Pathology Laboratory findings of COVID-19 patients.

Introduction

This project takes the datasets obtained from Kaggle.

Kaggle is a large, freely-available database comprising deidentified health-related data associated with over 40,000 patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2020 and 2021. This repository contains the code to derive the Hematology Complete Blood Count (CBC) as well as Biochemistry tests Dataset from the Kaggle dataset.

Derive the Hematology Complete Blood Count (CBC) as well as Biochemestry Tests Dataset from the Kaggle dataset. (Done)

This is a classification data.

Import all the liabraries -

Pandas - Used to analyse data. It has function for analysing,cleaning,exploring and manipulating data.

Numpy - Mostly work on numerical values for making Arithmatic Operations.

Matplotlib - Comprehensive library for creating static,animated and intractive visualization.

Seaborn - Seaborn is a python data visualization library based on matplotlib. It provides a high-level interface for drawing intractive and informative statastical graphics.

Warnings - warnings are provided to warn the developer of situation that are not necessarily exceptions and ignore them.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

By read_csv() function we are reading the dataset present in "Pathology Laboratory findings of COVID-19 patients..csv" file.

Took dataset from Kaggle

Will be making predictions models using Machine Learning Algorithams.

In [2]:
```
df=pd.read_csv("covid19.csv")
```

In [3]:
```
df
```

Out[3]:

	Unnamed: 0	Sex	Age	CA	CK	CREA	ALP	GGT	GLU	AST	...	MO	EO	BA	N
0	A00345_2020-03-25	1.0	82.0	2.09	NaN	1.150	95.0	40.0	78.0	26.0	...	9.5	2.9	0.5	6
1	A00791_2020-03-19	1.0	51.0	1.97	237.0	0.970	54.0	98.0	98.0	74.0	...	NaN	NaN	NaN	N
2	A00741_2020-03-04	1.0	58.0	2.11	NaN	1.000	80.0	147.0	106.0	41.0	...	7.3	0.3	0.1	5
3	A00605_2020-04-15	0.0	82.0	2.27	138.0	0.755	123.5	176.5	106.0	114.0	...	9.5	1.7	0.9	3
4	A00417_2020-02-24	1.0	79.0	2.07	73.0	1.810	62.0	36.5	96.0	28.0	...	10.0	8.5	0.5	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1731	49	0.0	NaN	2.38	40.0	0.800	68.0	9.0	128.0	13.0	...	NaN	NaN	NaN	N
1732	50	0.0	NaN	2.36	NaN	0.960	79.0	35.0	107.0	24.0	...	NaN	NaN	NaN	N
1733	51	1.0	NaN	2.28	NaN	1.420	NaN	NaN	136.0	53.0	...	NaN	NaN	NaN	N
1734	52	1.0	NaN	2.40	124.0	0.950	48.0	44.0	95.0	50.0	...	NaN	NaN	NaN	N
1735	53	1.0	NaN	2.14	NaN	1.040	NaN	NaN	179.0	28.0	...	NaN	NaN	NaN	N

1736 rows × 36 columns

Performing Exploratory Data Analysis

Find information about Data by using df.info()

This Covid19.csv dataset contains the information of patients health condition through their tests peroforms during covid19 with 1736 rows and 36 columns.

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1736 entries, 0 to 1735
Data columns (total 36 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  1736 non-null   object 
 1   Sex         1736 non-null   float64
 2   Age         1682 non-null   float64
 3   CA          1643 non-null   float64
 4   CK          704 non-null    float64
 5   CREA        1662 non-null   float64
 6   ALP         1262 non-null   float64
 7   GGT         1300 non-null   float64
 8   GLU         1638 non-null   float64
 9   AST         1638 non-null   float64
 10  ALT         1640 non-null   float64
 11  LDH         1433 non-null   float64
 12  PCR         1639 non-null   float64
 13  KAL         1656 non-null   float64
 14  NAT         1663 non-null   float64
 15  UREA        1060 non-null   float64
 16  WBC         1673 non-null   float64
 17  RBC         1673 non-null   float64
 18  HGB         1673 non-null   float64
 19  HCT         1673 non-null   float64
 20  MCV         1673 non-null   float64
 21  MCH         1673 non-null   float64
 22  MCHC        1673 non-null   float64
 23  PLT1        1673 non-null   float64
 24  NE          1374 non-null   float64
 25  LY          1374 non-null   float64
 26  MO          1374 non-null   float64
 27  EO          1374 non-null   float64
 28  BA          1374 non-null   float64
 29  NET         1374 non-null   float64
 30  LYT         1374 non-null   float64
 31  MOT         1374 non-null   float64
 32  EOT         1374 non-null   float64
 33  BAT         1374 non-null   float64
 34  Suspect     1736 non-null   float64
 35  target      1736 non-null   int64  
dtypes: float64(34), int64(1), object(1)
memory usage: 488.4+ KB
```

This dataset contain 1736 rows and 36 columns out of there are 35 numerical columns and 1 object columns.

To find how much null values in DataSet

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Unnamed: 0       0
Sex              0
Age             54
CA              93
CK            1032
CREA            74
ALP            474
GGT            436
GLU             98
AST             98
ALT             96
LDH            303
PCR             97
KAL             80
NAT             73
UREA           676
WBC             63
RBC             63
HGB             63
HCT             63
MCV             63
MCH             63
MCHC            63
PLT1            63
NE             362
LY             362
MO             362
EO             362
BA             362
NET            362
LYT            362
MOT            362
EOT            362
BAT            362
Suspect          0
target           0
dtype: int64
```

With the help of isnull().sum() function we got the null count of all columns.

Drop unwanted columns by using drop function

In [6]:
```
df.drop("CK",axis=1,inplace=True)
```

In [7]:
```
df.drop("Unnamed: 0",axis=1,inplace=True)
```

As the missing Data is more than 50% so we can drop the column "CK" and "Unnamed: 0" is unwanted column so we can drop it also.

Drop Null values

In [8]:
```
df.dropna(inplace=True)
```

Remaining columns consists null values are 0.5 % ,0.8 % so we can drop that row content null values.

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
Sex        0
Age        0
CA         0
CREA       0
ALP        0
GGT        0
GLU        0
AST        0
ALT        0
LDH        0
PCR        0
KAL        0
NAT        0
UREA       0
WBC        0
RBC        0
HGB        0
HCT        0
MCV        0
MCH        0
MCHC       0
PLT1       0
NE         0
LY         0
MO         0
EO         0
BA         0
NET        0
LYT        0
MOT        0
EOT        0
BAT        0
Suspect    0
target     0
dtype: int64
```

Now, our data does not contains any null value.

In [10]:
```
df
```

Out[10]:

	Sex	Age	CA	CREA	ALP	GGT	GLU	AST	ALT	LDH	...	MO
0	1.0	82.0	2.090000	1.150000	95.0	40.0	78.000000	26.000000	21.000000	307.0	...	9.50
3	0.0	82.0	2.270000	0.755000	123.5	176.5	106.000000	114.000000	63.000000	281.0	...	9.50
4	1.0	79.0	2.070000	1.810000	62.0	36.5	96.000000	28.000000	38.500000	264.0	...	10.00
5	1.0	84.0	2.060000	1.283333	75.0	75.0	95.500000	40.500000	27.000000	364.5	...	6.80
8	0.0	48.0	2.110000	0.660000	200.0	90.0	104.000000	38.000000	36.000000	189.0	...	9.00
...	...	...	...	...	...	...	...	...	...	...	...	...
1644	1.0	68.0	1.961070	0.688000	58.0	21.0	77.800000	12.200000	16.400000	185.0	...	7.44
1650	0.0	79.0	1.923645	0.509000	44.0	206.6	93.000000	46.300000	33.700000	422.3	...	4.30
1652	0.0	76.0	2.411833	0.750000	51.0	32.0	90.000000	25.333333	58.000000	233.0	...	8.00
1678	1.0	58.0	2.012633	0.736667	42.0	20.0	100.333333	21.000000	35.000000	242.5	...	11.80
1681	0.0	78.0	2.129067	0.483333	81.0	33.0	190.666667	21.000000	34.666667	303.0	...	10.00

826 rows × 34 columns

How many people get infected through covid and how many are not infected.

In [11]:
```
df["target"].value_counts()
```

Out[11]:
```
1    463
0    363
Name: target, dtype: int64
```

Here is 463 are infected peoples and 363 are not infected.

What is lowest Age of patient whoes admitted in hospital.

In [12]:
```
df["Age"].min()
```

Out[12]:
```
17.0
```

The smallest person was admitted in hospital during covid is 17 years old.

whoes age is less than 99 and its having target=0

In [13]:
```
df[(df["Age"]<99)&(df["target"]==0)]
```

Out[13]:

	Sex	Age	CA	CREA	ALP	GGT	GLU	AST	ALT	LDH	...	MO	EO	BA	NET
0	1.0	82.0	2.09	1.150000	95.0	40.0	78.000000	26.0	21.0	307.0	...	9.50	2.9	0.50	6.40
3	0.0	82.0	2.27	0.755000	123.5	176.5	106.000000	114.0	63.0	281.0	...	9.50	1.7	0.90	3.60
4	1.0	79.0	2.07	1.810000	62.0	36.5	96.000000	28.0	38.5	264.0	...	10.00	8.5	0.50	0.40
8	0.0	48.0	2.11	0.660000	200.0	90.0	104.000000	38.0	36.0	189.0	...	9.00	0.5	0.00	4.40
12	0.0	53.0	2.36	0.593333	62.0	21.5	91.666667	36.0	35.5	269.5	...	10.65	0.9	0.40	3.95
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1602	1.0	56.0	2.22	1.240000	38.5	12.0	102.000000	16.5	15.5	169.0	...	6.40	0.2	0.45	5.15
1606	0.0	52.0	2.35	0.520000	85.0	42.0	140.000000	27.0	34.0	196.0	...	12.40	3.2	0.70	4.90
1608	0.0	39.0	2.21	0.830000	26.0	11.0	79.000000	20.0	17.0	156.0	...	5.60	2.5	0.40	4.70
1610	1.0	74.0	2.48	0.940000	56.5	35.0	113.500000	34.0	40.0	335.0	...	10.55	3.0	0.90	6.00
1619	0.0	80.0	2.32	0.680000	95.0	21.5	113.000000	22.0	15.5	319.0	...	3.50	0.0	0.00	3.60

362 rows × 34 columns

There is 362 peoples are not infected according to age was less than 99 years old.

Whoes age is greater than 20 and having glucose level greater than 200.

In [14]:
```
df[(df["Age"]>20)&(df["GLU"]>200)]
```

Out[14]:

	Sex	Age	CA	CREA	ALP	GGT	GLU	AST	ALT	
13	0.0	76.0	2.200000	2.420000	159.500000	147.000000	640.000000	1018.500000	559.500000	69
31	1.0	78.0	1.700000	1.230000	70.000000	13.000000	278.000000	17.000000	10.000000	35
84	1.0	82.0	2.480000	1.360000	66.000000	21.000000	296.000000	42.000000	23.000000	30
95	1.0	84.0	2.370000	2.350000	73.000000	22.000000	265.000000	83.000000	52.000000	56
107	0.0	86.0	2.230000	2.070000	88.000000	52.000000	267.000000	244.000000	133.000000	44
111	0.0	66.0	2.210000	5.343333	51.000000	19.000000	282.666667	65.666667	24.333333	43
124	0.0	87.0	2.175000	1.395000	81.000000	26.000000	201.000000	45.500000	34.500000	26
125	1.0	66.0	2.370000	1.230000	66.000000	34.000000	450.000000	28.000000	27.000000	40
131	0.0	72.0	2.255000	1.600000	86.000000	146.000000	276.000000	66.000000	59.000000	31
150	1.0	76.0	2.010000	0.923333	62.000000	25.000000	289.000000	203.333333	65.333333	50
241	1.0	67.0	1.940000	1.120000	106.500000	84.000000	206.000000	281.000000	153.000000	98
257	1.0	63.0	2.030000	0.735000	85.000000	54.000000	295.000000	87.000000	63.000000	56
269	0.0	85.0	2.110000	1.190000	78.000000	22.000000	237.000000	53.000000	36.000000	43
277	0.0	73.0	2.090000	1.766000	65.666667	28.333333	260.000000	41.333333	18.000000	39
331	0.0	31.0	2.150000	0.590000	83.000000	75.000000	208.000000	101.000000	103.000000	31
402	1.0	46.0	2.090000	1.080000	49.000000	91.000000	258.500000	39.500000	40.500000	23
410	1.0	62.0	2.085000	2.385000	114.000000	64.000000	274.000000	38.000000	40.500000	23
411	1.0	75.0	2.090000	1.690000	56.000000	60.000000	459.000000	71.000000	59.000000	43
437	1.0	73.0	2.115000	2.590000	119.500000	17.000000	298.500000	18.000000	14.500000	30
441	1.0	44.0	2.300000	1.140000	45.000000	121.000000	207.000000	57.000000	52.000000	40
444	0.0	89.0	1.930000	0.885000	200.000000	377.000000	258.000000	105.000000	97.500000	74
471	0.0	90.0	2.075000	4.820000	96.000000	17.000000	309.000000	27.000000	21.000000	31
500	0.0	88.0	1.363333	2.405000	52.000000	39.000000	240.500000	356.500000	164.500000	99
524	0.0	69.0	2.290000	0.770000	82.000000	44.000000	282.500000	32.000000	21.000000	48
572	1.0	73.0	2.060000	1.470000	67.000000	50.500000	246.000000	75.500000	96.000000	59
700	0.0	43.0	1.930000	0.920000	57.000000	194.000000	450.000000	50.000000	38.000000	49
740	0.0	77.0	2.220000	1.600000	175.000000	17.000000	362.000000	18.000000	17.000000	33
799	0.0	41.0	2.060000	5.720000	201.000000	14.000000	271.000000	16.000000	11.000000	26
862	1.0	56.0	2.060000	1.105000	68.000000	56.000000	257.000000	57.000000	46.000000	56
874	1.0	55.0	1.850000	0.860000	102.000000	128.000000	232.000000	112.000000	77.000000	63
880	0.0	74.0	2.105000	0.660000	63.500000	49.000000	207.000000	34.500000	42.500000	51
921	0.0	82.0	1.985000	2.770000	212.333333	157.666667	386.000000	32.000000	32.333333	40
930	1.0	92.0	2.100000	1.400000	33.000000	35.000000	201.000000	53.000000	36.000000	39
1025	0.0	48.0	1.860000	0.555000	273.000000	58.000000	287.000000	48.500000	54.500000	57

	Sex	Age	CA	CREA	ALP	GGT	GLU	AST	ALT	
1070	0.0	52.0	2.100000	0.600000	101.000000	28.000000	205.000000	64.000000	86.000000	21
1073	1.0	79.0	2.255000	1.440000	63.000000	36.000000	223.000000	31.500000	19.500000	34
1081	0.0	69.0	2.290000	0.585000	66.000000	30.000000	243.500000	20.500000	27.000000	22
1135	1.0	61.0	2.793333	3.883333	96.000000	22.500000	204.000000	21.000000	18.000000	31
1153	1.0	76.0	1.985000	2.770000	212.333333	157.666667	386.000000	32.000000	32.333333	40
1169	0.0	70.0	2.160000	3.292500	72.500000	21.000000	269.500000	53.500000	58.500000	35
1215	1.0	81.0	2.190000	1.760000	45.000000	22.000000	311.000000	90.000000	48.000000	54
1221	0.0	72.0	2.095000	1.095000	60.000000	12.000000	279.500000	58.000000	58.500000	38
1225	0.0	86.0	2.020000	0.810000	176.000000	237.000000	204.000000	48.000000	52.000000	44
1258	1.0	85.0	2.165000	3.020000	64.000000	60.000000	206.000000	13.000000	11.000000	20
1272	1.0	75.0	2.075000	1.405000	128.500000	200.000000	288.500000	104.000000	46.500000	39
1291	1.0	71.0	1.980000	1.415000	68.000000	262.000000	219.000000	72.500000	51.000000	50
1345	1.0	74.0	1.950000	2.080000	92.000000	91.500000	226.000000	296.500000	126.500000	123
1373	1.0	68.0	2.150000	1.320000	60.000000	133.000000	270.000000	91.000000	50.000000	62
1402	1.0	55.0	2.190000	0.940000	129.000000	99.000000	216.000000	46.000000	108.000000	25
1404	1.0	75.0	2.240000	0.980000	579.000000	79.500000	383.000000	30.000000	19.000000	24
1517	1.0	76.0	2.060000	1.140000	78.500000	52.000000	257.500000	44.000000	55.000000	44
1570	0.0	37.0	2.335000	0.515000	77.000000	35.000000	270.000000	24.500000	26.000000	27
1581	1.0	62.0	2.065000	1.950000	60.500000	27.000000	255.500000	31.500000	22.000000	31
1585	1.0	86.0	2.200000	1.535000	108.000000	13.500000	855.000000	30.500000	15.000000	30
1586	1.0	61.0	2.195000	1.145000	96.000000	14.000000	210.500000	18.500000	22.500000	26

The 55 peoples having age greater than 20 and having high glucose level. i.e. >200

what is high and Low HB value.

In [15]:
```
df["HGB"].max()
```

Out[15]:
```
18.55
```

In [16]:
```
df["HGB"].min()
```

Out[16]:
```
6.65
```

Acoording to aur data the highest value of Hb is 18.55 and Lowest is 6.65

How many childerns are admitted in hospital

In [17]:
```
df[df["Age"]<18]
```

Out[17]:

	Sex	Age	CA	CREA	ALP	GGT	GLU	AST	ALT	LDH	...	MO
1391	1.0	17.0	2.283333	1.056667	61.0	36.5	87.5	28.333333	45.333333	234.666667	...	13.133333

1 rows × 34 columns

The only one child was admitted in Hospital during COVID 19

How many people are infected with covid

In [18]:
```
df[df["target"]==1]
```

Out[18]:

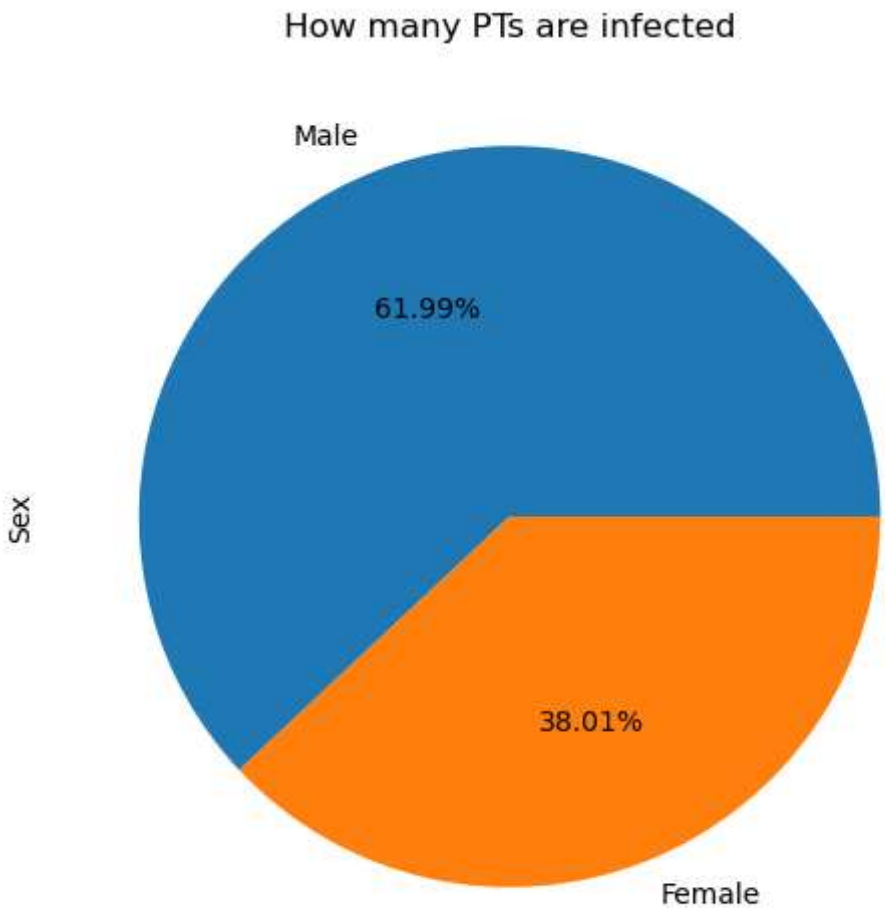
	Sex	Age	CA	CREA	ALP	GGT	GLU	AST	ALT	LDH	...	M
5	1.0	84.0	2.060000	1.283333	75.0	75.0	95.500000	40.500000	27.000000	364.5	...	6.8
9	0.0	67.0	1.980000	0.610000	47.0	23.0	106.000000	54.000000	27.000000	356.0	...	4.9
13	0.0	76.0	2.200000	2.420000	159.5	147.0	640.000000	1018.500000	559.500000	694.0	...	7.2
18	1.0	60.0	2.100000	2.380000	50.0	20.0	154.000000	131.000000	28.000000	775.0	...	5.5
23	1.0	85.0	1.940000	0.900000	44.0	33.0	101.000000	50.000000	33.000000	268.0	...	5.0
...	...	...	...	...	...	...	...	...	...	...	...	
1644	1.0	68.0	1.961070	0.688000	58.0	21.0	77.800000	12.200000	16.400000	185.0	...	7.4
1650	0.0	79.0	1.923645	0.509000	44.0	206.6	93.000000	46.300000	33.700000	422.3	...	4.3
1652	0.0	76.0	2.411833	0.750000	51.0	32.0	90.000000	25.333333	58.000000	233.0	...	8.0
1678	1.0	58.0	2.012633	0.736667	42.0	20.0	100.333333	21.000000	35.000000	242.5	...	11.8
1681	0.0	78.0	2.129067	0.483333	81.0	33.0	190.666667	21.000000	34.666667	303.0	...	10.0

463 rows × 34 columns

The 463 peoples are infected with Covid 19

Out of all the pts how many are infected ,what was the precentage of male and female.

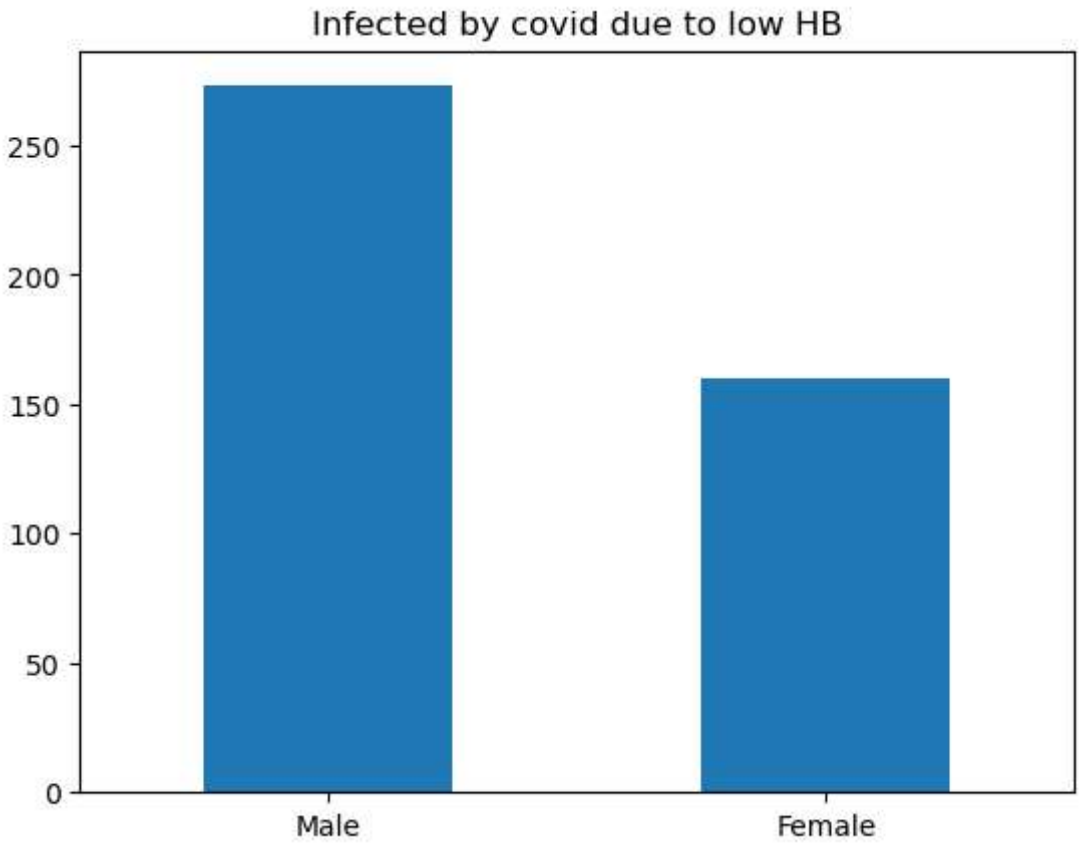
In [19]:
```
plt.figure(figsize=(6,6))
df[df["target"]==1]["Sex"].value_counts().plot.pie(autopct="%1.2f%%",labels=["Male","F
plt.title("How many PTs are infected")
plt.show()
```



The pie chart shows the count in percent of male and female infected through covid 19 in that Male count is 61.99% and Female count is 38.01%.

How many people may infected by covid on the basis of low HB

In [20]:
```
df[(df["HGB"]>10)&(df["target"]==1)]["Sex"].value_counts().plot.bar("HGB","Target")
plt.title("Infected by covid due to low HB")
plt.xticks(rotation=360,ticks=[0,1],labels=["Male","Female"])
plt.show()
```



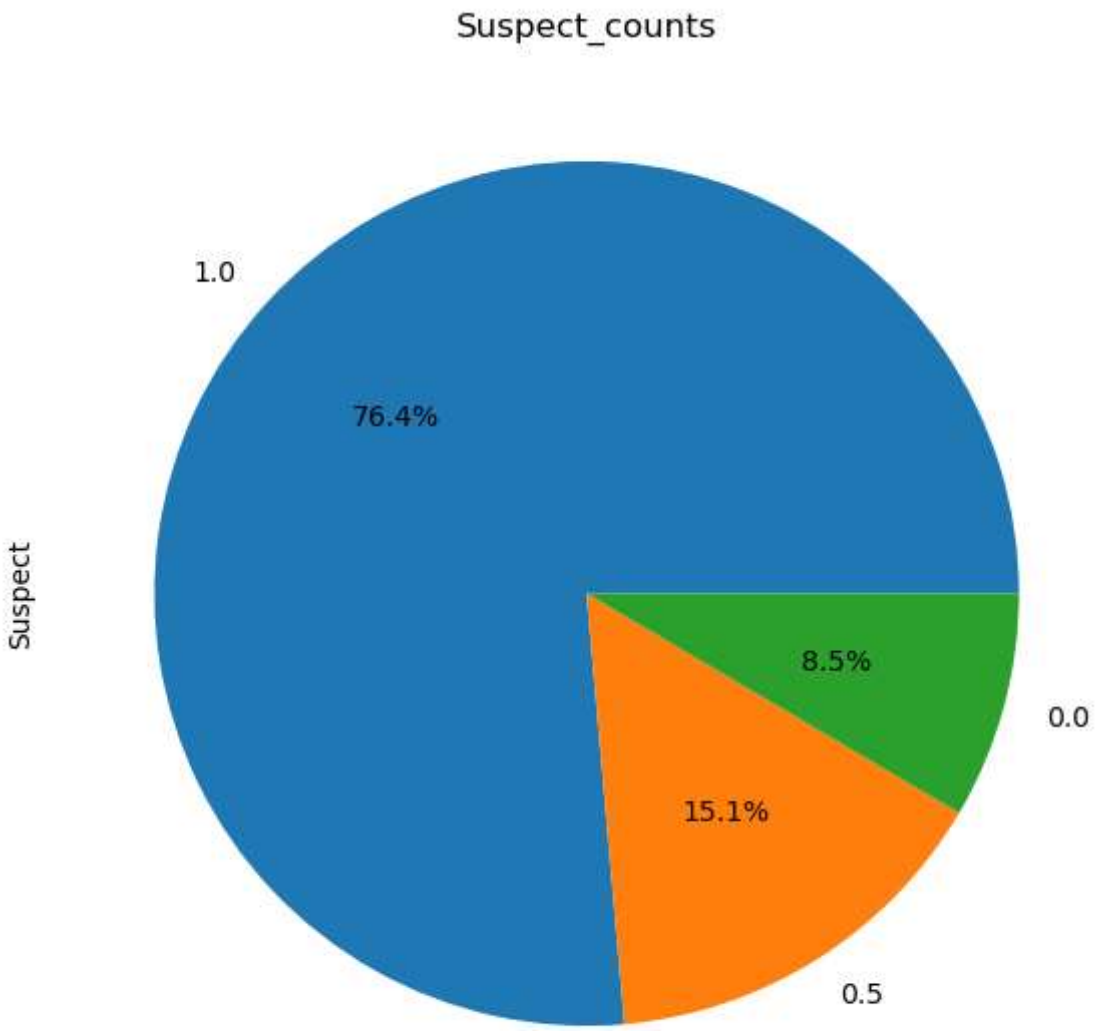
The Bar graph shows there is 300 Male and 150 Females are infected with COVID 19 and having low HB.

How many Suspects their

In [21]:
```
df["Suspect"].value_counts()
```

Out[21]:
```
1.0    631
0.5    125
0.0     70
Name: Suspect, dtype: int64
```

In [22]:
```
plt.figure(figsize=(7,7))
df["Suspect"].value_counts().plot.pie(autopct="%1.1f%%")
plt.title("Suspect_counts")
plt.show()
```



The piechart shows there is 76.4% peoples having severely ill pts, 15.1% peoples are in midstage and 8.5% of peoples having starting of symptoms.

To find Correlation between each other

In [23]:
```
df.corr()
```

Out[23]:

	Sex	Age	CA	CREA	ALP	GGT	GLU	AST	A
Sex	1.000000	0.032964	-0.176472	0.137660	-0.047572	0.058831	0.032916	0.078685	0.0935
Age	0.032964	1.000000	-0.238859	0.273081	0.120022	0.053419	0.249804	0.115363	-0.0411
CA	-0.176472	-0.238859	1.000000	-0.051125	-0.038610	-0.134819	-0.145684	-0.186710	-0.0790
CREA	0.137660	0.273081	-0.051125	1.000000	0.002392	-0.047184	0.144224	0.114154	-0.0219
ALP	-0.047572	0.120022	-0.038610	0.002392	1.000000	0.627228	0.105397	0.163809	0.3080
GGT	0.058831	0.053419	-0.134819	-0.047184	0.627228	1.000000	0.043929	0.245873	0.4051
GLU	0.032916	0.249804	-0.145684	0.144224	0.105397	0.043929	1.000000	0.282561	0.1515
AST	0.078685	0.115363	-0.186710	0.114154	0.163809	0.245873	0.282561	1.000000	0.7955
ALT	0.093531	-0.041111	-0.079054	-0.021936	0.308004	0.405130	0.151524	0.795529	1.0000
LDH	0.136243	0.217702	-0.363870	0.105350	0.067700	0.159610	0.218363	0.521652	0.3239
PCR	0.168879	0.271573	-0.370845	0.137722	0.149641	0.136683	0.201261	0.171029	0.0881
KAL	0.065228	0.137365	0.099978	0.349690	0.060161	-0.038302	0.198980	0.059899	0.0082
NAT	-0.068235	0.119231	0.186559	0.141179	-0.050146	-0.081768	-0.044650	-0.008688	-0.0631
UREA	0.090792	0.471753	-0.052998	0.779750	0.023025	-0.024742	0.264762	0.190825	0.0089
WBC	0.044195	0.211054	0.024329	0.126150	0.295461	0.076712	0.192364	0.090159	0.0825
RBC	0.171936	-0.364782	0.223227	-0.267463	-0.205150	-0.081024	-0.099958	-0.004908	0.0556
HGB	0.245951	-0.284492	0.216787	-0.242544	-0.212218	-0.039495	-0.108397	-0.011968	0.0735
HCT	0.216683	-0.243641	0.236125	-0.227804	-0.199339	-0.043371	-0.078642	-0.010606	0.0614
MCV	0.051476	0.300263	-0.029121	0.139621	0.052662	0.089164	0.053629	-0.013130	-0.0107
MCH	0.119644	0.150609	-0.025029	0.053350	-0.011921	0.069229	-0.019143	-0.021760	0.0126
MCHC	0.174265	-0.214973	0.003260	-0.129922	-0.123416	-0.006876	-0.141899	-0.026832	0.0469
PLT1	-0.089843	-0.120709	0.150566	-0.105594	0.147107	0.018547	0.103440	-0.007371	0.0653
NE	0.145248	0.365123	-0.302359	0.192153	0.102923	0.049562	0.243878	0.162003	0.0655
LY	-0.175347	-0.390806	0.276146	-0.204270	-0.119994	-0.045799	-0.250950	-0.151572	-0.0577
MO	0.017811	-0.066583	0.123827	-0.068322	-0.023203	-0.008706	-0.079999	-0.069567	-0.0421
EO	-0.046167	-0.186548	0.264232	-0.031158	0.023076	-0.060766	-0.104527	-0.116296	-0.0370
BA	-0.063545	-0.178190	0.333534	-0.101453	-0.001227	-0.059102	-0.145554	-0.130645	-0.0244
NET	0.073384	0.259953	-0.055236	0.159411	0.300469	0.091536	0.212451	0.122755	0.1042
LYT	-0.130488	-0.245906	0.326983	-0.142916	-0.005195	-0.052327	-0.107450	-0.109277	-0.0462
MOT	0.003576	0.106989	0.074064	0.016529	0.094119	0.009207	0.106548	0.025349	-0.0083
EOT	-0.065802	-0.144080	0.252001	-0.023793	0.044522	-0.059291	-0.066400	-0.108125	-0.0488
BAT	-0.080695	-0.034761	0.199333	0.008746	0.095796	-0.011792	-0.047000	-0.073203	-0.0051
Suspect	0.021589	-0.158580	-0.104685	-0.081629	-0.127160	0.015321	-0.014618	0.030381	0.0061
target	0.126874	0.044156	-0.379474	-0.047042	-0.099648	0.053349	0.091141	0.203083	0.1424

34 rows × 34 columns

Data is less correlate with each other

To check skewness of the data

In [24]:
```
df.describe()
```

Out[24]:

	Sex	Age	CA	CREA	ALP	GGT	GLU	A
count	826.000000	826.000000	826.000000	826.000000	826.000000	826.000000	826.000000	826.0000
mean	0.564165	62.133172	2.182430	1.227921	87.857244	68.300363	121.118503	49.4805
std	0.496166	17.485903	0.174258	1.154152	72.718560	122.832409	58.842110	58.6829
min	0.000000	17.000000	1.363333	0.420000	23.000000	4.000000	40.000000	12.0000
25%	0.000000	49.000000	2.070000	0.760000	56.000000	21.000000	92.000000	24.0000
50%	1.000000	62.000000	2.170000	0.950000	70.500000	35.000000	106.000000	34.0000
75%	1.000000	77.000000	2.290000	1.200000	93.000000	72.875000	127.000000	55.0000
max	1.000000	99.000000	3.280000	15.700000	780.500000	2272.000000	855.000000	1018.5000

8 rows × 34 columns

Splitting Data Into Features and Target

In [25]:
```
x=df.iloc[:,:-1]
x
```

Out[25]:

	Sex	Age	CA	CREA	ALP	GGT	GLU	AST	ALT	LDH	...	
0	1.0	82.0	2.090000	1.150000	95.0	40.0	78.000000	26.000000	21.000000	307.0	...	13.40(
3	0.0	82.0	2.270000	0.755000	123.5	176.5	106.000000	114.000000	63.000000	281.0	...	36.50(
4	1.0	79.0	2.070000	1.810000	62.0	36.5	96.000000	28.000000	38.500000	264.0	...	44.00(
5	1.0	84.0	2.060000	1.283333	75.0	75.0	95.500000	40.500000	27.000000	364.5	...	17.00(
8	0.0	48.0	2.110000	0.660000	200.0	90.0	104.000000	38.000000	36.000000	189.0	...	12.50(
...	...	...	...	...	...	...	...	...	...	...	...	
1644	1.0	68.0	1.961070	0.688000	58.0	21.0	77.800000	12.200000	16.400000	185.0	...	19.24(
1650	0.0	79.0	1.923645	0.509000	44.0	206.6	93.000000	46.300000	33.700000	422.3	...	7.11(
1652	0.0	76.0	2.411833	0.750000	51.0	32.0	90.000000	25.333333	58.000000	233.0	...	28.16(
1678	1.0	58.0	2.012633	0.736667	42.0	20.0	100.333333	21.000000	35.000000	242.5	...	31.93(
1681	0.0	78.0	2.129067	0.483333	81.0	33.0	190.666667	21.000000	34.666667	303.0	...	24.20(

826 rows × 33 columns

We are splitting overall data except target column that is "target" in x variable

In [26]:
```
y=df["target"]
y
```

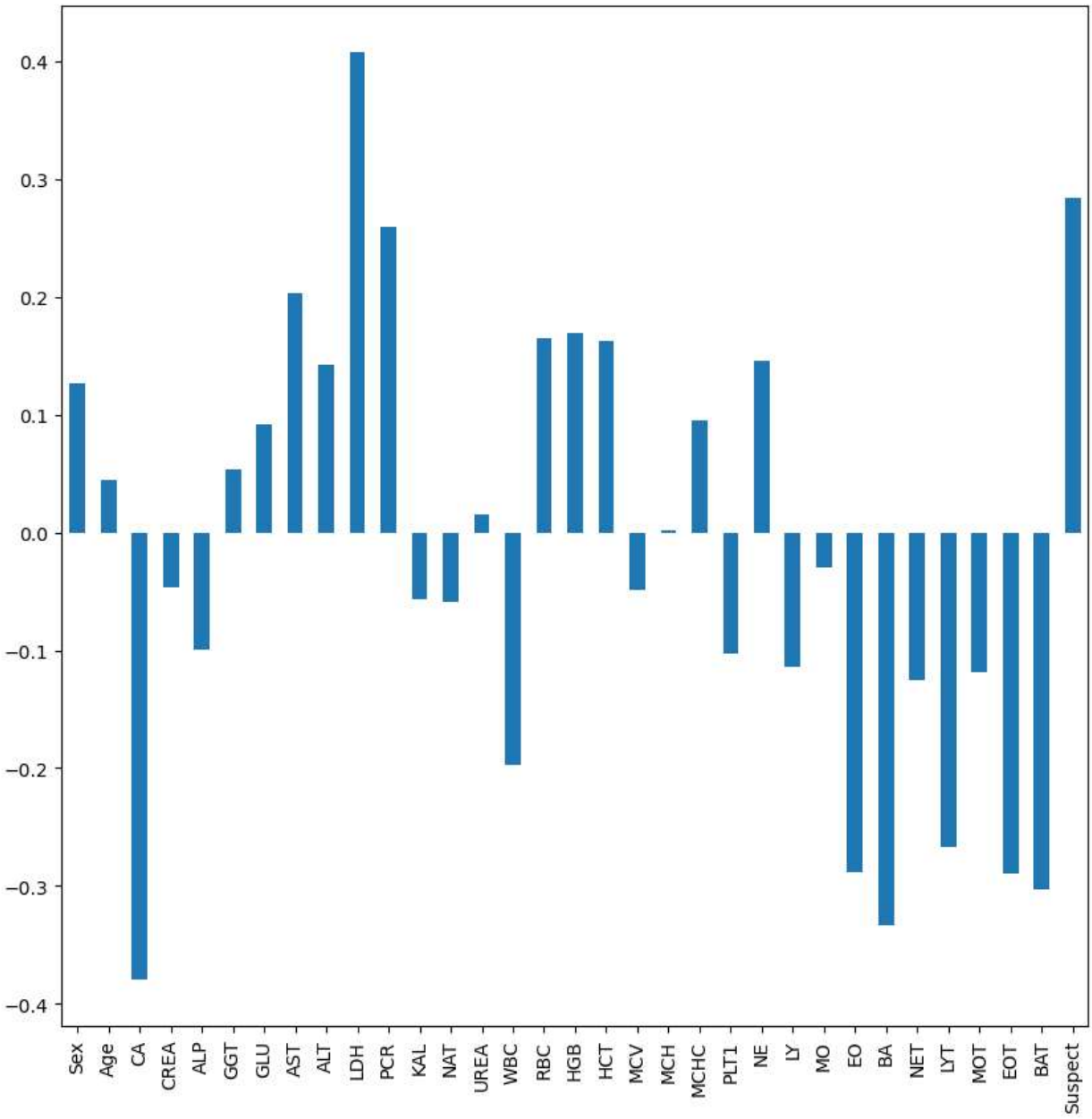
Out[26]:
```
0       0
3       0
4       0
5       1
8       0
       ..
1644    1
1650    1
1652    1
1678    1
1681    1
Name: target, Length: 826, dtype: int64
```

In that separate "target" column as a target in y variable.

Correlation of Feature and Target variable

In [27]:
```
x.corrwith(y).plot.bar(figsize=(10,10))
```

Out[27]:
```
<Axes: >
```



This bar shows the correlation between features(x) and target(Y)

Balance the Data

In [28]:
```
df["target"].value_counts()
```

Out[28]:
```
1    463
0    363
Name: target, dtype: int64
```

Our target column having inbalance data.i.e 1=463 and 0=363 We have to Balance the data.

Oversampling the data

In [29]:
```
import imblearn
```

For balancing data we use Oversampling technique.

In [30]:
```
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
```

We have to import Class RandomOverSampler and counter

In [31]:
```
ros=RandomOverSampler(random_state=1)
```

In [32]:
```
x_ros,y_ros=ros.fit_resample(x,y)
```

Make the object of class and pass the data in it.

In [33]:
```
print("Original Dataset",Counter(y))
print("Resampled Dataset",Counter(y_ros))
```

```
Original Dataset Counter({1: 463, 0: 363})
Resampled Dataset Counter({0: 463, 1: 463})
```

Here is the output , first we print original data and then we have to see our data is in balanced.

Apply Standard Scaler to Scale the data at one level

In [34]:
```
from sklearn.preprocessing import StandardScaler
```

By using standard scaler we have data in one level

In [35]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=1)
```

For standard scaler we need to split data into training and testing parts, in that testing data is 20% and random state is 1.

In [36]:
```
sc=StandardScaler()
xtrain=sc.fit_transform(xtrain)
xtest=sc.fit_transform(xtest)
```

We make object of standard scaler. In that we have to pass xtrain and xtest for transform into standard scaler.

In [37]:
```
xtrain
```

Out[37]:
```
array([[ 0.87177979,  1.1595822 , -0.51843211, ..., -0.2522526 ,
        -0.45159523, -1.07639066],
       [ 0.87177979,  1.1595822 ,  0.25074627, ..., -0.46243921,
        -0.45159523, -2.67283524],
       [-1.14707867,  1.27326673, -0.40447976, ..., -0.14715929,
        -0.45159523,  0.52005392],
       ...,
       [ 0.87177979,  0.42063276,  1.30480553, ...,  0.16812063,
        -0.45159523, -1.07639066],
       [-1.14707867,  0.87537088, -0.37599167, ...,  0.79868046,
        -0.45159523,  0.52005392],
       [ 0.87177979,  1.27326673, -0.46145594, ..., -0.46243921,
        -0.45159523, -1.07639066]])
```

This is our standardize xtrain data on which we performed Standard Scaling

In [38]:
```
xtest
```

Out[38]:
```
array([[-1.10151411, -0.72782876, -0.96412838, ..., -0.38556002,
        -0.43393576,  0.49712499],
       [ 0.9078413 ,  0.57897096, -1.02499824, ..., -0.38556002,
        -0.43393576,  0.49712499],
       [ 0.9078413 ,  0.57897096, -1.2076078 , ..., -0.38556002,
        -0.43393576,  0.49712499],
       ...,
       [-1.10151411, -1.08422868,  0.61848784, ...,  0.25730895,
         2.21922581,  0.49712499],
       [ 0.9078413 ,  0.69777094, -1.2076078 , ..., -0.38556002,
         0.89264502,  0.49712499],
       [-1.10151411,  1.46997077, -1.32934751, ..., -0.38556002,
        -0.43393576,  0.49712499]])
```

This is our standardize xtest data on which we performed Standard Scaling

Train_Test_Split for separating data into training and testing phase

In [39]:
```
from sklearn.model_selection import train_test_split
```

In [40]:
```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=1)
```

To build a module we have to saperate x and y for training and testing. In that test size is 20% and random state is 1.

Build a Model by using LogisticRegression Algoritham

In [41]:
```
from sklearn.linear_model import LogisticRegression
```

From class sklearn.linear_model build a model Logistic Regression

Classification Report of model trained on Logistic Regression

In [42]:
```
from sklearn.metrics import classification_report
```

In [43]:
```
lr=LogisticRegression()
lr.fit(xtrain,ytrain)
ypred_train=lr.predict(xtrain)
ypred_test=lr.predict(xtest)
```

In [44]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.76      0.78      0.77       294
           1       0.82      0.81      0.81       366

    accuracy                           0.79       660
   macro avg       0.79      0.79      0.79       660
weighted avg       0.79      0.79      0.79       660

Test Data
              precision    recall  f1-score   support

           0       0.80      0.80      0.80        69
           1       0.86      0.86      0.86        97

    accuracy                           0.83       166
   macro avg       0.83      0.83      0.83       166
weighted avg       0.83      0.83      0.83       166
```

Craete a object of this class and fit() method is used in order to train the model and use it for predictions.

ypred_train is variable that stored predicted values by model of training data.

ypred_test is variable that stored predicted values by model of testing data.

According to classification Report accuracy of training data is 79% and testing data is 83%. This is best fitted model.

By using Hyperparameter or Hypertunners

In [45]:
```
lr=LogisticRegression(solver="liblinear")
lr.fit(xtrain,ytrain)
ypred_train=lr.predict(xtrain)
ypred_test=lr.predict(xtest)
```

In [46]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.82      0.82      0.82       294
           1       0.86      0.86      0.86       366

    accuracy                           0.84       660
   macro avg       0.84      0.84      0.84       660
weighted avg       0.84      0.84      0.84       660

Test Data
              precision    recall  f1-score   support

           0       0.78      0.74      0.76        69
           1       0.82      0.86      0.84        97

    accuracy                           0.81       166
   macro avg       0.80      0.80      0.80       166
weighted avg       0.81      0.81      0.81       166
```

In [47]:
```
lr=LogisticRegression(solver="sag")
lr.fit(xtrain,ytrain)
ypred_train=lr.predict(xtrain)
ypred_test=lr.predict(xtest)
```

In [48]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.72      0.76      0.74       294
           1       0.80      0.77      0.78       366

    accuracy                           0.76       660
   macro avg       0.76      0.76      0.76       660
weighted avg       0.76      0.76      0.76       660

Test Data
              precision    recall  f1-score   support

           0       0.77      0.74      0.76        69
           1       0.82      0.85      0.83        97

    accuracy                           0.80       166
   macro avg       0.80      0.79      0.79       166
weighted avg       0.80      0.80      0.80       166
```

In [49]:
```
lr=LogisticRegression(solver="saga")
lr.fit(xtrain,ytrain)
ypred_train=lr.predict(xtrain)
ypred_test=lr.predict(xtest)
```

In [50]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
```

```
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.71      0.73      0.72       294
           1       0.78      0.75      0.77       366

    accuracy                           0.75       660
   macro avg       0.74      0.74      0.74       660
weighted avg       0.75      0.75      0.75       660

Test Data
              precision    recall  f1-score   support

           0       0.76      0.74      0.75        69
           1       0.82      0.84      0.83        97

    accuracy                           0.80       166
   macro avg       0.79      0.79      0.79       166
weighted avg       0.79      0.80      0.79       166
```

By using Hypertunning Parameters i.e "liblenear", "sag","saga" we have best fit model.

Build a model by using KNN Algoritham

In [51]:
```
from sklearn.neighbors import KNeighborsClassifier
```

From class sklearn.neighbors build a model KNeighborsClassifier

In [52]:
```
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(xtrain,ytrain)
ypred_train=knn.predict(xtrain)
ypred_test=knn.predict(xtest)
```

Craete a object of this class and fit() method is used in order to train the model and use it for predictions.

ypred_train is variable that stored predicted values by model of training data.

ypred_test is variable that stored predicted values by model of testing data.

Evaluate the model by using Classification_report

In [53]:
```
from sklearn.metrics import classification_report
```

In [54]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.81      0.76      0.79       294
           1       0.82      0.86      0.84       366

    accuracy                           0.82       660
   macro avg       0.82      0.81      0.81       660
weighted avg       0.82      0.82      0.82       660

Test Data
              precision    recall  f1-score   support

           0       0.80      0.57      0.66        69
           1       0.74      0.90      0.81        97

    accuracy                           0.76       166
   macro avg       0.77      0.73      0.74       166
weighted avg       0.77      0.76      0.75       166
```

According to classification Report accuracy of training data is 82% and testing data is 76%. This is best fitted model.

Build a model on SVM Algoritham

In [55]:
```
from sklearn.svm import SVC
```

From class sklearn.svm build a model svc(suport vector machine)

In [56]:
```
svm=SVC()
```

Create A object of class svm

In [57]:
```
def mymodel(model):
    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)
    print(classification_report(ytest,ypred))
    return model
```

We are making one function i.e mymodel in that we have to pass our model name.

Fit() method is used in order to train the model and use it for predictions.

Predicted values are stored in a ypred variable.

we have to print classification report of this model for predict accuracy of trainning data.

In return we have to pass name of our module

In [58]:
```
from sklearn.pipeline import Pipeline
```

From class sklearn.pipline we have to import Pipeline

We using pipeline beacause data should be process smoothly.

In [59]:
```
pipe=Pipeline(steps=[('scaler',StandardScaler()),('svm',SVC())])
```

In pipe we write steps so that steps will be executed one by one

In [60]:
```
pipe.fit(xtrain,ytrain)
ypred=pipe.predict(xtest)
```

In [61]:
```
mymodel(svm)
```

```
              precision    recall  f1-score   support

           0       0.77      0.72      0.75        69
           1       0.81      0.85      0.83        97

    accuracy                           0.80       166
   macro avg       0.79      0.78      0.79       166
weighted avg       0.79      0.80      0.79       166
```

Out[61]:
```
SVC()
```

fit() method is used in order to train the model and use it for predictions.

ypred is variable that stored predicted values by model of testing data

We have to call the function and pass model name

By using Hyperparameter or Hypertunners

In [62]:
```
svm=SVC(kernel='linear')
mymodel(svm)
```

```
              precision    recall  f1-score   support

           0       0.77      0.78      0.78        69
           1       0.84      0.84      0.84        97

    accuracy                           0.81       166
   macro avg       0.81      0.81      0.81       166
weighted avg       0.81      0.81      0.81       166
```

Out[62]:
```
SVC(kernel='linear')
```

By using linear kernel i.e hypertunning parameter we are having accuracy 81%.

In [63]:
```
svm=SVC(kernel='sigmoid')
mymodel(svm)
```

```
                precision    recall  f1-score   support

           0       0.45      0.54      0.49        69
           1       0.61      0.53      0.57        97

    accuracy                           0.53       166
   macro avg       0.53      0.53      0.53       166
weighted avg       0.54      0.53      0.53       166
```

Out[63]:
```
SVC(kernel='sigmoid')
```

By using sigmoid kernel i.e hypertunning parameter we are having accuracy 53%. Its very low accuracy.

In [64]:
```
svm=SVC(kernel='poly')
mymodel(svm)
```

```
              precision    recall  f1-score   support

           0       0.67      0.81      0.74        69
           1       0.84      0.72      0.78        97

    accuracy                           0.76       166
   macro avg       0.76      0.77      0.76       166
weighted avg       0.77      0.76      0.76       166
```

Out[64]:
```
SVC(kernel='poly')
```

By using poly kernel i.e hypertunning parameter we are having accuracy 76%.

Build a model by using Decision Tree Classifier

In [65]:
```
from sklearn.tree import DecisionTreeClassifier
```

From class sklearn.tree build a model Decision Tree Classifier

In [66]:
```
dt=DecisionTreeClassifier()
dt.fit(xtrain,ytrain)
ytrain_pred=dt.predict(xtrain)
ytest_pred=dt.predict(xtest)
```

Craete a object of this class and fit() method is used in order to train the model and use it for predictions.

ypred_train is variable that stored predicted values by model of training data.

ypred_test is variable that stored predicted values by model of testing data.

Evaluate the model by using Classification_report

In [67]:
```
from sklearn.metrics import classification_report,accuracy_score
```

In [68]:
```
print("Train Data")
print(classification_report(ytrain,ytrain_pred))
print("Test Data")
print(classification_report(ytest,ytest_pred))
```

```
Train Data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       294
           1       1.00      1.00      1.00       366

    accuracy                           1.00       660
   macro avg       1.00      1.00      1.00       660
weighted avg       1.00      1.00      1.00       660

Test Data
              precision    recall  f1-score   support

           0       0.70      0.74      0.72        69
           1       0.81      0.77      0.79        97

    accuracy                           0.76       166
   macro avg       0.75      0.76      0.75       166
weighted avg       0.76      0.76      0.76       166
```

According to classification Report accuracy of training data should be 100% its high bias and testing data is 76% its variance is high. Beacause of this scenario it is underfitted model.

Performing Hypertunning on model

Hyper parameters with impurity checking gini

In [69]:
```
for i in range(1,31):
    dt1=DecisionTreeClassifier(max_depth=i)
    dt1.fit(xtrain,ytrain)
    ypred=dt1.predict(xtest)
    ac=accuracy_score(ytest,ypred)
    print(f"Max Depth:{i} accuracy): {ac}")
```

```
Max Depth:1 accuracy): 0.7289156626506024
Max Depth:2 accuracy): 0.7650602409638554
Max Depth:3 accuracy): 0.7951807228915663
Max Depth:4 accuracy): 0.7590361445783133
Max Depth:5 accuracy): 0.7771084337349398
Max Depth:6 accuracy): 0.7590361445783133
Max Depth:7 accuracy): 0.7530120481927711
Max Depth:8 accuracy): 0.7650602409638554
Max Depth:9 accuracy): 0.7771084337349398
Max Depth:10 accuracy): 0.7650602409638554
Max Depth:11 accuracy): 0.7590361445783133
Max Depth:12 accuracy): 0.7590361445783133
Max Depth:13 accuracy): 0.7590361445783133
Max Depth:14 accuracy): 0.7228915662650602
Max Depth:15 accuracy): 0.7891566265060241
Max Depth:16 accuracy): 0.7530120481927711
Max Depth:17 accuracy): 0.7409638554216867
Max Depth:18 accuracy): 0.7469879518072289
Max Depth:19 accuracy): 0.7891566265060241
Max Depth:20 accuracy): 0.7590361445783133
Max Depth:21 accuracy): 0.7530120481927711
Max Depth:22 accuracy): 0.7590361445783133
Max Depth:23 accuracy): 0.7469879518072289
Max Depth:24 accuracy): 0.7710843373493976
Max Depth:25 accuracy): 0.7349397590361446
Max Depth:26 accuracy): 0.7349397590361446
Max Depth:27 accuracy): 0.7409638554216867
Max Depth:28 accuracy): 0.7590361445783133
Max Depth:29 accuracy): 0.7650602409638554
Max Depth:30 accuracy): 0.7650602409638554
```

We splin a loop in range 1 to 31, Dt model apply on this range and this values are stored into dt1 variable.

Fit() method is used in order to train the model and use it for predictions.

Ypred is a variable its stored predicted values of xtest

Accuracy score of this values are stored in ac variable.

print maximum depth and accuracy score of each .

Building model with final value of max depth as 3

In [70]:
```
dt1=DecisionTreeClassifier(max_depth=3)
mymodel(dt1)
```

```
              precision    recall  f1-score   support

           0       0.73      0.81      0.77        69
           1       0.85      0.78      0.82        97

    accuracy                           0.80       166
   macro avg       0.79      0.80      0.79       166
weighted avg       0.80      0.80      0.80       166
```

Out[70]:
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3)
```

In these accuracy we want to select the max depth in wich accuracy score is decreases. In these scenario max depth 3 is the point when accuracy score was decrease.

With the help of gini index Hypertunning parameter accuracy is be 80%.

Lets checking overfitting scenario

In [71]:
```
dt1.score(xtrain,ytrain)
```

Out[71]:
```
0.7909090909090909
```

With the help of gini index Hypertunning parameter model is best fitted.

Hyper parameters with impurity checking min_sample_split

In [72]:
```
for i in range(5,50):
    dt3=DecisionTreeClassifier(min_samples_split=i)
    dt3.fit(xtrain,ytrain)
    ypred=dt3.predict(xtest)
    ac=accuracy_score(ytest,ypred)
    print(f"min sample split:{i} accuracy): {ac}")
```

```
min sample split:5 accuracy): 0.7710843373493976
min sample split:6 accuracy): 0.7590361445783133
min sample split:7 accuracy): 0.7590361445783133
min sample split:8 accuracy): 0.7710843373493976
min sample split:9 accuracy): 0.7650602409638554
min sample split:10 accuracy): 0.7590361445783133
min sample split:11 accuracy): 0.7590361445783133
min sample split:12 accuracy): 0.7590361445783133
min sample split:13 accuracy): 0.7469879518072289
min sample split:14 accuracy): 0.7409638554216867
min sample split:15 accuracy): 0.7710843373493976
min sample split:16 accuracy): 0.7650602409638554
min sample split:17 accuracy): 0.7650602409638554
min sample split:18 accuracy): 0.7710843373493976
min sample split:19 accuracy): 0.7771084337349398
min sample split:20 accuracy): 0.7710843373493976
min sample split:21 accuracy): 0.7710843373493976
min sample split:22 accuracy): 0.7650602409638554
min sample split:23 accuracy): 0.7650602409638554
min sample split:24 accuracy): 0.7650602409638554
min sample split:25 accuracy): 0.7650602409638554
min sample split:26 accuracy): 0.7650602409638554
min sample split:27 accuracy): 0.7590361445783133
min sample split:28 accuracy): 0.7650602409638554
min sample split:29 accuracy): 0.7710843373493976
min sample split:30 accuracy): 0.7771084337349398
min sample split:31 accuracy): 0.7710843373493976
min sample split:32 accuracy): 0.7771084337349398
min sample split:33 accuracy): 0.7710843373493976
min sample split:34 accuracy): 0.7771084337349398
min sample split:35 accuracy): 0.7771084337349398
min sample split:36 accuracy): 0.7771084337349398
min sample split:37 accuracy): 0.7710843373493976
min sample split:38 accuracy): 0.7710843373493976
min sample split:39 accuracy): 0.7831325301204819
min sample split:40 accuracy): 0.7831325301204819
min sample split:41 accuracy): 0.7831325301204819
min sample split:42 accuracy): 0.7831325301204819
min sample split:43 accuracy): 0.7891566265060241
min sample split:44 accuracy): 0.7831325301204819
min sample split:45 accuracy): 0.7891566265060241
min sample split:46 accuracy): 0.7891566265060241
min sample split:47 accuracy): 0.7891566265060241
min sample split:48 accuracy): 0.7891566265060241
min sample split:49 accuracy): 0.7891566265060241
```

We splin a loop in range 5 to 50, Dt model apply on this range and this values are stored into dt3 variable.

Fit() method is used in order to train the model and use it for predictions.

Ypred is a variable its stored predicted values of xtest

Accuracy score of this values are stored in ac variable.

print values if min_samples_split and accuracy score of each .

Building model with final value of max depth as 8

In [73]:
```
dt3=DecisionTreeClassifier(min_samples_split=8)
mymodel(dt3)
```

```
              precision    recall  f1-score   support

           0       0.71      0.77      0.74        69
           1       0.82      0.77      0.80        97

    accuracy                           0.77       166
   macro avg       0.77      0.77      0.77       166
weighted avg       0.78      0.77      0.77       166
```

Out[73]:

DecisionTreeClassifier

DecisionTreeClassifier(min_samples_split=8)

In these accuracy we want to select the values ofmin_samples_split in wich accuracy score is decreases. In these scenario min_samples_split=8 is the point when accuracy score was decrease.

With the help of min_samples_split Hypertunning parameter accuracy is be 78%.

Lets checking overfitting scenario

In [74]:
```
dt3.score(xtrain,ytrain)
```

Out[74]:
```
0.9621212121212122
```

With the help of min_samples_split Hypertunning parameter model is over fitted.

Hyper parameters with impurity checking min_sample_leaf

In [75]:
```
for i in range(1,51):
    dt4=DecisionTreeClassifier(min_samples_leaf=i)
    dt4.fit(xtrain,ytrain)
    ypred=dt4.predict(xtest)
    ac=accuracy_score(ytest,ypred)
    print(f"min sample leaf:{i} accuracy): {ac}")
```

```
min sample leaf:1 accuracy): 0.7710843373493976
min sample leaf:2 accuracy): 0.7771084337349398
min sample leaf:3 accuracy): 0.7771084337349398
min sample leaf:4 accuracy): 0.7650602409638554
min sample leaf:5 accuracy): 0.7048192771084337
min sample leaf:6 accuracy): 0.7349397590361446
min sample leaf:7 accuracy): 0.7228915662650602
min sample leaf:8 accuracy): 0.7228915662650602
min sample leaf:9 accuracy): 0.7710843373493976
min sample leaf:10 accuracy): 0.7469879518072289
min sample leaf:11 accuracy): 0.7650602409638554
min sample leaf:12 accuracy): 0.7590361445783133
min sample leaf:13 accuracy): 0.7710843373493976
min sample leaf:14 accuracy): 0.7710843373493976
min sample leaf:15 accuracy): 0.7831325301204819
min sample leaf:16 accuracy): 0.7831325301204819
min sample leaf:17 accuracy): 0.7409638554216867
min sample leaf:18 accuracy): 0.7228915662650602
min sample leaf:19 accuracy): 0.7710843373493976
min sample leaf:20 accuracy): 0.8012048192771084
min sample leaf:21 accuracy): 0.7951807228915663
min sample leaf:22 accuracy): 0.7951807228915663
min sample leaf:23 accuracy): 0.7951807228915663
min sample leaf:24 accuracy): 0.7951807228915663
min sample leaf:25 accuracy): 0.7951807228915663
min sample leaf:26 accuracy): 0.7951807228915663
min sample leaf:27 accuracy): 0.7951807228915663
min sample leaf:28 accuracy): 0.7951807228915663
min sample leaf:29 accuracy): 0.7951807228915663
min sample leaf:30 accuracy): 0.7951807228915663
min sample leaf:31 accuracy): 0.8072289156626506
min sample leaf:32 accuracy): 0.8072289156626506
min sample leaf:33 accuracy): 0.7951807228915663
min sample leaf:34 accuracy): 0.7951807228915663
min sample leaf:35 accuracy): 0.7951807228915663
min sample leaf:36 accuracy): 0.7951807228915663
min sample leaf:37 accuracy): 0.7951807228915663
min sample leaf:38 accuracy): 0.7951807228915663
min sample leaf:39 accuracy): 0.7951807228915663
min sample leaf:40 accuracy): 0.7951807228915663
min sample leaf:41 accuracy): 0.7951807228915663
min sample leaf:42 accuracy): 0.7951807228915663
min sample leaf:43 accuracy): 0.7951807228915663
min sample leaf:44 accuracy): 0.7951807228915663
min sample leaf:45 accuracy): 0.7951807228915663
min sample leaf:46 accuracy): 0.7951807228915663
min sample leaf:47 accuracy): 0.7951807228915663
min sample leaf:48 accuracy): 0.7951807228915663
min sample leaf:49 accuracy): 0.7951807228915663
min sample leaf:50 accuracy): 0.7951807228915663
```

We splin a loop in range 1 to 51, Dt model apply on this range and this values are stored into dt4 variable.

Fit() method is used in order to train the model and use it for predictions.

Ypred is a variable its stored predicted values of xtest

Accuracy score of this values are stored in ac variable.

print values if min_samples_leaf and accuracy score of each .

Building model with final value of max depth as 2

In [76]:
```
dt4=DecisionTreeClassifier(min_samples_leaf=2)
mymodel(dt4)
```

```
              precision    recall  f1-score   support

           0       0.68      0.81      0.74        69
           1       0.85      0.73      0.78        97

    accuracy                           0.77       166
   macro avg       0.76      0.77      0.76       166
weighted avg       0.78      0.77      0.77       166
```

Out[76]:
```
DecisionTreeClassifier(min_samples_leaf=2)
```

In these accuracy we want to select the values of min_samples_leaf in wich accuracy score is decreases. In these scenario min_samples_leaf=8 is the point when accuracy score was decrease.

With the help of min_samples_leaf Hypertunning parameter accuracy is be 78%.

Lets checking overfitting scenario

In [77]:
```
dt4.score(xtrain,ytrain)
```

Out[77]:
```
0.9575757575757575
```

With the help of min_samples_leaf Hypertunning parameter model is over fitted.

Bagging

In [78]:
```
from sklearn.ensemble import BaggingClassifier
```

From class sklearn.ensemble build a model BaggingClassifier

In [79]:
```
bg=BaggingClassifier(LogisticRegression())
bg.fit(xtrain,ytrain)
ypred_train=bg.predict(xtrain)
ypred_test=bg.predict(xtest)
```

Craete a object of this class and pass a module (Logistic regression) in that and fit() method is used in order to train the model and use it for predictions.

ypred_train is variable that stored predicted values by model of training data.

ypred_test is variable that stored predicted values by model of testing data.

Classification Report

In [80]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.77      0.76      0.77       294
           1       0.81      0.82      0.81       366

    accuracy                           0.79       660
   macro avg       0.79      0.79      0.79       660
weighted avg       0.79      0.79      0.79       660

Test Data
              precision    recall  f1-score   support

           0       0.79      0.80      0.79        69
           1       0.85      0.85      0.85        97

    accuracy                           0.83       166
   macro avg       0.82      0.82      0.82       166
weighted avg       0.83      0.83      0.83       166
```

According to classification Report accuracy through Logistic Regression of training data is 80% and testing data is 83%. This is best fitted model.

In [81]:
```
bg=BaggingClassifier(DecisionTreeClassifier())
bg.fit(xtrain,ytrain)
ypred_train=bg.predict(xtrain)
ypred_test=bg.predict(xtest)
```

Craete a object of this class and pass a module (DecisionTreeClassifier) in that and fit() method is used in order to train the model and use it for predictions.

ypred_train is variable that stored predicted values by model of training data.

ypred_test is variable that stored predicted values by model of testing data.

Classification Report

In [82]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       294
           1       1.00      0.97      0.98       366

    accuracy                           0.98       660
   macro avg       0.98      0.98      0.98       660
weighted avg       0.98      0.98      0.98       660

Test Data
              precision    recall  f1-score   support

           0       0.76      0.83      0.79        69
           1       0.87      0.81      0.84        97

    accuracy                           0.82       166
   macro avg       0.81      0.82      0.82       166
weighted avg       0.82      0.82      0.82       166
```

According to classification Report accuracy through DecisionTreeClassifier of training data is 99% and testing data is 82%. This is over fitted model.

In [83]:
```
bg=BaggingClassifier(KNeighborsClassifier())
bg.fit(xtrain,ytrain)
ypred_train=bg.predict(xtrain)
ypred_test=bg.predict(xtest)
```

Craete a object of this class and pass a module (KNeighborsClassifier) in that and fit() method is used in order to train the model and use it for predictions.

ypred_train is variable that stored predicted values by model of training data.

ypred_test is variable that stored predicted values by model of testing data.

Classification Report

In [84]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.80      0.76      0.78       294
           1       0.82      0.85      0.83       366

    accuracy                           0.81       660
   macro avg       0.81      0.80      0.81       660
weighted avg       0.81      0.81      0.81       660

Test Data
              precision    recall  f1-score   support

           0       0.80      0.58      0.67        69
           1       0.75      0.90      0.82        97

    accuracy                           0.77       166
   macro avg       0.78      0.74      0.74       166
weighted avg       0.77      0.77      0.76       166
```

According to classification Report accuracy through KNeighborsClassifier of training data is 82% and testing data is 78%. This is best fitted model.

RandomForest

In [85]:
```
from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier()
rf.fit(xtrain,ytrain)
ypred_train=rf.predict(xtrain)
ypred_test=rf.predict(xtest)
```

From class sklearn.ensemble build a model RandomForestClassifier

Craete a object of this class and fit() method is used in order to train the model and use it for predictions.

ypred_train is variable that stored predicted values by model of training data.

ypred_test is variable that stored predicted values by model of testing data.

Classification Report

In [86]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       294
           1       1.00      1.00      1.00       366

    accuracy                           1.00       660
   macro avg       1.00      1.00      1.00       660
weighted avg       1.00      1.00      1.00       660

Test Data
              precision    recall  f1-score   support

           0       0.85      0.84      0.85        69
           1       0.89      0.90      0.89        97

    accuracy                           0.87       166
   macro avg       0.87      0.87      0.87       166
weighted avg       0.87      0.87      0.87       166
```

According to classification Report accuracy of training data should be 100% its high bias and testing data is 86% its variance is high. Beacause of this scenario it is underfitted model.

Voting

In [87]:
```
from sklearn.ensemble import VotingClassifier
```

From class sklearn.ensemble build a model VotingClassifier

In [88]:
```
model=[]
model.append(("Logistic Regression",LogisticRegression()))
model.append(("Decision Tree",DecisionTreeClassifier()))
```

In [89]:
```
model
```

Out[89]:
```
[('Logistic Regression', LogisticRegression()),
 ('Decision Tree', DecisionTreeClassifier())]
```

Make a two lists i.e. Model, Accuracy. Logistic and Decision Tree algo uppend in the model list.

In [90]:
```
vc=VotingClassifier(estimators=model)
vc.fit(xtrain,ytrain)
ypred_train=vc.predict(xtrain)
ypred_test=vc.predict(xtest)
```

Craete a object of this class and pass estimators is model and fit() method is used in order to train the model and use it for predictions.

ypred_train is variable that stored predicted values by model of training data.

ypred_test is variable that stored predicted values by model of testing data.

Classification Report

In [91]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.81      1.00      0.89       294
           1       1.00      0.81      0.89       366

    accuracy                           0.89       660
   macro avg       0.90      0.90      0.89       660
weighted avg       0.91      0.89      0.89       660

Test Data
              precision    recall  f1-score   support

           0       0.72      0.88      0.79        69
           1       0.90      0.75      0.82        97

    accuracy                           0.81       166
   macro avg       0.81      0.82      0.81       166
weighted avg       0.82      0.81      0.81       166
```

According to classification Report accuracy of training data is 89% and testing data is 81% . Beacause of this scenario it is best fitted model.

Boosting

AdaBoost

In [92]:
```
from sklearn.ensemble import AdaBoostClassifier
ad=AdaBoostClassifier()
ad.fit(xtrain,ytrain)
ypred_train=ad.predict(xtrain)
ypred_test=ad.predict(xtest)
```

From class sklearn.ensemble build a model AdaBoostClassifier

Craete a object of this class and fit() method is used in order to train the model and use it for predictions.

ypred_train is variable that stored predicted values by model of training data.

ypred_test is variable that stored predicted values by model of testing data.

Classification Report

In [93]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.87      0.87      0.87       294
           1       0.89      0.90      0.89       366

    accuracy                           0.88       660
   macro avg       0.88      0.88      0.88       660
weighted avg       0.88      0.88      0.88       660

Test Data
              precision    recall  f1-score   support

           0       0.77      0.77      0.77        69
           1       0.84      0.84      0.84        97

    accuracy                           0.81       166
   macro avg       0.80      0.80      0.80       166
weighted avg       0.81      0.81      0.81       166
```

According to classification Report accuracy of training data is 88% and testing data is 81% . Beacause of this scenario it is best fitted model.

GBoost

In [94]:
```
from sklearn.ensemble import GradientBoostingClassifier
gd=GradientBoostingClassifier()
gd.fit(xtrain,ytrain)
ypred_train=gd.predict(xtrain)
ypred_test=gd.predict(xtest)
```

From class sklearn.ensemble build a model GradientBoostingClassifier

Craete a object of this class and fit() method is used in order to train the model and use it for predictions.

ypred_train is variable that stored predicted values by model of training data.

ypred_test is variable that stored predicted values by model of testing data.

Classification Report

In [95]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       294
           1       0.97      0.99      0.98       366

    accuracy                           0.98       660
   macro avg       0.98      0.97      0.98       660
weighted avg       0.98      0.98      0.98       660

Test Data
              precision    recall  f1-score   support

           0       0.80      0.75      0.78        69
           1       0.83      0.87      0.85        97

    accuracy                           0.82       166
   macro avg       0.82      0.81      0.81       166
weighted avg       0.82      0.82      0.82       166
```

According to classification Report accuracy of training data is 98% and testing data is 82% . Beacause of this scenario it is over fitted model.

XGBoost

In [96]:
```
from xgboost import XGBClassifier
xg=XGBClassifier()
xg.fit(xtrain,ytrain)
ypred_train=xg.predict(xtrain)
ypred_test=xg.predict(xtest)
```

From class sklearn.ensemble build a model XGBClassifier

Craete a object of this class and fit() method is used in order to train the model and use it for predictions.

ypred_train is variable that stored predicted values by model of training data.

ypred_test is variable that stored predicted values by model of testing data.

Classification Report

In [97]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       294
           1       1.00      1.00      1.00       366

    accuracy                           1.00       660
   macro avg       1.00      1.00      1.00       660
weighted avg       1.00      1.00      1.00       660

Test Data
              precision    recall  f1-score   support

           0       0.78      0.74      0.76        69
           1       0.82      0.86      0.84        97

    accuracy                           0.81       166
   macro avg       0.80      0.80      0.80       166
weighted avg       0.81      0.81      0.81       166
```

According to classification Report accuracy of training data is 100% that is high bias and testing data is 81% with high varieance . Beacause of this scenario it is over fitted model.

The Best Prediction was done by model trained on AdaBoost Algoritham with accuracy of 88% on Train Data and 81% on Test Data

In [ ]: