")
```

In [32]:

```
cr.LoanAmount = cr.LoanAmount.fillna( cr.LoanAmount.mean())
```

In [33]:

```
cr.Loan_Amount_Term = cr.Loan_Amount_Term.fillna(cr.Loan_Amount_Term.mean())
```

In [34]:

```
cr.Credit_History = cr.Credit_History.fillna(0)
```

In [35]:

```
cr.isnull().sum()
```

Out[35]:

```
Loan_ID             0
Gender              0
Married             0
Dependents          0
Education           0
Self_Employed       0
ApplicantIncome     0
CoapplicantIncome   0
```

```
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

In [36]:

```
cr.groupby('Gender').ApplicantIncome.agg(['count','min','max','mean'])
```

Out[36]:

	count	min	max	mean
Gender				
Female	182	0	19484	4458.906593
Male	799	0	81000	5344.002503

In [37]:

```
cr.groupby('Gender').ApplicantIncome.agg(['median','mean'])
```

Out[37]:

	median	mean
Gender		
Female	3634.5	4458.906593
Male	3859.0	5344.002503

In [38]:

```
cr.groupby('Gender').LoanAmount.agg(['median','mean'])
```

Out[38]:

	median	mean
Gender		
Female	113.0	127.082671
Male	130.0	146.025989

In [39]:

```
#try read different files
#remove the null values
#try to group by based on several categorical column
#run head and tail
```

In [41]:

```
#13-07-2020
```

In [43]:

```
cr.groupby(["Gender" ,"Education" ]).ApplicantIncome.mean()
#values are mean values
```

Out[43]:

```
Gender  Education
Female  Graduate        4598.175676
        Not Graduate    3852.676471
Male    Graduate        5844.117073
        Not Graduate    3672.423913
Name: ApplicantIncome, dtype: float64
```

Name: ApplicantIncome, dtype: float64

In [47]:

```
pd.crosstab(cr.Gender   ,   cr.Loan_Status)   #confusion matrix
```

Out[47]:

Loan_Status	N	Y
Gender		
Female	55	127
Male	214	585

In [48]:

```
cr.ApplicantIncome.describe() #analysis of the column
```

Out[48]:

```
count      981.000000
mean      5179.795107
std       5695.104533
min          0.000000
25%       2875.000000
50%       3800.000000
75%       5516.000000
max      81000.000000
Name: ApplicantIncome, dtype: float64
```

#25% people are earning less or equal to 2875

In [49]:

```
cr.ApplicantIncome.describe(percentiles = [.1 , .2 , .3 ,.4 , .5 , .6 , .7 , .8 , .9 ,
1])
```

Out[49]:

```
count      981.000000
mean      5179.795107
std       5695.104533
min          0.000000
10%       2221.000000
20%       2609.000000
30%       3062.000000
40%       3413.000000
50%       3800.000000
60%       4301.000000
70%       5000.000000
80%       6080.000000
90%       8750.000000
100%     81000.000000
max      81000.000000
Name: ApplicantIncome, dtype: float64
```

In [50]:

```
#rename the columns

cr1= cr.rename( columns = {"ApplicantIncome"  : "Applicant-Income"})
```

In [51]:

```
cr1.head()
```

Out[51]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Applicant-Income	CoapplicantIncome	LoanAmount	Loan_Am
0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	142.51153	

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Am
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.00000	
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	0.0	66.00000	
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	2358.0	120.00000	
4	LP001008	Male	No	0.0	Graduate	No	6000	0.0	141.00000	

In [52]:

```
cr1= cr.rename( columns = {"Gender"  : "Gender1" , "Married" : "Married1"})
```

In [54]:

```
cr1.head()
```

Out[54]:

	Loan_ID	Gender1	Married1	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount
0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	142.51153
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.00000
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	0.0	66.00000
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	2358.0	120.00000
4	LP001008	Male	No	0.0	Graduate	No	6000	0.0	141.00000

In [61]:

```
cr.drop(["Loan_ID"] , axis=1)   #is used to drop the column   axis =1 means we are applyi
ng the function
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-61-d44030af1b01> in <module>
----> 1 cr.drop(["Loan_ID"] , axis=1)   #is used to drop the column   axis =1 means we ar
e applying the function

~\anaconda3\lib\site-packages\pandas\core\frame.py in drop(self, labels, axis, index, col
umns, level, inplace, errors)
   3995             level=level,
   3996             inplace=inplace,
-> 3997             errors=errors,
   3998         )
   3999

~\anaconda3\lib\site-packages\pandas\core\generic.py in drop(self, labels, axis, index, c
olumns, level, inplace, errors)
   3934         for axis, labels in axes.items():
   3935             if labels is not None:
-> 3936                 obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   3937
   3938         if inplace:

~\anaconda3\lib\site-packages\pandas\core\generic.py in _drop_axis(self, labels, axis, le
vel, errors)
   3968                 new_axis = axis.drop(labels, level=level, errors=errors)
   3969             else:
-> 3970                 new_axis = axis.drop(labels, errors=errors)
   3971             result = self.reindex(**{axis_name: new_axis})
   3972

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in drop(self, labels, errors)
   5016         if mask.any():
   5017             if errors != "ignore":
-> 5018                 raise KeyError(f"{labels[mask]} not found in axis")
```

```
   5019             indexer = indexer[~mask]
   5020         return self.delete(indexer)

KeyError: "['Loan_ID'] not found in axis"
```

In [56]:

```
cr = cr.drop(["Loan_ID"] , axis=1)
```

In [57]:

```
cr.head()
```

Out[57]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount
0	Male	No	0.0	Graduate	No	5849	0.0	142.51153	
1	Male	Yes	1.0	Graduate	No	4583	1508.0	128.00000	
2	Male	Yes	0.0	Graduate	Yes	3000	0.0	66.00000	
3	Male	Yes	0.0	Not Graduate	No	2583	2358.0	120.00000	
4	Male	No	0.0	Graduate	No	6000	0.0	141.00000	

In [2]:

```
cr.drop( ["Loan_ID"] , axis = 1 , inplace = True)    #error because loadid id dropped alr
eady
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-e73910bfb8fa> in <module>
----> 1 cr.drop( ["Loan_ID"] , axis = 1 , inplace = True)    #error because loadid id dro
pped already

NameError: name 'cr' is not defined
```

In [4]:

```
# 14-07-2020
import pandas as pd
```

In [6]:

```
cr =pd.read_csv(r"C:\Users\nb291\Desktop\CreditRisk - CreditRisk.csv")
```

In [8]:

```
cr.head(2)
```

Out[8]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.0	

In [9]:

```
cr.rename(columns = {cr.columns[1] : "Gender1"} , inplace  = True)
```

In [10]:

```
cr.head(2)
```

Out[10]:

Out[10]:

	Loan_ID	Gender1	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	L
0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.0	

In [13]:

```
cr.sort_values(['ApplicantIncome'])    #sorting of data with respect to applicant income i
n asc order
```

Out[13]:

	Loan_ID	Gender1	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount
639	LP001153	Male	No	0.0	Graduate	No	0	24000.0	148.0
730	LP001607	Female	No	0.0	Not Graduate	No	0	1760.0	180.0
216	LP001722	Male	Yes	0.0	Graduate	No	150	1800.0	135.0
468	LP002502	Female	Yes	2.0	Not Graduate	NaN	210	2917.0	98.0
600	LP002949	Female	No	4.0	Graduate	NaN	416	41667.0	350.0
...	...	...	...	...	...	...	...	...	...
155	LP001536	Male	Yes	4.0	Graduate	No	39999	0.0	600.0
171	LP001585	NaN	Yes	4.0	Graduate	No	51763	0.0	700.0
333	LP002101	Male	Yes	0.0	Graduate	NaN	63337	0.0	490.0
695	LP001428	Male	Yes	4.0	Graduate	No	72529	0.0	360.0
409	LP002317	Male	Yes	4.0	Graduate	No	81000	0.0	360.0

981 rows × 13 columns

In [14]:

```
#by pressing shift tab in the parenthesis if the function we can chcek the detailing
```

In [15]:

```
#iloc and loc : if you want to select rows and columns
```

In [16]:

```
#suppose i want to select 1, 10 , 20 , 21, 25 rows and 1, 4, 5, 7columns
```

In [17]:

```
cr.iloc[[0,9,19,20,24 ]   ,[0,3,4,6]]   #indexlocation (values on the left side of the c
omma indicates rows)
```

Out[17]:

	Loan_ID	Dependents	Education	ApplicantIncome
0	LP001002	0.0	Graduate	5849
9	LP001020	1.0	Graduate	12841
19	LP001041	0.0	Graduate	2600
20	LP001043	0.0	Not Graduate	7660
24	LP001052	1.0	Graduate	3717

In [18]:

```
#i want to select all rows and some columns of my interest
```

cr.iloc[: , [0,3,4,6]] #": " indicates all rows

In [22]:

```
cr.iloc[ :  , [0,3,4,6]]  #": " indicates all rows
```

Out[22]:

	Loan_ID	Dependents	Education	ApplicantIncome
0	LP001002	0.0	Graduate	5849
1	LP001003	1.0	Graduate	4583
2	LP001005	0.0	Graduate	3000
3	LP001006	0.0	Not Graduate	2583
4	LP001008	0.0	Graduate	6000
...	...	...	...	...
976	LP002971	4.0	Not Graduate	4009
977	LP002975	0.0	Graduate	4158
978	LP002980	0.0	Graduate	3250
979	LP002986	0.0	Graduate	5000
980	LP002989	0.0	Graduate	9200

981 rows × 4 columns

In [23]:

```
cr.iloc[0:981   , [0,3,4,6]]
```

Out[23]:

	Loan_ID	Dependents	Education	ApplicantIncome
0	LP001002	0.0	Graduate	5849
1	LP001003	1.0	Graduate	4583
2	LP001005	0.0	Graduate	3000
3	LP001006	0.0	Not Graduate	2583
4	LP001008	0.0	Graduate	6000
...	...	...	...	...
976	LP002971	4.0	Not Graduate	4009
977	LP002975	0.0	Graduate	4158
978	LP002980	0.0	Graduate	3250
979	LP002986	0.0	Graduate	5000
980	LP002989	0.0	Graduate	9200

981 rows × 4 columns

In [25]:

```
cr.iloc[:,[-1]]  #-1 will give the last column
```

Out[25]:

	Loan_Status
0	Y

	Loan_Status
1	N
2	Y
3	Y
4	Y
...	...
976	Y
977	Y
978	Y
979	N
980	Y

981 rows × 1 columns

In [27]:

```
# 1 df which has records from 0 to 200
#2 df which has records from 400 to 600
#then merge

#                    df3=pd.concat([df1,df2] , axis=0)
#axis=0: rows
#axis=1  : columns
```

In [30]:

```
cr.loc[: , ["Dependents" , "Education"]]   # we use loc when we give name of the columns
```

Out[30]:

	Dependents	Education
0	0.0	Graduate
1	1.0	Graduate
2	0.0	Graduate
3	0.0	Not Graduate
4	0.0	Graduate
...	...	...
976	4.0	Not Graduate
977	0.0	Graduate
978	0.0	Graduate
979	0.0	Graduate
980	0.0	Graduate

981 rows × 2 columns

In [32]:

```
cr.isnull().sum()*100/981        #for getting the percentage of the null values
```

Out[32]:

```
Loan_ID              0.000000
Gender1              2.446483
Married              0.305810
Dependents           2.548420
Education            0.000000
Self_Employed        5.606524
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           2.752294
```

```
Loan_Amount_Term      2.038736
Credit_History        8.053007
Property_Area         0.000000
Loan_Status           0.000000
dtype: float64
```

In [33]:

```
cr =pd.read_csv(r"C:\Users\nb291\Desktop\CreditRisk - CreditRisk.csv")
```

In [34]:

```
cr.head(5)
```

Out[34]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0.0	Graduate	No	6000	0.0	141.0	

In [39]:

```
cr.Self_Employed.replace({"No":0 , "Yes":1} , inplace=True)
```

In [40]:

```
cr.head(5)
```

Out[40]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	LP001002	Male	No	0.0	Graduate	0.0	5849	0.0	NaN	
1	LP001003	Male	Yes	1.0	Graduate	0.0	4583	1508.0	128.0	
2	LP001005	Male	Yes	0.0	Graduate	1.0	3000	0.0	66.0	
3	LP001006	Male	Yes	0.0	Not Graduate	0.0	2583	2358.0	120.0	
4	LP001008	Male	No	0.0	Graduate	0.0	6000	0.0	141.0	

In [43]:

```
cr.info()    #gives the information of the columns , object type
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 981 entries, 0 to 980
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Loan_ID             981 non-null    object 
 1   Gender              957 non-null    object 
 2   Married             978 non-null    object 
 3   Dependents          956 non-null    float64
 4   Education           981 non-null    object 
 5   Self_Employed       926 non-null    float64
 6   ApplicantIncome     981 non-null    int64  
 7   CoapplicantIncome   981 non-null    float64
 8   LoanAmount          954 non-null    float64
 9   Loan_Amount_Term    961 non-null    float64
 10  Credit_History      902 non-null    float64
```

```
 11   Property_Area       981 non-null    object
 12   Loan_Status         981 non-null    object
dtypes: float64(6), int64(1), object(6)
memory usage: 99.8+ KB
```

In [44]:

```
#inplace signifies to make the changes in the same data frame
```

In [48]:

```
cr.replace({"Gender"  :{"Male":0   , "Female":1 },
           "Married" :{"No":0  ,"Yes":1}}, inplace = True)
#replace the values of multiple columns
```

In [49]:

```
#now lets select the data based in conditions
```

In [50]:

```
#i want to select all the records which are earning more

aa = cr.ApplicantIncome > 3000    #aa is a variable
```

In [51]:

```
aa
```

Out[51]:

```
0       True
1       True
2      False
3      False
4       True
       ...
976     True
977     True
978     True
979     True
980     True
Name: ApplicantIncome, Length: 981, dtype: bool
```

In [52]:

```
len(aa)
```

Out[52]:

```
981
```

In [53]:

```
cr.shape
```

Out[53]:

```
(981, 13)
```

In [54]:

```
#number of records in aa and cr is same ..... we get true or false for each record
```

In [55]:

```
df4 = cr[aa]
```

In [57]:

```
df4 #consists data of people having applicantincome more than 3000
```

Out[57]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount
0	LP001002	0.0	0.0	0.0	Graduate	0.0	5849	0.0	NaN
1	LP001003	0.0	1.0	1.0	Graduate	0.0	4583	1508.0	128.0
4	LP001008	0.0	0.0	0.0	Graduate	0.0	6000	0.0	141.0
5	LP001011	0.0	1.0	2.0	Graduate	1.0	5417	4196.0	267.0
7	LP001014	0.0	1.0	4.0	Graduate	0.0	3036	2504.0	158.0
...	...	...	...	...	...	...	...	...	...
976	LP002971	0.0	1.0	4.0	Not Graduate	1.0	4009	1777.0	113.0
977	LP002975	0.0	1.0	0.0	Graduate	0.0	4158	709.0	115.0
978	LP002980	0.0	0.0	0.0	Graduate	0.0	3250	1993.0	126.0
979	LP002986	0.0	1.0	0.0	Graduate	0.0	5000	2393.0	158.0
980	LP002989	0.0	0.0	0.0	Graduate	1.0	9200	0.0	98.0

700 rows × 13 columns

In [61]:

```
df5 = cr[cr.ApplicantIncome >= 3000]     #another method of ding the same thing
```

In [62]:

```
df5
```

Out[62]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount
0	LP001002	0.0	0.0	0.0	Graduate	0.0	5849	0.0	NaN
1	LP001003	0.0	1.0	1.0	Graduate	0.0	4583	1508.0	128.0
2	LP001005	0.0	1.0	0.0	Graduate	1.0	3000	0.0	66.0
4	LP001008	0.0	0.0	0.0	Graduate	0.0	6000	0.0	141.0
5	LP001011	0.0	1.0	2.0	Graduate	1.0	5417	4196.0	267.0
...	...	...	...	...	...	...	...	...	...
976	LP002971	0.0	1.0	4.0	Not Graduate	1.0	4009	1777.0	113.0
977	LP002975	0.0	1.0	0.0	Graduate	0.0	4158	709.0	115.0
978	LP002980	0.0	0.0	0.0	Graduate	0.0	3250	1993.0	126.0
979	LP002986	0.0	1.0	0.0	Graduate	0.0	5000	2393.0	158.0
980	LP002989	0.0	0.0	0.0	Graduate	1.0	9200	0.0	98.0

703 rows × 13 columns

In [63]:

```
df6 = cr[- (cr.ApplicantIncome >= 3000)]    # -sign is for not equal to
```

In [64]:

```
# create df7  based on condition: earning more than 300 person should be male and married
```

In [67]:

```
df7 =cr[(cr.ApplicantIncome >= 3000)&(cr.Gender==0) &(cr.Married==1)]
```

In [68]:

```
df7
```

Out[68]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount
1	LP001003	0.0	1.0	1.0	Graduate	0.0	4583	1508.0	128.0
2	LP001005	0.0	1.0	0.0	Graduate	1.0	3000	0.0	66.0
5	LP001011	0.0	1.0	2.0	Graduate	1.0	5417	4196.0	267.0
7	LP001014	0.0	1.0	4.0	Graduate	0.0	3036	2504.0	158.0
8	LP001018	0.0	1.0	2.0	Graduate	0.0	4006	1526.0	168.0
...	...	...	...	...	...	...	...	...	...
970	LP002935	0.0	1.0	1.0	Graduate	0.0	3791	1936.0	85.0
972	LP002954	0.0	1.0	2.0	Not Graduate	0.0	3132	0.0	76.0
976	LP002971	0.0	1.0	4.0	Not Graduate	1.0	4009	1777.0	113.0
977	LP002975	0.0	1.0	0.0	Graduate	0.0	4158	709.0	115.0
979	LP002986	0.0	1.0	0.0	Graduate	0.0	5000	2393.0	158.0

411 rows × 13 columns

In [69]:

```
cr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 981 entries, 0 to 980
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            981 non-null    object 
 1   Gender             957 non-null    float64
 2   Married            978 non-null    float64
 3   Dependents         956 non-null    float64
 4   Education          981 non-null    object 
 5   Self_Employed      926 non-null    float64
 6   ApplicantIncome    981 non-null    int64  
 7   CoapplicantIncome  981 non-null    float64
 8   LoanAmount         954 non-null    float64
 9   Loan_Amount_Term   961 non-null    float64
 10  Credit_History     902 non-null    float64
 11  Property_Area      981 non-null    object 
 12  Loan_Status        981 non-null    object 
dtypes: float64(8), int64(1), object(4)
memory usage: 99.8+ KB
```

In [70]:

```
cr.ApplicantIncome = cr.ApplicantIncome.astype("float")
```

In [72]:

```
cr.info()          #applicant income data type is now float
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 981 entries, 0 to 980
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            981 non-null    object
```

```
 0   Loan_ID            981 non-null    object
 1   Gender             957 non-null    float64
 2   Married            978 non-null    float64
 3   Dependents         956 non-null    float64
 4   Education          981 non-null    object
 5   Self_Employed      926 non-null    float64
 6   ApplicantIncome    981 non-null    float64
 7   CoapplicantIncome  981 non-null    float64
 8   LoanAmount         954 non-null    float64
 9   Loan_Amount_Term   961 non-null    float64
 10  Credit_History     902 non-null    float64
 11  Property_Area      981 non-null    object
 12  Loan_Status        981 non-null    object
dtypes: float64(9), object(4)
memory usage: 99.8+ KB
```

In [73]:

```
# label incoder
# to convert non numeric to numeric
```

In [75]:

```
import sklearn        #sklearn is a  library
```

In [76]:

```
from sklearn.preprocessing import LabelEncoder
```

In [78]:

```
le= LabelEncoder()     # create a object of the label encoder as python the object orient
ed programming
```

In [79]:

```
cr =pd.read_csv(r"C:\Users\nb291\Desktop\CreditRisk - CreditRisk.csv")
```

In [80]:

```
cr = cr.dropna()
```

In [84]:

```
cr.Gender = le.fit_transform(cr.Gender)
cr.Self_Employed = le.fit_transform(cr.Self_Employed)     #labelencoder will transform
cr.Education = le.fit_transform(cr.Education)
```

In [85]:

```
cr.head()
```

Out[85]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
1	LP001003	1	Yes	1.0	0	0	4583	1508.0	128.0	
2	LP001005	1	Yes	0.0	0	1	3000	0.0	66.0	
3	LP001006	1	Yes	0.0	1	0	2583	2358.0	120.0	
4	LP001008	1	No	0.0	0	0	6000	0.0	141.0	
5	LP001011	1	Yes	2.0	0	1	5417	4196.0	267.0	

In [86]:

```
cr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 769 entries, 1 to 980
```

```
Int64Index: 769 entries, 1 to 980
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            769 non-null    object 
 1   Gender             769 non-null    int32  
 2   Married            769 non-null    object 
 3   Dependents         769 non-null    float64
 4   Education          769 non-null    int32  
 5   Self_Employed      769 non-null    int32  
 6   ApplicantIncome    769 non-null    int64  
 7   CoapplicantIncome  769 non-null    float64
 8   LoanAmount         769 non-null    float64
 9   Loan_Amount_Term   769 non-null    float64
 10  Credit_History     769 non-null    float64
 11  Property_Area      769 non-null    object 
 12  Loan_Status        769 non-null    object 
dtypes: float64(5), int32(3), int64(1), object(4)
memory usage: 75.1+ KB
```

In [88]:

```
cr =pd.read_csv(r"C:\Users\nb291\Desktop\CreditRisk - CreditRisk.csv")
cr = cr.dropna()
cr1 = cr.iloc[ : , 1:13]
```

In [89]:

```
cr.head()
```

Out[89]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0.0	Graduate	No	6000	0.0	141.0	
5	LP001011	Male	Yes	2.0	Graduate	Yes	5417	4196.0	267.0	

In [94]:

```
cr1[cr1.select_dtypes(include=['object']).columns]  = cr1[cr1.select_dtypes(include=['obj
ect']).columns].apply(le.fit_transform)
```

In [95]:

```
cr1.head()
```

Out[95]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount
1	1	1	1.0	0	0	4583	1508.0	128.0	
2	1	1	0.0	0	1	3000	0.0	66.0	
3	1	1	0.0	1	0	2583	2358.0	120.0	
4	1	0	0.0	0	0	6000	0.0	141.0	
5	1	1	2.0	0	1	5417	4196.0	267.0	

In [96]:

```
cr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 769 entries, 1 to 980
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            769 non-null    int32
 1   Gender             769 non-null    int32
 2   Married            769 non-null    int32
 3   Dependents         769 non-null    float64
 4   Education          769 non-null    int32
 5   Self_Employed      769 non-null    int32
 6   ApplicantIncome    769 non-null    int64
 7   CoapplicantIncome  769 non-null    float64
 8   LoanAmount         769 non-null    float64
 9   Loan_Amount_Term   769 non-null    float64
 10  Credit_History     769 non-null    float64
 11  Property_Area      769 non-null    int32
 12  Loan_Status        769 non-null    int32
dtypes: float64(5), int32(7), int64(1)
memory usage: 63.1 KB
```

In [97]:

```
cr =pd.read_csv(r"C:\Users\nb291\Desktop\CreditRisk - CreditRisk.csv")
```

In [99]:

```
cr.corr()         #gives the co-rel matrix , in this we have name of the columns ...can be
only found on numeric data
```

Out[99]:

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
Dependents	1.000000	0.137819	-0.003428	0.149586	-0.087534	-0.057913
ApplicantIncome	0.137819	1.000000	-0.114247	0.551811	-0.023089	0.023378
CoapplicantIncome	-0.003428	-0.114247	1.000000	0.179228	-0.043860	-0.027253
LoanAmount	0.149586	0.551811	0.179228	1.000000	0.055636	-0.008235
Loan_Amount_Term	-0.087534	-0.023089	-0.043860	0.055636	1.000000	-0.020439
Credit_History	-0.057913	0.023378	-0.027253	-0.008235	-0.020439	1.000000

In [100]:

```
#all diagonal elements are 1 because variables are realted to itself completely
```

In [101]:

```
#co-rel(x,y)=co-rel(y,x)
```

In [9]:

```
#       SAMPLING
 #   divide the data in training data and the testing data
import pandas as pd
```

In [10]:

```
from sklearn.model_selection import train_test_split
```

In [11]:

```
cr =pd.read_csv(r"C:\Users\nb291\Desktop\CreditRisk - CreditRisk.csv")
```

In [12]:

```
cr_x = cr.iloc[ : ,1:12] #selecting all x variables
```

In [13]:

```
cr_y = cr.iloc[:,-1]   #selecting target variable
```

In [14]:

```
# when ever you build a predictive model , always divide the data into trail/test

# train data build the model

# test data do the prediction and use the evaluation matrics to judge the model performan
ce

# sampling should always be a random sampling

# any row of the original can be part of either train or test

# majority data should be Train

# less data in testing

# train : test
#  80      20
#  75      25
#  83      17
#  70      30
#  90      10
```

In [15]:

```
cr_x_train , cr_x_test , cr_y_train , cr_y_test   =   train_test_split(cr_x , cr_y ,test
_size = .2)
```

In [16]:

```
print(cr_x_train.shape)
print(cr_y_train.shape)
print("----")
print(cr_x_test.shape)
print(cr_y_test.shape)
```

```
(784, 11)
(784,)
----
(197, 11)
(197,)
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [141]:

```
# PLOTS
```

In [48]:

```
import matplotlib.pyplot as plt
```

In [49]:

```
import seaborn as sns
```

In [50]:

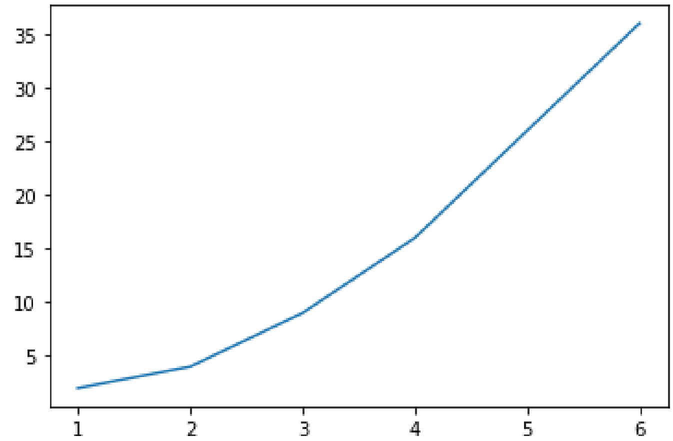
```
x  = [1,2,3,4,6]
y  = [2,4,9,16,36]
```

In [51]:

```
plt.plot(x,y)
```

Out[51]:

```
[<matplotlib.lines.Line2D at 0x1953b4725c8>]
```

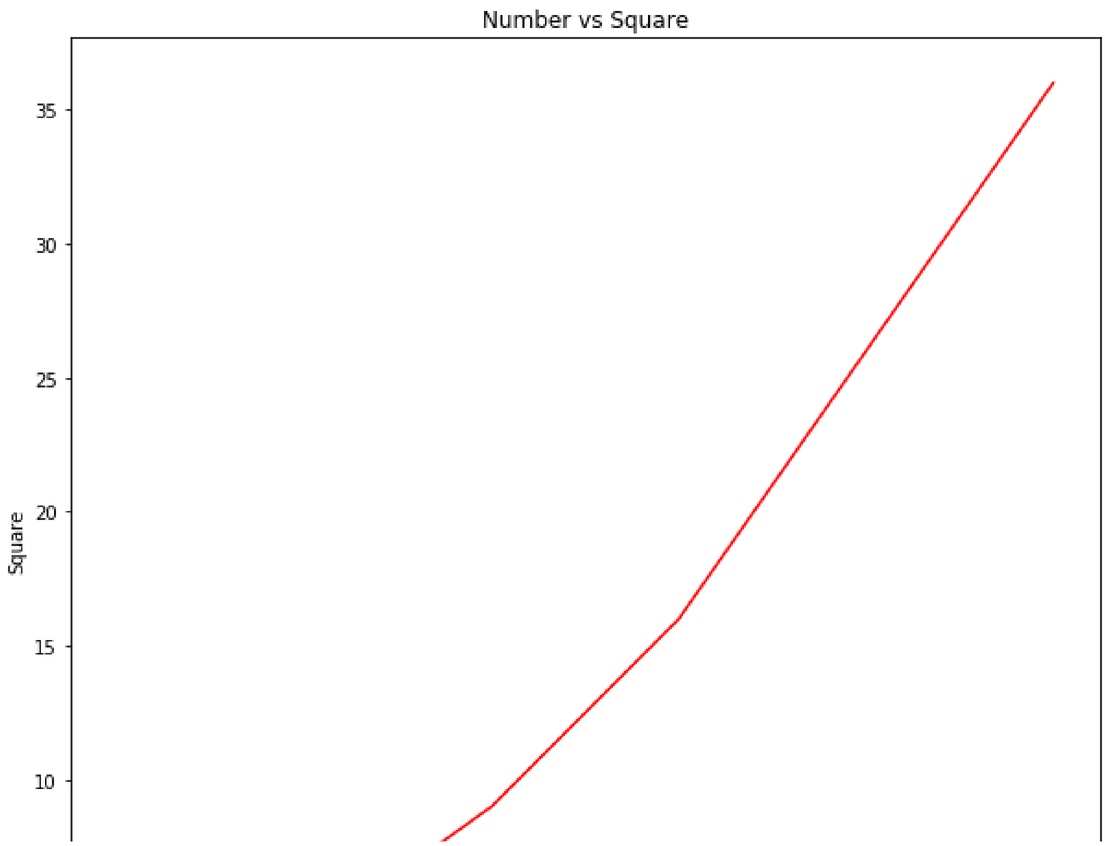


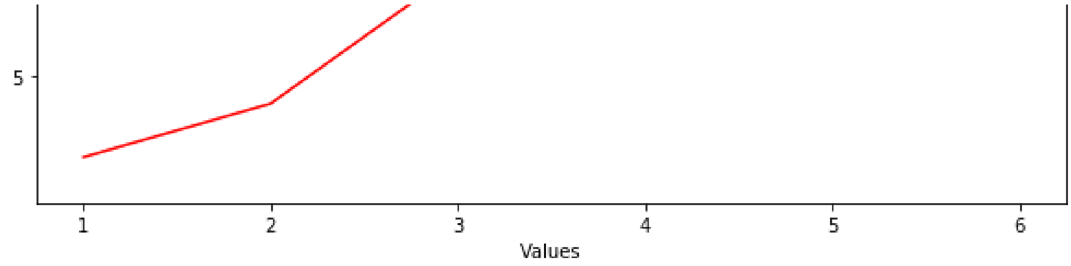
In [52]:

```
plt.figure(figsize = (10,10))  #size is changed  (figure command is always before plot)
plt.plot(x,y, color = "r")  #color is red "r"
plt.xlabel("Values")   #labelling the x,y axis
plt.ylabel("Square")
plt.title("Number vs Square")
```

Out[52]:

```
Text(0.5, 1.0, 'Number vs Square')
```



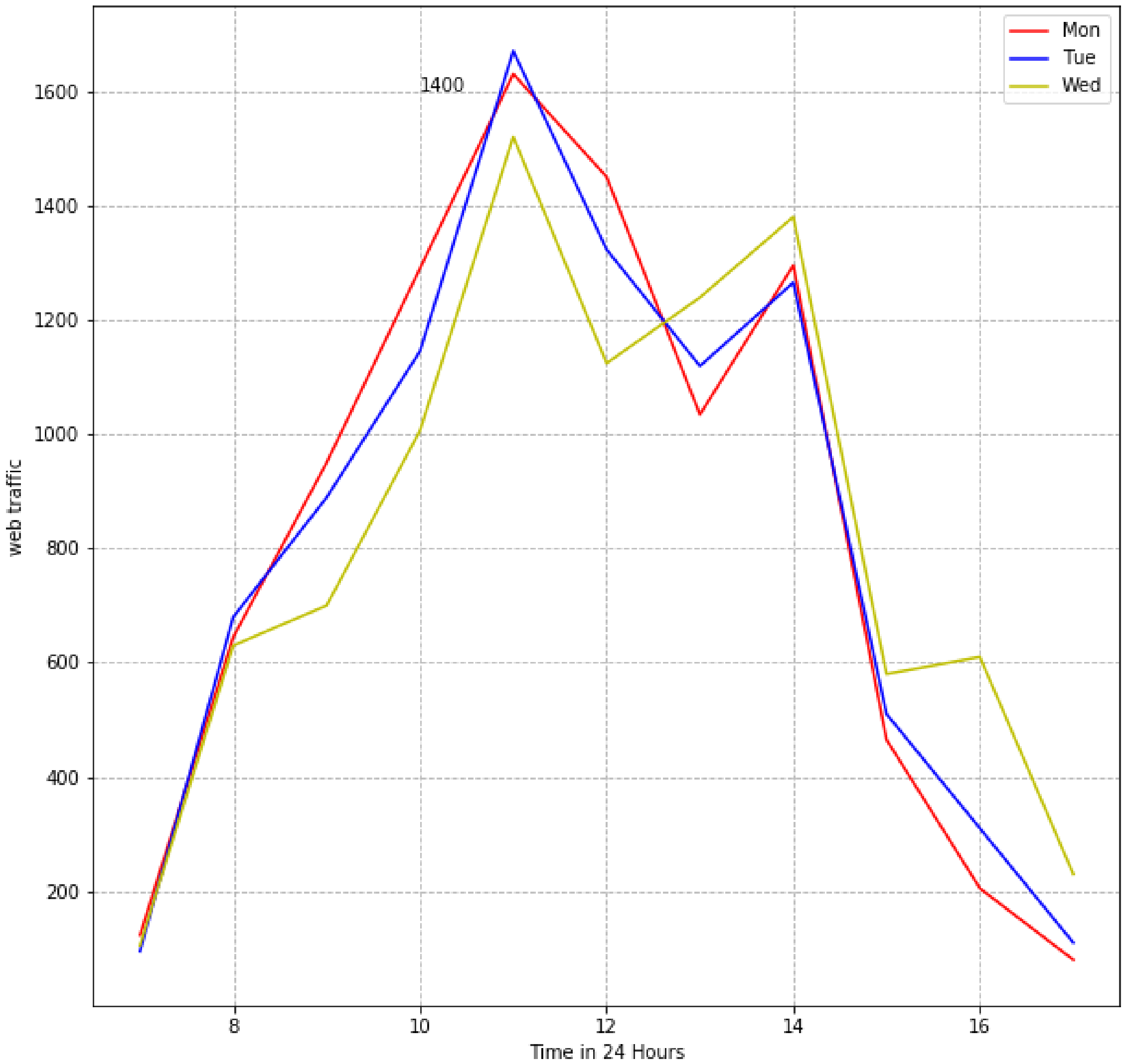


In [53]:

```
web_monday     = [123,645,950,1290,1630,1450,1034,1295,465,205,80]
web_tuseday    = [95,680,889,1145,1670,1323,1119,1265,510,310,110]
web_wednesday =  [ 105,630,700,1006,1520,1124,1239,1380,580,610,230]
time_hrs       =  [7,8,9,10,11,12,13,14,15,16,17]




#  ABOVE IS THE DATA AND BELOW IS THE CODE
plt.figure(figsize= (10   , 10 ))
plt.plot(time_hrs , web_monday , color = 'r' , label = "Mon" )
plt.plot(time_hrs , web_tuseday , color = 'b' , label = "Tue" )
plt.plot(time_hrs , web_wednesday , color = 'y' , label = "Wed" )
plt.xlabel("Time in 24 Hours")
plt.ylabel('web traffic')
plt.legend()
plt.grid(linestyle = "--")

plt.text(10, 1600,1400)
```

Out[53]:

```
Text(10, 1600, '1400')
```

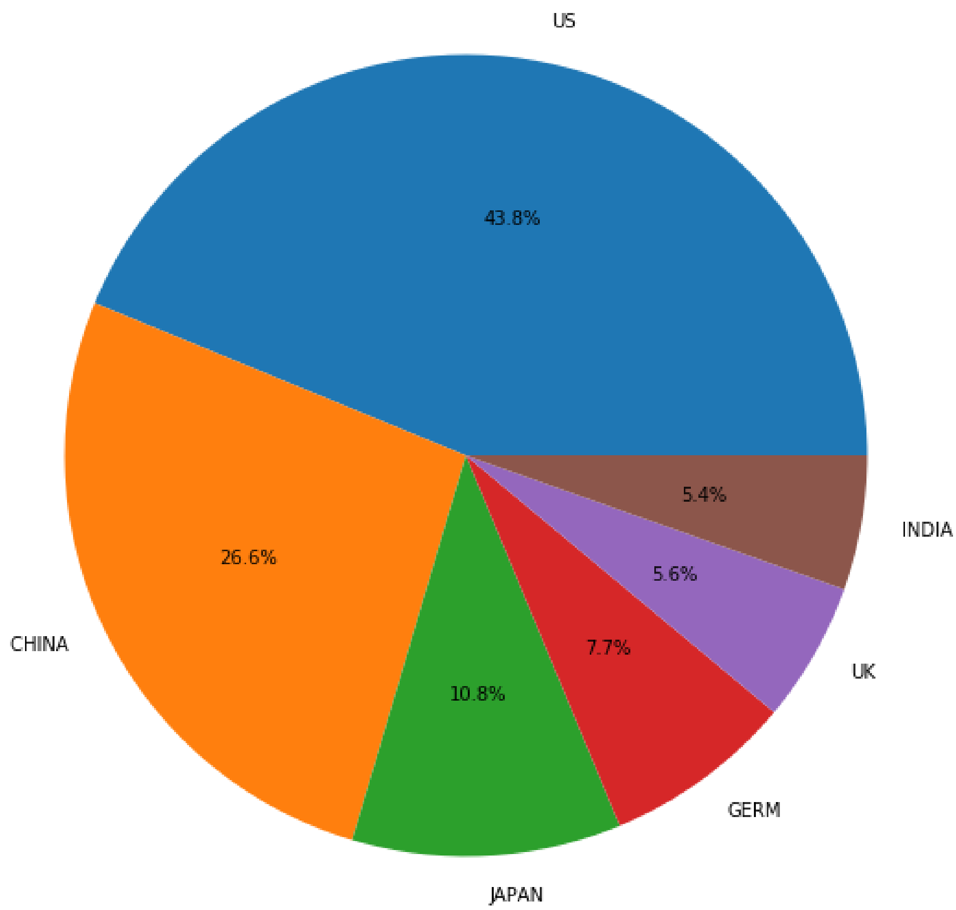


In [54]:

```
GDP         =    (19.4,   11.8 ,    4.8,     3.4,    2.5,   2.4)
countries   =   ('US' ,' CHINA', 'JAPAN','GERM','UK', 'INDIA')
plt.figure(figsize= (10   , 10 ))
plt.pie(  GDP , labels = countries  , autopct=     '%.1f%%'     )
```

Out[54]:

```
([<matplotlib.patches.Wedge at 0x1953b710f48>,
  <matplotlib.patches.Wedge at 0x1953b71c688>,
  <matplotlib.patches.Wedge at 0x1953b71cf08>,
  <matplotlib.patches.Wedge at 0x1953b723888>,
  <matplotlib.patches.Wedge at 0x1953b72c4c8>,
  <matplotlib.patches.Wedge at 0x1953b72cec8>],
 [Text(0.21316471456361924, 1.0791481846646507, 'US'),
  Text(-0.9920308589942705, -0.47526284811995345, ' CHINA'),
  Text(0.05847809043212525, -1.098444496977163, 'JAPAN'),
  Text(0.6522301842354419, -0.8857741172399437, 'GERM'),
  Text(0.9558614433705955, -0.544361002531851, 'UK'),
  Text(1.084106096082776, -0.1863168603110388, 'INDIA')],
 [Text(0.11627166248924685, 0.5886262825443549, '43.8%'),
  Text(-0.541107741269602, -0.2592342807927019, '26.6%'),
  Text(0.031897140235704675, -0.5991515438057251, '10.8%'),
  Text(0.35576191867387735, -0.4831495184945147, '7.7%'),
  Text(0.5213789691112339, -0.29692418319919145, '5.6%'),
  Text(0.5913305978633322, -0.10162737835147571, '5.4%')])
```



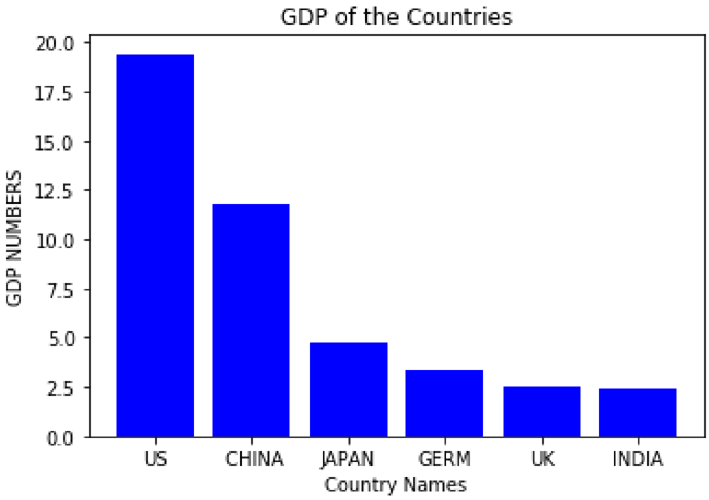
In [55]:

```
# 15-07-2020



#create a bar plot

plt.bar(countries , GDP  ,  color = 'b' )
```

```
plt.xlabel('Country Names')
plt.ylabel('GDP NUMBERS')
plt.title("GDP of the Countries")
```

Out[55]:

Text(0.5, 1.0, 'GDP of the Countries')



In [56]:

```
#  BOX plot : 5 data points  :-  min ...q1..m...q3 ....max

    # used to find outliers ..
    #  outliers:
    # they are observation which is not part of data but for some reason are included in our data
    # they are extreme points , may be on the higher or lower side of the mean

    # outlier on higher side : values greater than q3+1.5(Iqr) {inter quartile range}
    #outliers on lower side : values less than q1-1.5(iqr)
    #iqr=q3-q1
    # m can also be an outlier
```

In [57]:

```
#study normal distribution
```

In [58]:

```
# it is symmetric around mean
# mean= median= mode
#skewness = 0
```

In [59]:

```
import pandas as pd
lcn =pd.read_csv(r"C:\Users\nb291\Desktop\LungCapData - LungCapData.csv")
```

In [60]:

```
lcn.head()
```

Out[60]:

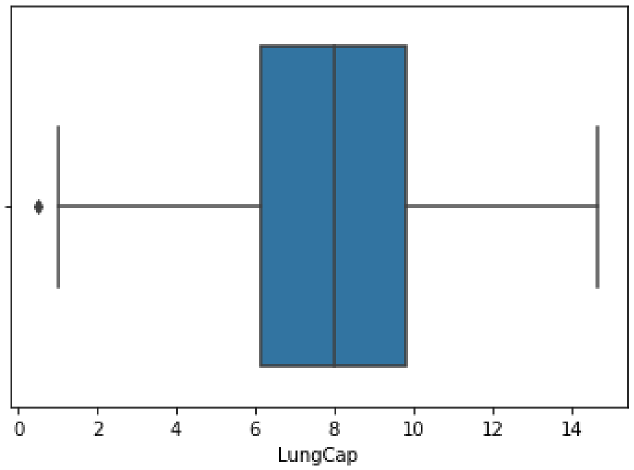
	LungCap	Age	Height	Smoke	Gender	Caesarean
0	6.475	6	62.1	no	male	no
1	10.125	18	74.7	yes	female	no
2	9.550	16	69.7	no	female	yes
3	11.125	14	71.0	no	male	no
4	4.800	5	56.9	no	male	no

In [61]:

```
sns.boxplot(lcn.LungCap )
```

Out[61]:

<matplotlib.axes._subplots.AxesSubplot at 0x1953b9057c8>

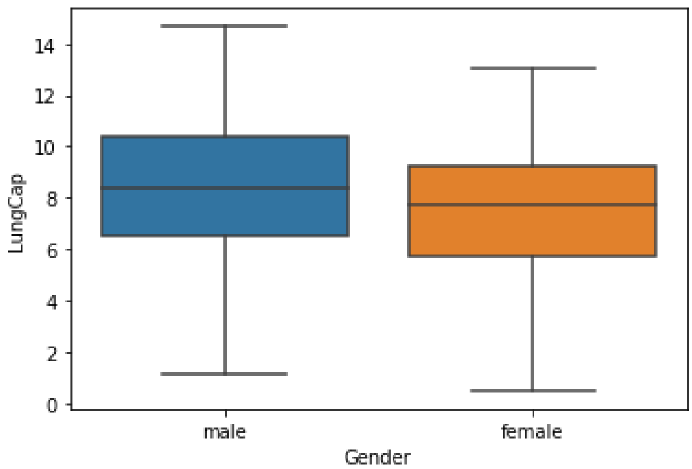


In [62]:

```
sns.boxplot( x= "Gender"  , y  = "LungCap"  , data =lcn )
```

Out[62]:

<matplotlib.axes._subplots.AxesSubplot at 0x1953bc988c8>

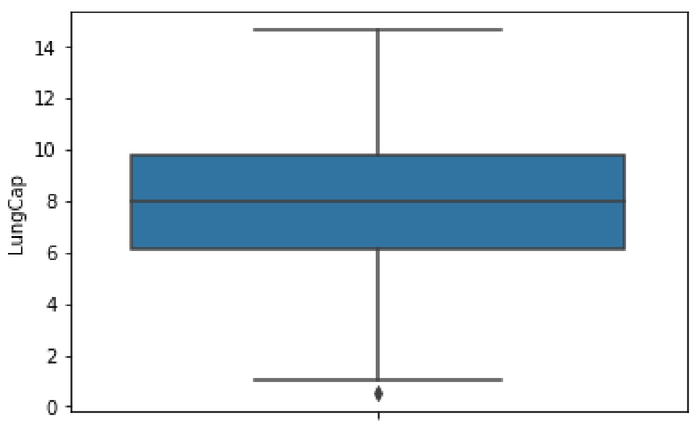


In [63]:

```
sns.boxplot(lcn.LungCap  , orient ="v")
```

Out[63]:

<matplotlib.axes._subplots.AxesSubplot at 0x1953bca88c8>



In [64]:

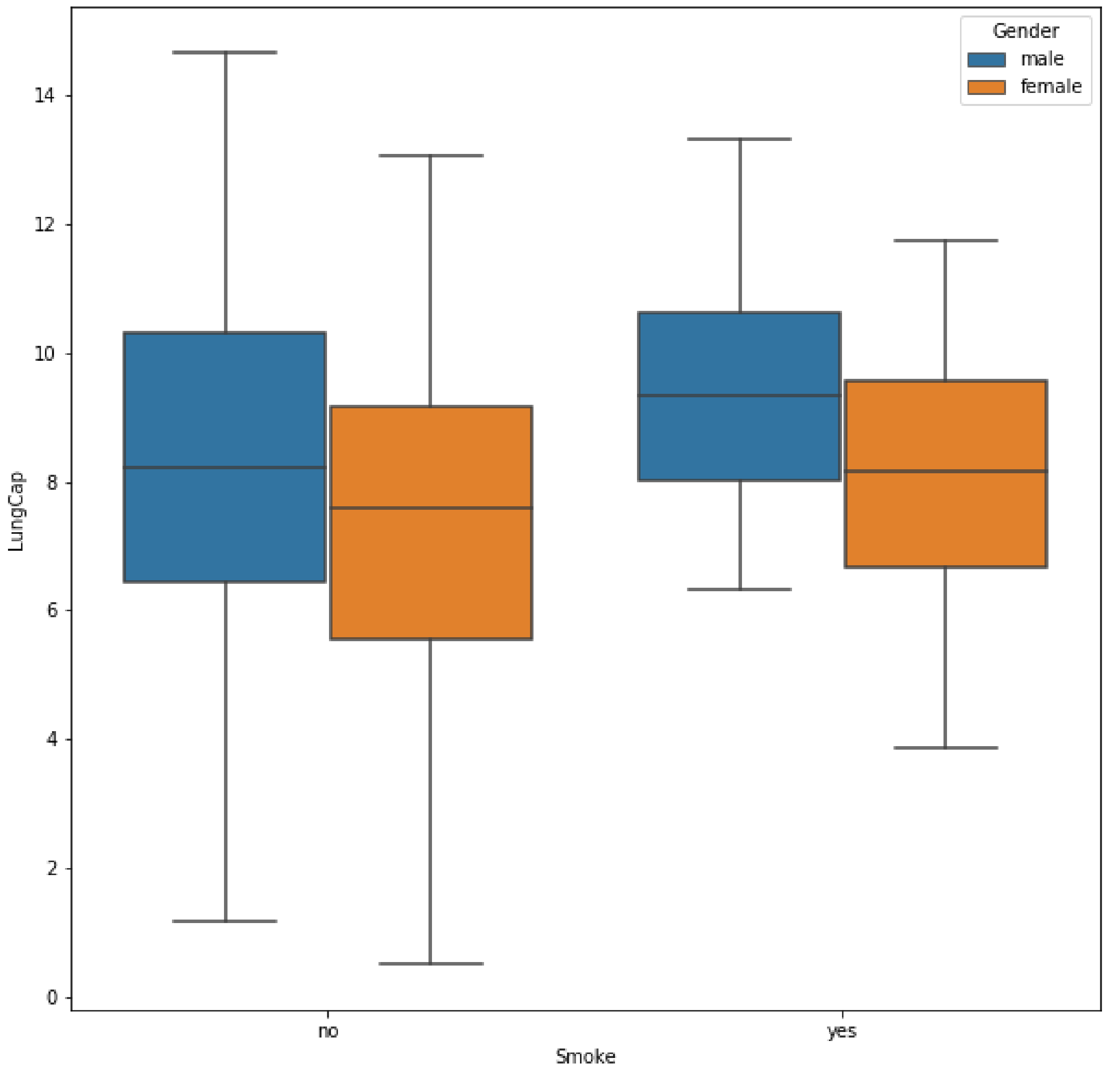
```
# dot is outlier
```

In [65]:

```
plt.figure(figsize=(10,10))
sns.boxplot(x= "Smoke" , y = "LungCap" , data = lcn, hue = "Gender")
```

Out[65]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1953bd8e6c8>
```



In [66]:

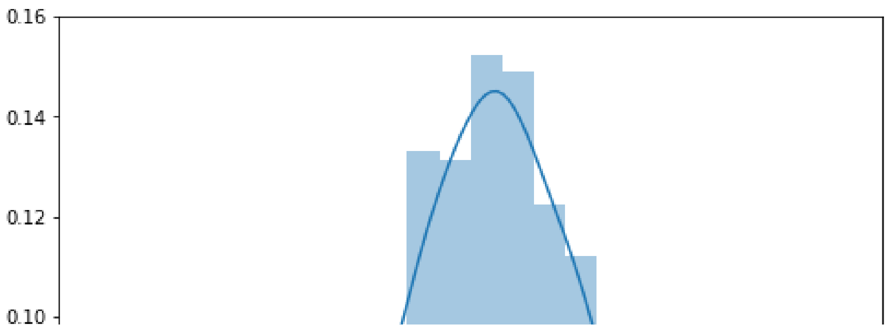
```
# build a df                    HOMEWORK
# where people are more than or equal to 17 and again build this plot
```

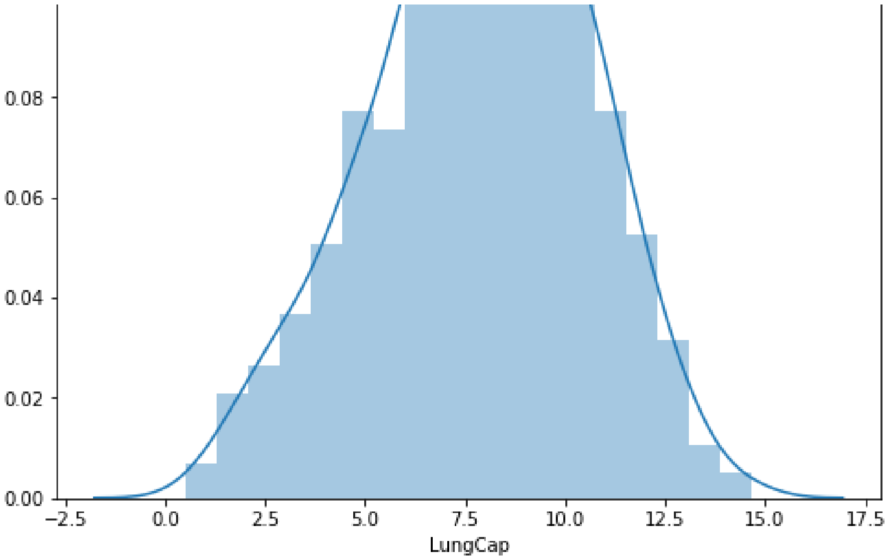
In [67]:

```
plt.figure(figsize= (8 , 8))
sns.distplot(lcn.LungCap)
```

Out[67]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1953c041148>
```



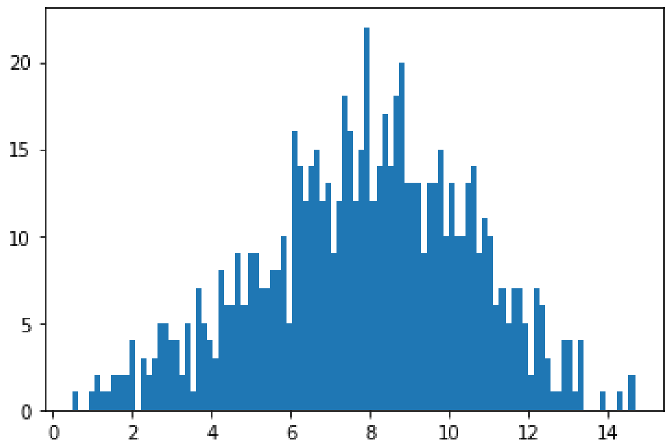


In [68]:

```
plt.hist(lcn.LungCap , bins = 100)
```

Out[68]:

```
(array([ 1.,  0.,  0.,  1.,  2.,  1.,  1.,  2.,  2.,  2.,  4.,  0.,  3.,
         2.,  3.,  5.,  5.,  4.,  4.,  2.,  5.,  1.,  7.,  5.,  4.,  3.,
         8.,  6.,  6.,  9.,  6.,  9.,  9.,  7.,  7.,  8.,  8., 10.,  5.,
        16., 14., 12., 14., 15., 12., 13.,  9., 12., 18., 16., 12., 15.,
        22., 12., 14., 17., 14., 18., 20., 13., 13., 13.,  9., 13., 13.,
        15., 10., 13., 10., 10., 13., 14.,  9., 11., 10.,  6.,  7.,  5.,
         7.,  7.,  5.,  2.,  7.,  6.,  3.,  1.,  1.,  4.,  4.,  1.,  4.,
         0.,  0.,  0.,  1.,  0.,  0.,  1.,  0.,  2.]),
 array([ 0.507   ,  0.64868,  0.79036,  0.93204,  1.07372,  1.2154 ,
         1.35708,  1.49876,  1.64044,  1.78212,  1.9238 ,  2.06548,
         2.20716,  2.34884,  2.49052,  2.6322 ,  2.77388,  2.91556,
         3.05724,  3.19892,  3.3406 ,  3.48228,  3.62396,  3.76564,
         3.90732,  4.049   ,  4.19068,  4.33236,  4.47404,  4.61572,
         4.7574 ,  4.89908,  5.04076,  5.18244,  5.32412,  5.4658 ,
         5.60748,  5.74916,  5.89084,  6.03252,  6.1742 ,  6.31588,
         6.45756,  6.59924,  6.74092,  6.8826 ,  7.02428,  7.16596,
         7.30764,  7.44932,  7.591   ,  7.73268,  7.87436,  8.01604,
         8.15772,  8.2994 ,  8.44108,  8.58276,  8.72444,  8.86612,
         9.0078 ,  9.14948,  9.29116,  9.43284,  9.57452,  9.7162 ,
         9.85788,  9.99956, 10.14124, 10.28292, 10.4246 , 10.56628,
        10.70796, 10.84964, 10.99132, 11.133   , 11.27468, 11.41636,
        11.55804, 11.69972, 11.8414 , 11.98308, 12.12476, 12.26644,
        12.40812, 12.5498 , 12.69148, 12.83316, 12.97484, 13.11652,
        13.2582 , 13.39988, 13.54156, 13.68324, 13.82492, 13.9666 ,
        14.10828, 14.24996, 14.39164, 14.53332, 14.675  ]),
 <a list of 100 Patch objects>)
```



In [69]:

```
#16-07-2020
```

In [70]:

```
import pandas as pd
lcn =pd.read_csv(r"C:\Users\nb291\Desktop\LungCapData - LungCapData.csv")
```

In [71]:

```
lcn.shape
```

Out[71]:

```
(725, 6)
```

In [72]:

```
lcn.head()
```

Out[72]:

	LungCap	Age	Height	Smoke	Gender	Caesarean
0	6.475	6	62.1	no	male	no
1	10.125	18	74.7	yes	female	no
2	9.550	16	69.7	no	female	yes
3	11.125	14	71.0	no	male	no
4	4.800	5	56.9	no	male	no

In [73]:

```
# STEPS INVOLVED IN MODEL BUILDING
```

In [74]:

```
# 1.Understand the problem statement, define the target variable
# 2. acquiring the data /fetch the data
# 3. data cleaning /data prepration
# 4. 4 plots /exploratory data anlysis
# 5. sampling (divide the data into train and test)
# 6. for my building the  model ( train the model)
# 7. test the model performance
```

In [75]:

```
lcn.isnull().sum()
```

Out[75]:

```
LungCap      0
Age          0
Height       0
Smoke        0
Gender       0
Caesarean    0
dtype: int64
```

In [76]:

```
lcn.corr()
```

Out[76]:

	LungCap	Age	Height
LungCap	1.000000	0.819675	0.912187
Age	0.819675	1.000000	0.835737
Height	0.912187	0.835737	1.000000

In [77]:

In [77]:

```
#to convert non numeric and numeric
```

In [78]:

```
lcn.Gender.replace({"male" :1 , "female":0} ,inplace = True)
lcn.Smoke.replace({"no" : 0 , "yes" :1} , inplace = True)
lcn.Caesarean.replace({"no" : 0 , "yes" :1} , inplace = True)
```

In [79]:

```
#sampling


import sklearn
from sklearn.model_selection import train_test_split
```

In [80]:

```
lcn_x  = lcn.iloc[: , [1,2,3,4,5]]
lcn_y = lcn.iloc[: , 0]
# divide the data into the x and y
```

In [81]:

```
lcn_x_train , lcn_x_test , lcn_y_train, lcn_y_test = train_test_split(lcn_x , lcn_y , tr
ain_size = .8)
```

In [82]:

```
print(lcn_x_train.shape)
print(lcn_y_train.shape)
print("------")
print(lcn_x_test.shape)
print(lcn_y_test.shape)
```

```
(580, 5)
(580,)
------
(145, 5)
(145,)
```

In [83]:

```
# for building a model in python always use sklearn





#in python --> SKLEARN, import necessary module
# create a object of it
#run the "fit " method to train the model
# run the " predict" method to predict it
```

In [84]:

```
from sklearn import linear_model      # model has been created
reg = linear_model.LinearRegression() #reg is object
#IMPORTING LINEAR_MODEL BECAUSE WE ARE BUILDING A LINEAR EQUATION
```

In [85]:

```
reg.fit(lcn_x_train,lcn_y_train )  #your model has been created
 #FIT IS AN INBUILT FUNCTION : TO MAKE A COMPILER NDERTAND THAT WE ARE BUILDING A MODEL


#equation :
# lungcap = B0 + B1.AGE + B2.HEIGHT +B3.SMOKE + B4.GENDER +B5.CESCAERIAN
```

Out[85]:

Out[85]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [86]:

```
reg.intercept_   # WHERE LINE CUTS THE Y AXIS (B0 VALUE )
```

Out[86]:

```
-11.282851019145955
```

In [87]:

```
reg.coef_ #(B1,B2.....B5 VALUES)
```

Out[87]:

```
array([ 0.16857646,  0.26193564, -0.58521342,  0.42384389, -0.20604558])
```

In [88]:

```
lcn_x_train.columns
```

Out[88]:

```
Index(['Age', 'Height', 'Smoke', 'Gender', 'Caesarean'], dtype='object')
```

In [89]:

```
reg.score(lcn_x_train, lcn_y_train) # this is r square value
```

Out[89]:

```
0.8498365033518073
```

In [90]:

```
pred_train = reg.predict(lcn_x_train)    # for prediction only x data is neededd
```

In [91]:

```
pred_train
```

Out[91]:

```
array([ 9.72031032,  9.7558303 ,  7.80685532, 10.06082666,  3.44262034,
        6.78550445,  5.92428211,  5.44949316, 11.6554688 , 11.05287731,
       11.02578039,  8.9526453 ,  7.68730257,  5.78342216,  7.31728788,
        5.68854003,  6.26053167,  5.60995934,  7.01451969,  4.41652908,
        9.77606087,  8.00967699,  9.95947445,  7.58391175,  8.36847475,
        3.95646   ,  9.91850239,  9.04264107, 10.18512627,  7.14865279,
        7.06676731,  8.86615247,  9.8548397 ,  9.83090807,  8.11464939,
        8.56799163,  7.35153309,  5.96189075,  7.34348144,  4.83226269,
       11.36720008,  5.59518085,  7.58589173, 10.9324212 ,  8.27972291,
        7.81504648,  9.62364634,  2.72748197,  4.43935922,  9.50607358,
        7.44014542,  9.06216642, 11.58005338,  8.98694915,  5.67376154,
        2.2719616 ,  6.35493372,  5.83338299, 11.64735852,  2.53271194,
        9.24888479,  9.71135532,  8.72040616,  9.83901835,  3.60452859,
        8.37638689,  8.0826661 ,  5.9061402 , 10.77279975,  6.91296934,
        5.70331852,  8.49930306, 10.95951812,  8.24561721, 10.84010488,
       10.81377181,  6.46431533,  7.43347721,  6.95071749,  8.89695338,
        5.51906022,  6.03380323,  7.56774982,  9.56987578,  6.87199728,
        8.08845582,  7.27156896,  2.79939445,  9.50517022,  9.71700553,
        7.88893894,  3.10210401,  6.02569295, 10.44245456, 10.97429661,
        8.63832253,  7.10101252, 10.36037093,  8.3880001 ,  4.70796308,
       11.27630096, 11.79434876,  3.36067623,  9.98230459,  6.01673794,
        6.45626368,  9.48462687, 11.75674012,  9.48462687,  7.09310038,
        8.06542754,  7.80823876, 11.37386829,  8.2192255 ,  7.84110053,
        5.76712072,  4.32192597,  8.07704075,  8.35025196, 12.40821581,
        9.8161882 ,  9.14411054,  8.19323008,  6.34597872, 11.61435723,
        6.85721879,  4.00884712,  7.78066175,  4.25600427,  6.41865504,
       11.56527489,  5.92351827,  6.59528315,  7.31392446, 10.26384647,
       10.50948067,  6.41859641,  7.01768497,  9.12122176, 10.57328287,
```

```
 2.9712757 ,  8.83224783, 11.05287731, 10.69283562,  8.67172302,
 7.69541285, 10.15088105,  5.00420257,  9.02455779,  8.02451411,
 5.80472936,  8.64168595,  3.65230837,  6.50192396,  5.71823653,
 3.45739884,  6.38098777, 13.60170469, 12.50157499,  6.57926073,
 8.1554233 , 10.83185509,  7.06340388, 11.31811774,  6.75911275,
 8.12256153,  8.64643281, 11.46050064,  5.12389483,  7.60205366,
 8.62235874,  7.09310038,  1.08856298, 10.73290432,  4.50177799,
 9.83181142, 11.73300663,  7.87065752,  7.62824723,  7.23079504,
 8.14751116,  9.42412946,  7.85726246,  7.82163381, 10.02321802,
 4.88801324,  7.971165  ,  8.1178733 ,  7.04738146,  9.83319486,
 3.72738613,  5.14658546,  8.42877402,  5.72614867,  8.50741334,
 5.43801946, 10.01426302,  7.97557129,  5.75708909,  8.1063996 ,
 2.58159614,  8.49607915,  8.1339766 ,  9.13950319,  4.98956359,
 6.35072557, 11.6257723 ,  5.13056304,  6.17954954, 10.27619866,
 8.21467678,  9.94133254,  8.00162534,  8.09973138,  9.74319909,
 8.31278283,  3.83552381,  8.28885119,  5.16136395,  3.11015566,
 6.47909382,  9.54718514,  9.1065019 ,  7.52677776,  3.26737567,
 6.31718557,  8.47785636, 10.80235674,  2.94632606, 10.60772622,
 7.78540861,  3.06587881,  2.36868421, 10.13610256,  7.76497991,
 5.72284388,  4.33670446,  5.77186758,  7.38705307,  2.55678602,
 9.85938842, 13.33976905, 10.81377181,  7.18982299,  9.17711182,
10.21253597,  5.87203449,  6.63501128,  7.17622979,  7.48916913,
 9.88888678,  5.17277903, 11.56443017, 10.49484169,  8.88217489,
10.22609832, 11.32622802,  9.47988001,  9.17717045,  7.26153733,
 6.53478574, 10.44251319,  5.68523524,  9.88097464,  6.48505682,
10.91518264,  7.68976263,  8.20781043,  8.12922974,  8.45166279,
10.63041686,  5.37681684,  8.60407732,  9.36507412,  4.86987133,
 7.04057374,  5.94711225,  7.23073641,  8.74646021,  6.7659791 ,
10.39329134, 11.58816367,  3.13634922,  3.28209554,  9.28985685,
 6.78194289,  8.5221332 ,  8.11128597, 11.41147693,  6.52811753,
11.00379497,  9.01644751,  4.46080593,  7.82968546,  5.35277055,
 8.13734002,  4.18409179,  8.24541907,  9.87662698,  7.08148717,
10.90713099,  9.78411252,  7.95734849,  5.95047567,  4.01689878,
11.15200135,  6.66239014, 10.15088105,  6.10763706,  9.10980669,
 7.20440334,  6.67380521,  6.7000574 ,  8.92314695,  8.51546499,
10.13065048,  9.91513897,  8.18972715, 12.12345002, 11.919725  ,
 6.56680279,  7.61118194,  8.53368779,  3.38686979,  7.23642039,
 7.08821401,  9.67939689, 12.1497831 , 10.16229612,  4.32653332,
 9.9675261 , 10.18374283, 11.94023458,  3.21162512,  5.11228162,
 8.04412034,  6.95402228,  5.38899575,  4.60318882,  5.80478799,
 8.60091204, 11.51288776, 10.89571592,  6.0337446 , 11.2590624 ,
 7.49253255,  6.9914914 ,  6.37307563,  4.28892468,  6.30715394,
 6.52337067, 10.06893694, 10.34895586, 12.01881528,  8.46974608,
 5.41518932,  6.48239861,  4.99272887,  3.50311775, 12.1200866 ,
11.73721477,  7.23746325,  3.39023321, 10.78757824,  7.75935456,
 9.46524103,  4.88464982,  7.4631737 ,  9.56320756, 10.29097715,
 9.86611526,  6.68878184,  8.24541907,  6.41865504, 12.46060294,
10.92402898,  7.84115916,  5.08747149,  9.25100428,  9.54368221,
 9.05993826, 11.55982282,  9.69081196,  9.34693221,  9.32410206,
 8.50280599,  6.20568447, 10.06419008,  4.65557595,  9.3979359 ,
 8.75787528,  7.2729524 ,  8.97092673,  3.10348745, 10.16091268,
 7.31728788,  7.53019981,  9.27171494, 10.43336004,  8.02115069,
10.71566576, 10.36723729,  4.41652908, 10.99251941,  2.81872166,
 7.11454708, 11.36720008,  9.14869009,  8.95614824,  6.34021386,
 9.8219193 ,  9.20330539,  9.255553  ,  7.07343552,  9.56651236,
 3.99743205,  6.1419409 ,  5.81283964,  3.92565908, 10.50541203,
 4.30028112,  7.95398507, 11.21670691,  3.85513004, 10.74858617,
 7.29109431,  6.28426517,  5.36280219, 10.29326394,  8.84787105,
 8.34460174, 11.55860668,  5.857256  ,  9.7684893 , 10.47992368,
 6.17143926,  5.78189922,  5.35957828,  8.2603957 ,  8.64643281,
 4.05648739, 12.40346895,  9.42082467,  6.1304672 ,  7.7308742 ,
 8.15898486,  7.02579525,  9.2864348 ,  6.84910851,  6.31382215,
 8.07690124,  9.44227138, 12.98797715, 12.40366709,  5.45616137,
 4.2445892 ,  7.51536269, 12.38202224, 10.11321378,  5.65904167,
 6.98007633,  6.02583246,  3.5259479 , 11.12354586,  7.26826417,
10.91854607, 10.26370696,  9.27844178, 12.3591921 , 10.02567809,
10.93999277,  8.0701744 ,  9.94800075, 11.15428814,  4.94845202,
 5.06800477,  6.23117282,  4.9027331 ,  6.32860065,  8.48116115,
 9.3218739 ,  6.75580795,  3.48042712,  3.22165676,  7.47354591,
 9.30457671,  8.78070543,  7.21245499,  5.66494604,  6.66835314,
 7.98354206,  2.88266337, 11.27714569,  8.32405839, 11.72718314,
 5.6999551 ,  9.22178787,  7.65444079,  7.6908055 ,  8.05070768,
```

```
        4.49841457,  9.43560316,  6.98007633, 13.02544628,  6.86408514,
        6.25826974, 12.1200866 , 11.67355208,  5.69540638, 11.18384513,
        8.48116115,  6.1419409 ,  7.70696743,  8.13643667,  7.24201197,
        9.28084322,  3.6375885 , 11.45719584, 10.37198415,  7.9310963 ,
       10.7158639 ,  9.88558199,  8.66697616,  6.40718134,  9.22599602,
        7.93445972, 12.46746929,  7.80012848,  7.95734849,  8.435699  ,
        5.49377001, 10.18394097,  5.54279372,  8.03923398,  8.97553408,
       10.18283947,  3.96999456,  7.56108161, 10.87619057,  9.42076604,
       11.37525173,  1.97578074, 11.05624073,  7.77399354, 10.7944446 ,
        2.1443572 ,  3.28559847,  5.19422573,  8.25683414,  8.98694915,
        9.84000258,  7.75130291,  9.99035624,  8.13259316,  9.30127192,
        6.22520983,  8.14064481,  8.94597709,  2.14772062,  9.45837468,
        2.61584136,  8.81026241,  6.96866126,  9.91039211, 10.88444036,
        5.00996743,  1.53385357,  8.84456626,  6.50542689, 10.334236  ,
       12.3888886 ,  8.50404992,  9.83649965,  7.16357079,  5.14652683,
        7.13407244,  6.65343513,  6.48576203,  7.19773513, 10.08702022])
```

In [92]:

```
pred_actual_df = pd.DataFrame()
pred_actual_df['Predicted'] = pred_train
pred_actual_df['actual'] = lcn_y_train   #Y IS TARGET VARIABLE
```

In [93]:

```
pred_actual_df
```

Out[93]:

	Predicted	actual
0	9.720310	6.475
1	9.755830	10.125
2	7.806855	9.550
3	10.060827	NaN
4	3.442620	4.800
...	...	...
575	7.134072	13.050
576	6.653435	NaN
577	6.485762	10.700
578	7.197735	NaN
579	10.087020	8.225

580 rows × 2 columns

In [94]:

```
Rsquare= reg.score(lcn_x_train , lcn_y_train)
k =5   # k is  5 because of 5 x variables
n =580 # number of records i hve build the model (train)

Adjrsquare = 1- (1-Rsquare)*(n-1)/(n-k-1)
```

In [95]:

```
Adjrsquare
```

Out[95]:

```
0.8485284589559171
```

In [96]:

```
error = pred_train - lcn_y_train   #ERROR IS ACTUAL - PREDICTED
```

In [97]:

```
import matplotlib.pyplot as plt
```

In [98]:

```
error
len(error)
```

Out[98]:

580

In [99]:

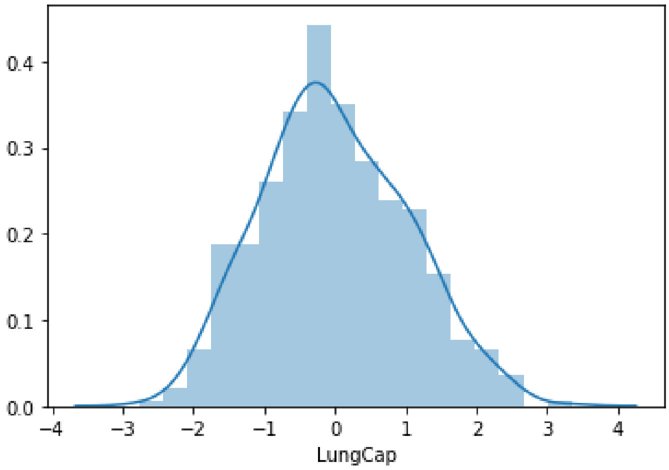
```
import seaborn as sns
```

In [100]:

```
sns.distplot(error)     #NORMAL DISTRIBUTION
#100% NORMAL DISTRIBUTION IS NOT POSSIBLE
```

Out[100]:

<matplotlib.axes._subplots.AxesSubplot at 0x1953c00c9c8>

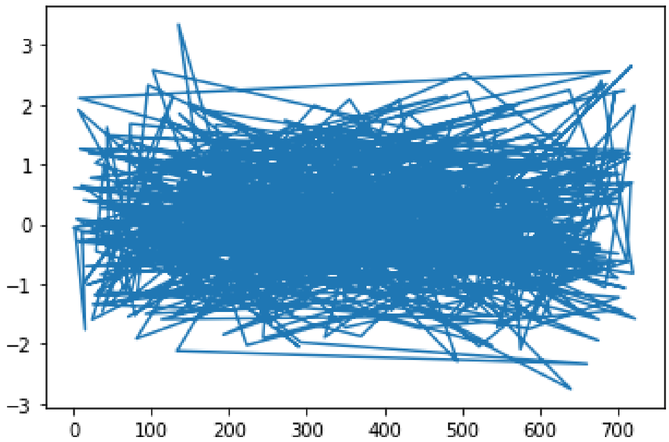


In [101]:

```
plt.plot(error)
# scattered plot
```

Out[101]:

[<matplotlib.lines.Line2D at 0x1953d285848>]



In [102]:

```
import numpy as np
np.mean(error)
```

Out[102]:

```
-1.0106857879346253e-16
```

In [103]:

```
history
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-103-47b4ab3daefe> in <module>
----> 1 history
      2
      3

NameError: name 'history' is not defined
```

In [106]:

```
pred_test = reg.predict(lcn_x_test)
```

In [107]:

```
error_test = lcn_y_test - pred_test  #  actual - predicted

MSE = np.mean(error_test * error_test)
MSE
```

Out[107]:

```
0.9380232253824625
```

In [108]:

```
np.power(36, .5)  # np.power is inbuilt function on numpy
```

Out[108]:

```
6.0
```

In [109]:

```
RMSE = np.power(MSE , .5)
RMSE
```

Out[109]:

```
0.9685159912889733
```

In [110]:

```
MAE = np.mean(np.absolute( error_test))
```

In [111]:

```
MAE
```

Out[111]:

```
0.7707859632401407
```

In [112]:

```
MAPE = np.mean(np.absolute ( error_test/lcn_y_test))* 100
```

In [113]:

```
MAPE
```

Out[113]:

```
13.049255405966772
```

In [114]:

```
accu = 100- MAPE
accu
```

Out[114]:

```
86.95074459403322
```

In [115]:

```
#import seaborn as sns
# import matplotlib.pyplot as plt
```

In [1]:

```
#sns.jointplot(x = 'Actual' , y = 'predicted' , kind = 'reg , data =pred_actual_df')
```

In [2]:

```
# PROPERTY PRICE
# MODEL
import pandas as pd
```

In [3]:

```
pa = pd.read_csv(r"C:\Users\nb291\Desktop\Property_Price_Train - Property_Price_Train.csv
")
```

In [4]:

```
pa.head()
```

Out[4]:

	Id	Building_Class	Zoning_Class	Lot_Extent	Lot_Size	Road_Type	Lane_Type	Property_Shape	Land_Outline	Utility_Type
0	1	60	RLD	65.0	8450	Paved	NaN	Reg	Lvl	AllPub
1	2	20	RLD	80.0	9600	Paved	NaN	Reg	Lvl	AllPub
2	3	60	RLD	68.0	11250	Paved	NaN	IR1	Lvl	AllPub
3	4	70	RLD	60.0	9550	Paved	NaN	IR1	Lvl	AllPub
4	5	60	RLD	84.0	14260	Paved	NaN	IR1	Lvl	AllPub

5 rows × 81 columns

In [5]:

```
#last column sale_price is the target variable
```

In [6]:

```
#unique identifiers never play any role in finding the outcome
```

In [7]:

```
pa = pa.drop(['Id'], axis=1)   #remove unique identifiers
```

In [12]:

```
pa.isnull().sum()
```

Out[12]:

```
Building_Class       0
Zoning_Class         0
Lot_Extent         259
Lot_Size             0
Road_Type            0
                  ...
```

```
Month_Sold          0
Year_Sold           0
Sale_Type           0
Sale_Condition      0
Sale_Price          0
Length: 80, dtype: int64
```

In [15]:

```
pa.fillna(  pa.median(),  inplace=True)
```

In [16]:

```
pa.isnull().sum()
```

Out[16]:

```
Building_Class    0
Zoning_Class      0
Lot_Extent        0
Lot_Size          0
Road_Type         0
                 ..
Month_Sold        0
Year_Sold         0
Sale_Type         0
Sale_Condition    0
Sale_Price        0
Length: 80, dtype: int64
```

In [18]:

```
pa.Basement_Height.fillna('TA'  , inplace= True)
```

In [19]:

```
pa.Exposure_Level.fillna('No'  , inplace= True)
```

In [20]:

```
pa.BsmtFinType1.fillna('Unf'  , inplace= True)
```

In [21]:

```
pa.BsmtFinType2.fillna('Unf'  , inplace= True)
```

In [22]:

```
pa.Electrical_System.fillna('sBrkr'  , inplace= True)
```

In [23]:

```
pa.Garage.fillna('Attchd'  , inplace= True)
```

In [24]:

```
pa.Garage_Finish_Year.fillna('Unf'  , inplace= True)
```

In [25]:

```
pa.Garage_Quality.fillna('TA'  , inplace= True)
```

In [26]:

```
pa.Garage_Condition.value_counts()
```

Out[26]:

```
TA    1325
Fa      35
Gd       9
```

```
Po        7
Ex        2
Name: Garage_Condition, dtype: int64
```

In [27]:

```
pa.Basement_Condition.fillna('TA'  , inplace= True)
```

In [29]:

```
pa.Brick_Veneer_Type.fillna('None'  , inplace= True)
```

In [30]:

```
#marking all the columns with a lot of null values as 1 because they have more null value
s than the actual values
#we can remove those columns too {another solution}
```

In [31]:

```
pa.Lane_Type = pa.Lane_Type.fillna("1")
```

In [32]:

```
pa.Garage_Condition = pa.Garage_Condition.fillna("1")
```

In [33]:

```
pa.Fireplace_Quality = pa.Fireplace_Quality.fillna("1")
```

In [34]:

```
pa.Pool_Quality = pa.Pool_Quality.fillna("1")
```

In [35]:

```
pa.Fence_Quality = pa.Fence_Quality.fillna("1")
```

In [36]:

```
pa.Miscellaneous_Feature = pa.Miscellaneous_Feature.fillna("1")
```

In [37]:

```
pa.Miscellaneous_Value = pa.Miscellaneous_Value.fillna("1")
```

In [38]:

```
from sklearn.preprocessing import LabelEncoder
```

In [39]:

```
le=LabelEncoder()
```

In [42]:

```
pa[pa.select_dtypes(include =['object']).columns]= pa[pa.select_dtypes(include=['object'
]).columns].apply(le.fit_transform)
```

In [43]:

```
pa_x= pa.drop (['Sale_Price']  , axis=1)
```

In [45]:

```
pa_y = pa.Sale_Price
```

In [46]:

```
#data cleaning is done
```

In [47]:

```
from sklearn.model_selection import train_test_split
```

In [50]:

```
pa_train_x , pa_test_x , pa_train_y , pa_test_y = train_test_split(pa_x,pa_y , test_size
= 0.1 , random_state=101)
```

In [51]:

```
from sklearn import linear_model
```

In [52]:

```
reg = linear_model.LinearRegression()
```

In [53]:

```
reg.fit(pa_train_x, pa_train_y)
```

Out[53]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [54]:

```
Rsquare = reg.score(pa_train_x  , pa_train_y)
print("value of Rsquare is--->", Rsquare)
```

```
value of Rsquare is---> 0.8667867725602165
```

In [55]:

```
pa_train_x.shape
```

Out[55]:

```
(1313, 79)
```

In [56]:

```
K=79
N=1313
AdjRsquare = 1-  (1-Rsquare)*(N-1)  /( N-K-1)
print("value of adjRsquare is ---->" , AdjRsquare)
```

```
value of adjRsquare is ----> 0.8582516184906764
```

In [57]:

```
reg.intercept_
```

Out[57]:

```
714724.0907488171
```

In [58]:

```
reg.coef_
```

Out[58]:

```
array([-8.93573799e+01, -1.99731514e+03, -1.25422142e+02,  4.15043460e-01,
        2.88658304e+04, -4.01978424e+03, -9.77888554e+02,  1.53279563e+03,
       -4.80346873e+04, -8.89836331e+01,  5.06411898e+03,  3.83145016e+02,
       -1.15746751e+03, -9.60202954e+03, -2.48073003e+03, -8.08863436e+02,
        1.00071890e+04,  5.92294443e+03,  2.20078820e+02,  1.37627854e+01,
        6.66263084e+02,  2.09634281e+04, -9.90906936e+02,  5.16362502e+02,
        5.03990497e+03,  3.89123567e+01, -7.87650160e+03,  9.85625901e+02,
        2.12500631e+03, -9.02000826e+03,  1.59436447e+03, -3.72581581e+03,
```

```
        2.12500651e+03,  9.02000820e+03,  1.59430447e+03,  3.72581381e+03,
       -4.65209356e+02,  8.34967158e+00,  1.66468658e+03,  1.19225003e+01,
       -4.46165726e+00,  1.58105145e+01, -3.75174908e+02, -1.01146695e+03,
        1.41852120e+03, -4.15742924e+02,  2.16323942e+01,  2.42273193e+01,
       -2.56229227e+01,  2.02367915e+01,  4.11234256e+03, -3.86017114e+03,
        2.86606499e+03, -2.02800014e+02, -3.75411346e+03, -1.40998949e+04,
       -7.28815544e+03,  4.04000181e+03,  1.89454969e+03,  8.80513803e+03,
       -1.39560763e+03,  2.99902188e+02, -2.33153334e+01, -1.26847322e+03,
        1.01600248e+04,  1.40290579e+00,  2.15717296e+02, -2.10898642e+03,
        2.03196086e+03, -7.53530184e+00,  1.99446817e+01,  1.74438790e+01,
        3.82559966e+01,  3.85613308e+01,  6.82196508e+02, -1.85927454e+05,
        9.28931611e+01, -1.20745352e+03,  6.29572474e-01, -1.53237770e+02,
       -6.00769494e+02, -6.40953919e+02,  3.34822589e+03])
```

In [59]:

```
coef_values =  pd.DataFrame({"Faeture_Names": pa_train_x.columns , "coeff": reg.coef_})
```

In [60]:

```
coef_values
```

Out[60]:

	Faeture_Names	coeff
0	Building_Class	-89.357380
1	Zoning_Class	-1997.315142
2	Lot_Extent	-125.422142
3	Lot_Size	0.415043
4	Road_Type	28865.830448
...	...	...
74	Miscellaneous_Value	0.629572
75	Month_Sold	-153.237770
76	Year_Sold	-600.769494
77	Sale_Type	-640.953919
78	Sale_Condition	3348.225894

79 rows × 2 columns

In [62]:

```
coef_values.sort_values( "coeff" , ascending = False)
```

Out[62]:

	Faeture_Names	coeff
4	Road_Type	28865.830448
21	Roof_Quality	20963.428084
60	Garage_Size	10160.024770
16	Overall_Material	10007.188962
55	Fireplaces	8805.138029
...	...	...
29	Basement_Height	-9020.008256
13	Condition2	-9602.029541
51	Kitchen_Above_Grade	-14099.894904
8	Utility_Type	-48034.687287
71	Pool_Quality	-185927.454225

79 rows × 2 columns

In [63]:

```
#this will show which column has higher weightage or impact on the outcome
```

In [64]:

```
#prediction ....error
```

In [69]:

```
pred_train = reg.predict(pa_train_x)
```

In [71]:

```
error_train = pa_train_y-pred_train
```

In [72]:

```
error_train
```

Out[72]:

```
902    -20376.393802
306    -21820.184298
1016   -18959.785978
1320   -21930.814759
677    -11433.967885
            ...
1417    30861.039177
75       3469.873833
599     -6094.633821
1361   -11736.026323
863     -1411.858182
Name: Sale_Price, Length: 1313, dtype: float64
```

In [73]:

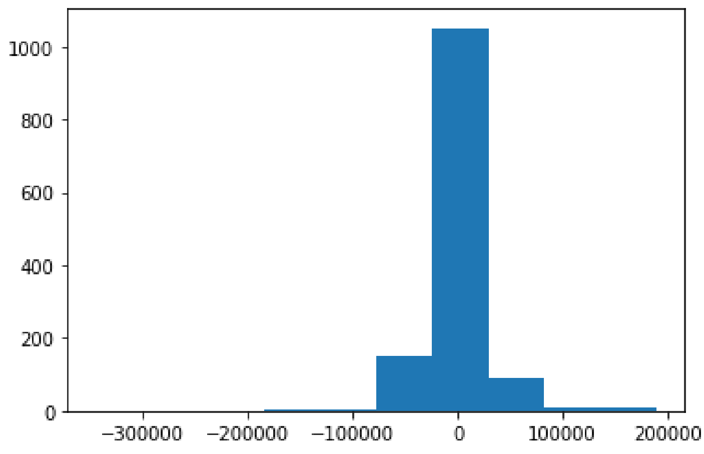
```
import matplotlib.pyplot as plt
```

In [74]:

```
plt.hist(error_train)
```

Out[74]:

```
(array([1.000e+00, 0.000e+00, 1.000e+00, 2.000e+00, 3.000e+00, 1.490e+02,
        1.051e+03, 8.900e+01, 1.000e+01, 7.000e+00]),
 array([-344542.32101958, -291166.43574423, -237790.55046888,
        -184414.66519353, -131038.77991818,  -77662.89464283,
         -24287.00936749,   29088.87590786,   82464.76118321,
         135840.64645856,  189216.53173391]),
 <a list of 10 Patch objects>)
```



In [75]:

```
#normally distributed
```

In [76]:

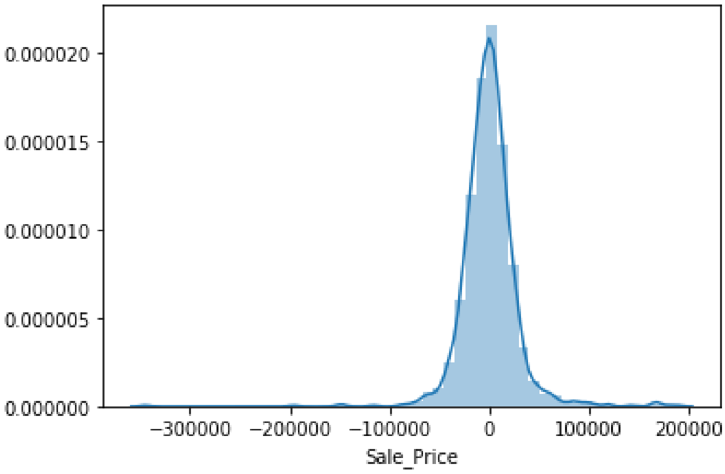
```
import seaborn as sns
```

In [77]:

```
sns.distplot(error_train)
```

Out[77]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x169c462ba48>
```



In [78]:

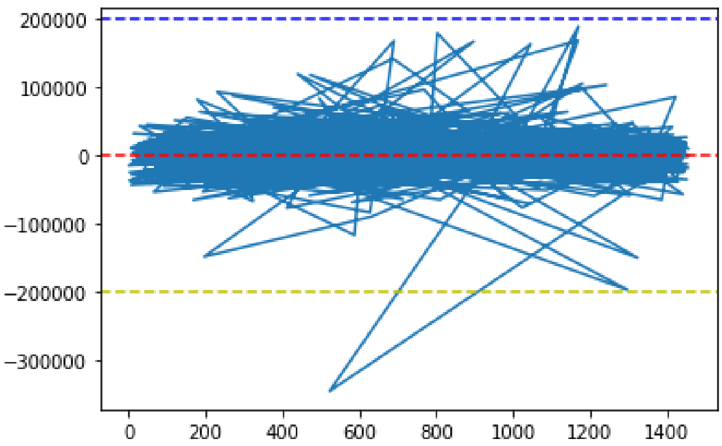
```
#normal distribution
```

In [80]:

```
plt.plot(error_train)
plt.axhline(y=0 , linestyle ='--' , color ='red')
plt.axhline(y=200000 , linestyle ='--', color='b')
plt.axhline(y = -200000 , linestyle = '--', color='y')
```

Out[80]:

```
<matplotlib.lines.Line2D at 0x169c4b20a08>
```



In [81]:

```
#higher side more error, lower side less error : understanding from graph
```

In [82]:

```
#predicion on the test data
pred_test = reg.predict(pa_test_x)
```

In [83]:

```
error = pa_test_y-pred_test
```

In [84]:

```
import numpy as np
```

In [85]:

```
MSE = np.mean(error*error)
MSE
```

Out[85]:

```
591296344.1000159
```

In [88]:

```
RMSE = np.power(MSE, .5)
```

In [89]:

```
RMSE
```

Out[89]:

```
24316.585782136764
```

In [91]:

```
MAPE = np.mean(np.absolute(error/pa_test_y))*100
```

In [92]:

```
MAPE
```

Out[92]:

```
11.015952289198594
```

In [94]:

```
#we are calculating MSE AND RMSE because when we will make changes in the data and then a
gain model them then we'll compare the values of mse and rmse of different data sets and
whichever is less will be the best
```

In [97]:

```
#how to chcek multi co-linearity :1. using corelation method
       # 2.using vif: variance inflation factor
        #0-5(no multi co linearity)
        #5-10 (may exist )
        #10 above( exists)
```

In [ ]:

```

```