# LABORATORY RECORD

**ON**

## ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING ( 20CS51I )

Submitted to the **State Board of Technical Examinations, Government of Karnataka** in Partial Fulfillment of the Requirements for the Award of

**Diploma in**
**"Computer Science a n d Engineering"**

**By**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Name** | | | | | | | | | | |
| **Register No.** | **1** | **6** | **3** | **C** | **S** | **2** | | | | |

**Course Coordinator: Mr. Gagandeep**

**DEPARTMENT OF COLLEGIATE & TECHNICAL EDUCATION**

**GOVERNMENT POLYTECHNIC, BANTWAL-574211**
**DAKSHINA KANNADA**

**November -2023**

# EXP NO:1

**Create two series as shown using pd. series( )function.**

Series — A = [10, 20, 30, 40, 50]

Series — B = [40, 50, 60, 70, 80]

***(i) Get the items not common to both***

In [1]:
```
import pandas as pd
import numpy as np
series_A=pd.Series([10,20,30,40,50])
series_B=pd.Series([40,50,60,70,80])
union=pd.Series(np.union1d(series_A,series_B))
intersect=pd.Series(np.intersect1d(series_A,series_B))
notcommonseries=union[-union.isin(intersect)]
print(notcommonseries)
```

```
0    10
1    20
2    30
5    60
6    70
7    80
dtype: int64
```

***(ii)Identify the smallest and largest element in the Series A.***

In [3]:
```
print('Largest number in series_A is:',series_A.max())
print('Smallest number in series_A is',series_A.min())
```

```
Largest number in series_A is: 50
Smallest number in series A is 10
```

***(iii)Find the sum of Series B.***

In [5]:
```
print('The sum of Series_B is:',series_B.sum())
```

```
The sum of Series_B is: 300
```

***(iv)Calculate average in the Series A***

In [6]:
```
print('The average of series_A is:',series_A.mean())
```

```
The average of series_A is: 30.0
```

***(v)Find median in the given Series B***

In [7]:
```
print('The median of series_B is:',series_B.median())
```

```
The median of series_B is: 60.0
```

# EXP NO:2

## 2. Create a data frame with the following data.

In [8]:
```
import pandas as pd
import numpy as np
data=(
    ' FirstName' :['Aryan', 'Rohan','Riya','Yash','Siddini'],
    'lastname' :['Singh','Agarwal', 'Shah','Bhatia','Khanna'],
    'Type':['Full-time employee', 'intem','Full-time employee', 'Part-time emp
    'Department' :['Administration','Technical', 'Administration','Technical',
    'Yoe':[2,3,5,7,6],
    'Salary' :[20000,5000,10000,10000,20000]

df=pd.DataFrame(data)
print(df)
```

```
  FirstName lastname                Type      Department  Yoe  Salary
0     Aryan    Singh  Full-time employee  Administration    2   20000
1     Rohan  Agarwal               intem      Technical    3    5000
2      Riya     Shah  Full-time employee  Administration    5   10000
3      Yash   Bhatia  Part-time employee      Technical    7   10000
4   Siddini   Khanna  Full-time employee     Management    6   20000
```

**a)** ***Make a pivot table that shows the average salary of each type of employee for eachdepartment***.

In [9]:
```
table=pd.pivot_table(df,index=['Type','Department'],values='Salary',aggfunc='m‹
table
```

Out[9]:

| | | Salary |
|---|---|---|
| **Type** | **Department** | |
| **Full-time employee** | **Administration** | 15000 |
| | **Management** | 20000 |
| **Part-time employee** | **Technical** | 10000 |
| **intern** | **Technical** | 5000 |

**b)** ***Make a pivot table that shows the sum and mean of the salaries of each type of employeeand the number of employees of each type***.

In [11]:
```
pivot=pd.pivot_table(data=df,index='Type',values='Salary',aggfunc=['mean',  'sum
print(pivot)
```

```
                            mean    sum
                          Salary Salary
Type
Full-time employee  16666.666667  50000
Part-time employee  10000.000000  10000
intem                5000.000000   5000
```

In [12]:
```
df.pivot_table(index='Type',values='Salary',aggfunc='count')
```

Out[12] :

| | | Salary |
|---|---|---|
| Type | | |
| Full-time | employee | 3 |
| Part-time | employee | 1 |
| | intem | 1 |

***c) Make a pivot table that shows standard deviation for salary column.***

In [13]:
```
df.pivot_table(index='Type',values='Salary',aggfunc='std')
```

Out[13]:

| | | Salary |
|---|---|---|
| Type | | |
| Full-time | employee | 5773.502692 |

# EXP NO:3

Write python code to explain map (), filter (),reduce (), lambda()

***Lambda Function***

In [14]:
```
Max = lambda a, b : a if(a > b) else b
Max(4,5)
```

Out[14]: 5

In [15]:
```
x=lambda a:a+1
print(x(5))
```

```
6
```

In [16]:
```
x=lambda a,b,c:a+b+c
print(x(5,6,2))
```

```
13
```

### *Map Function*

In [19]:
```
def add4(x):
    return x+4
listl = [4,6,7,8,9]
list2 = list(map(add4,listl))
print(list2)
```

```
[8, 10, 11, 12, 13]
```

In [20]:
```
def starts_with_A(s):
    return s[0]=="A"
fruit=["Apple","banana","pear","Apricot","orange"]
map_object=map(starts_with_A,fruit)
print(list(map_object))
```

```
[True, False, False, True, False]
```

### *Filter Function*

In [21]:
```
def oddeven(x):
    if x%2 == 0:
        return True
    else:
        return False
listl = [4,5,6,7,8,9]
evenlist = list(filter(oddeven,listl))
print(evenlist)
```

```
[4, 6, 8]
```

In [22]:
```
def starts_with_A(s):
    return s[0]=="A"
fruit=["Apple","banana", "pear","Apricot","orange"]
filter_object=filter(starts_with_A,fruit)
print(list(filter_object))
```

```
['Apple', 'Apricot']
```

### *Reduce Function*

In [23]:
```
from functools import reduce
def sum(x,y):
    return x+y
list1 = [6,7,8,9]
s = reduce(sum,listl)
print(s)
```

```
30
```

# EXP NO:4

Ramesh decides to walk 10000 steps every day to combat the effect that lockdown has had on his body's agility, mobility, flexibility and strength.Consider the following data from fitness tracker over a period of 10 days.

In [24]:
```
import pandas as pd
arr={
    'Day_number' :[1,2,3,4,5,6,7,8,9,10],
    'Steps_Walked':[4335,9552,7332,4503,5335,7552,8332,6004,8965,7699]

df=pd.DataFrame(arr)
print(df)
```

```
   Day_number   Steps_Walked
0            1          4335
1            2          9552
2            3          7332
3            4          4503
4            5          5335
5            6          7552
6            7          8332
7            8          6004
8            9          8965
9           10          7699
```

- ***Code to add 1000 steps to all the observations***.

In [26]:
```
arr['steps_walked']=df['Steps_Walked']+1000
print(arr['steps_walked'])
```

```
0     5335
1    10552
2     8332
3     5503
4     6335
5     8552
6     9332
7     7004
8     9965
9     8699
Name: Steps_Walked, dtype: int64
```

***Code to find out the days on which Ramesh walked more than 7000 steps***.

```
days_more_than_7k=df.loc[df['Steps_Walked']>7000]
days_more_than_7k
```

Out [ 27 ] :

| | Day_number | Steps_Walked |
|---|---|---|
| 1 | 2 | 9552 |
| 2 | 3 | 7332 |
| 5 | 6 | 7552 |
| 6 | 7 | 8332 |
| 8 | 9 | 8965 |
| **9** | 10 | 7699 |

# EXP NO:5

Perform the **following** operations on car manufacturing company dataset auto-mpg. csv.Write code given below in Pand /numpy.

- ***Read data from auto-mpg.csv***

In [29] :

```
import pa ndas as pd
df=pd.re ad_csv(" D:\\Dataset\\Auto_r g.csv.csv")
df
```

Out[29] :

| | mpg | Cylinder | Displacement | HP | Weight | Accleartion | Modelyear | Origin | Car-name |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 18 | 4 | 300 | 130 | 3504 | 12.0 | 70 | 1 | Honda |
| 1 | 15 | 8 | 325 | 165 | 3645 | 15.5 | 71 | 1 | Nexon |
| 2 | 18 | 8 | 318 | 160 | 3440 | 11.0 | 70 | 1 | Ford |
| 3 | 16 | 6 | 305 | 160 | 3450 | 12.0 | 75 | 1 | Indica |
| 4 | 17 | 8 | 307 | 150 | 3449 | 10.5 | 80 | 1 | Swift |

- ***Give code to get all cars with 8 cylinders***.

In [30]:

```
matched=df['Cylinder']==8
new_df=df[matched]
new_df
```

Out [30]:

| | mpg | **Cylinder** | **Displacement** | **HP** | **Weight** | Accleartion | **Modelyear** | Origin | Car-name |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 15 | 8 | 325 | 165 | 3645 | 15.5 | 71 | 1 | Nexon |
| 2 | 18 | 8 | 318 | 160 | 3440 | 11.0 | 70 | 1 | Ford |
| 4 | 17 | 8 | 307 | 150 | 3449 | 10.5 | 80 | 1 | Swift |

- ***Get the number of Cars manufactured in each year***

In [31] :

```
df.groupby(['Modelyear']).count()
```

Out [31]:

| Modelyear | mpg | Cylinder | Displacement | HP | Weight | Accleartion | Origin | Car-name |
|---|---|---|---|---|---|---|---|---|
| 70 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 71 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 75 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 80 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

# EXP NO:6

**Use 'Cars93' dataset to answer the above questions.**
**The information that the columns of this dataset contain is given below:**

| Manufacturer | Model | Type | Price | MPG.city | MPG.highway | Horsepower | Rear.seat.room | Passengers |
|---|---|---|---|---|---|---|---|---|
| Manufacturer. | Model. | Type: a factor with levels "Small", "Sporty", " Compact", "Midsize", "Large" and "Van". | Midrange Price (in \$1,000). | City MPG (miles per US gallon by EPA rating). | Highway MPG. | Horsepower (maximum). | Rear seat room (inches) (missing for 2-seater vehicles). | Passenger capacity (persons) |

Create the following plots to visualize/summarize the data and customize it appropriately.

In [8]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
```

In [3]:
```
cars_df=pd.read_csv('/content/Cars93.csv')
cars_df
```

Out[3]:

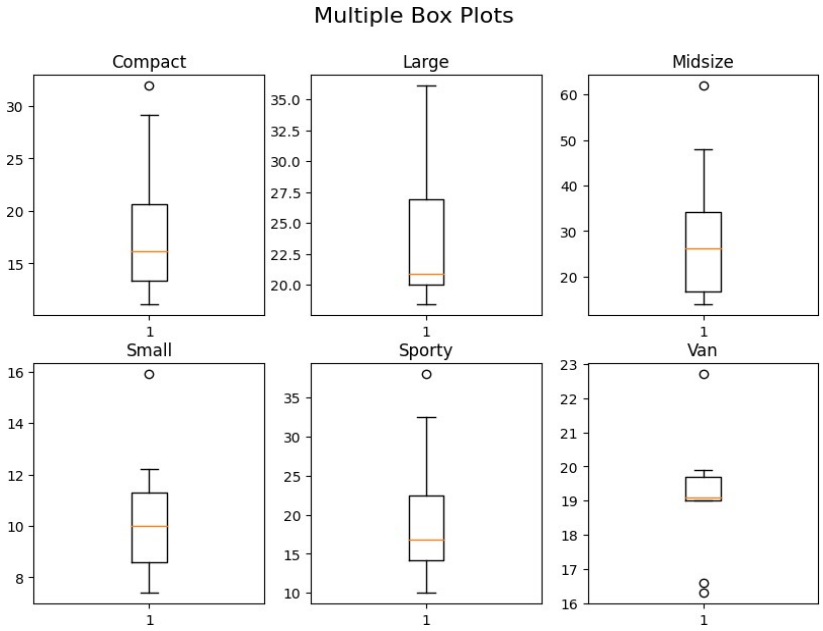
| | Unnamed: 0 | Manufacturer | Model | Type | Min.Price | Price | Max.Price | MPG.cityMPG.h |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | Acura | Integra | Small | 12.9 | 15.9 | 18.8 | 25 |
| **1** | 2 | Acura | Legend | Midsize | 29.2 | 33.9 | 38.7 | 18 |
| **2** | 3 | Audi | 90 | Compact | 25.9 | 29.1 | 32.3 | 20 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **91** | 92 | Volvo | 240 | Compact | 21.8 | 22.7 | 23.5 | 21 |
| **92** | 93 | Volvo | 850 | Midsize | 24.8 | 26.7 | 28.5 | 20 |

93 rows × 28 columns

**Use a box plot to determine the price range of all different car available in the market? Andinterpret the five-number summary**

In [4]:
```
fig,ax=plt.subplots(2,3)
fig.set_figwidth(10)
fig.set_figheight(7)
fig.suptitle("Multiple Box Plots",fontsize = 16)
ax[0][0].boxplot(cars_df["Price"][cars_df["Type"]=="Compac
t"]) ax[0][0].set_title('Compact')
ax[0][1].boxplot(cars_df["Price"][cars_df["Type"]=="Large"])
ax[0][1].set_title('Large')
ax[0][2].boxplot(cars_df["Price"][cars_df["Type"]=="Midsiz
e"]) ax[0][2].set_title('Midsize')
ax[1][0].boxplot(cars_df["Price"][cars_df["Type"]=="Small"])
ax[1][0].set_title('Small')
ax[1][1].boxplot(cars_df["Price"][cars_df["Type"]=="Sporty"])
ax[1][1].set_title('Sporty')
ax[1][2].boxplot(cars_df["Price"][cars_df["Type"]=="Van"])
```

Out[4]: Text(0.5, 1.0, 'Van')




**Five-Number Summary**

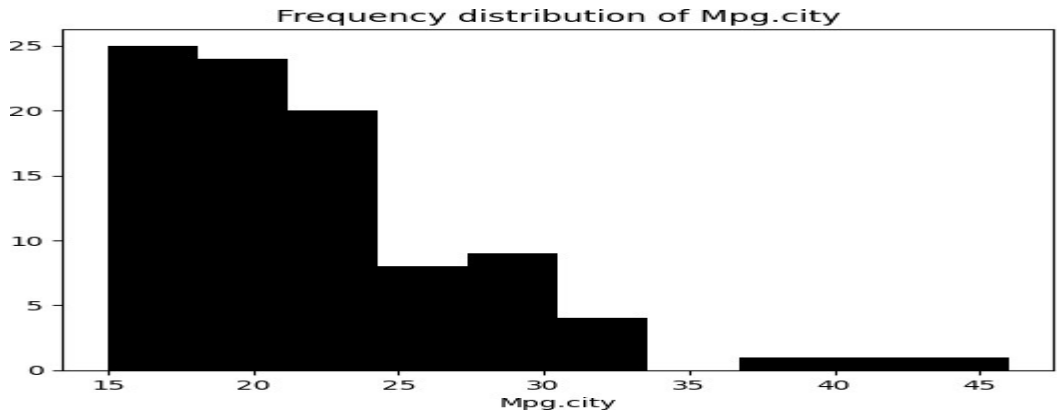
In [31]:
```
min_value = np.min(cars_df['Price'])
q1=np.percentile(cars_df['Price'],25) median
= np.median(cars_df['Price'])
q3=np.percentile(cars_df['Price'],75)
max_value = np.max(cars_df['Price'])

print('MinimumValue:',min_value)
print('Q1:',q1)
print('Median:',median)
print('Q3:',q3)
print('Maximum Value:',max_value)
```

```
Minimum Value: 7.4
Q1: 12.2
Median: 17.7
Q3: 23.3
Maximum Value: 61.9
```

**Histogram to check the frequency distribution of the variable 'Mpg.city'(Miles per gallon) and note down the interval having the highest frequency.**

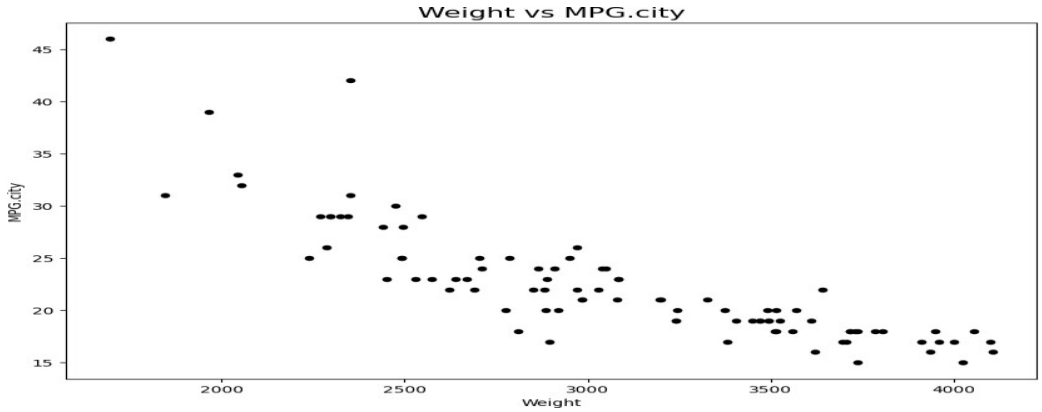
In [29]:
```
plt.hist(cars_df['MPG.city'],color='black') plt.xlabel('Mpg.city')
plt.title('FrequencydistributionofMpg.city') plt.show()
```




**Use a scatter plot to determine whether a car with higher horsepower gives lower mileage?**

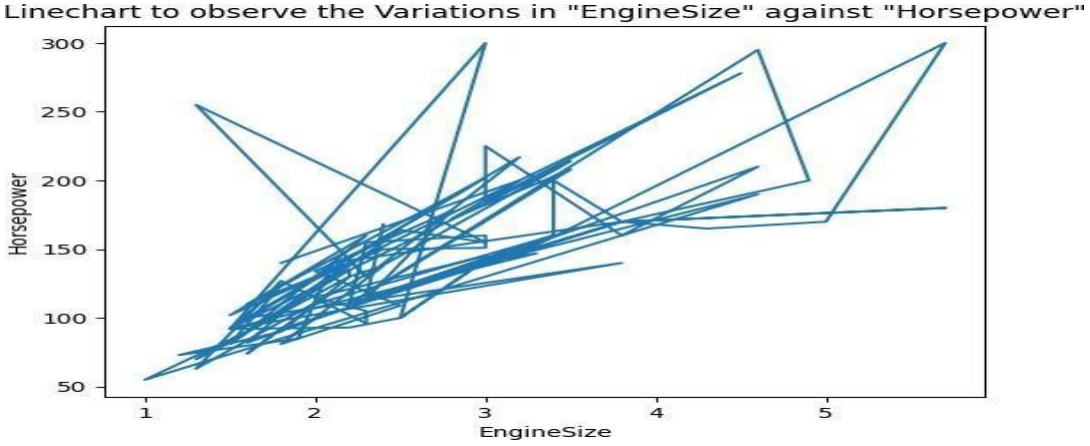
In [25]:
```
ax=cars_df.plot(["Weight"],["MPG.city"],kind="scatter",color="black",
ax.set_xlabel("Weight")
ax.set_ylabel("MPG.city")
ax.set_title("Weight vs MPG.city",fontsize = 16)
```

Out[25]: Text(0.5, 1.0, 'Weight vs MPG.city')




***Use a line chart to observe the variations in 'Engine Size', against 'Horsepower'***

In [30]:
```
plt.plot(cars_df['EngineSize'],cars_df['Horsepower'])
plt.xlabel('EngineSize')
plt.ylabel('Horsepower')
plt.title('LinecharttoobservetheVariationsin"EngineSize"against"Hor
plt.show()
```



***Create a git repository and push source code to the repo. Use the 'matcars.csv' dataset to answerthe above questions.***

### step 1:

Create a local directory using the following command:

$ mkdir test

$ cd test

### step 2:

The next step is to initialize the directory:

$ git init

### step 3:

Go to the folder where "test" is created and add 'matcars.csv'.Save and close the file.

### step 4:

Enter the Git bash interface and type in the following command to check the status:

$ git status

### step 5:

Add the "matcars.csv" to the current directory using the following command:

### step 6:

$ git add matcars.csv

### step 7:

Next, make a commit using the following command:

$ git commit -m "committing a text file"

### step 8:

Open your Github account and create a new repository with the name "test" and click on "Create repository." This is the remote repository. Next, copy the link of "test."

### step 9:

Go back to Git bash and link the remote and local repository using the following command:

$ git remote add origin

Here, is the link copied in the previous step.

### step 10:

Push the local file onto the remote repository using the following command:

$ git push origin master

### step 11:

Move back to Github and click on "test" and check if the local file "matcars.csv" is pushed to this repository

# EXP NO:7

**Use the 'matcars.csv' dataset to answer the above questions.**

| model | mpg | cyl | disp | hp | drat | wt | qsec | vs | am | gear |
|---|---|---|---|---|---|---|---|---|---|---|
| model name | Miles/(US) gallon | Number of cylinders | Displacement (cu.in.) | Gross horsepower | Rear axle ratio | Weight (1000 lbs) | 1/4 mile time | Engine (0 = V-shaped, 1 = straight) | Transmission (0 = automatic, 1 = manual) | Numb forwa gears |

***Create the following plots to visualize/summarize the data and customize it appropriately.***

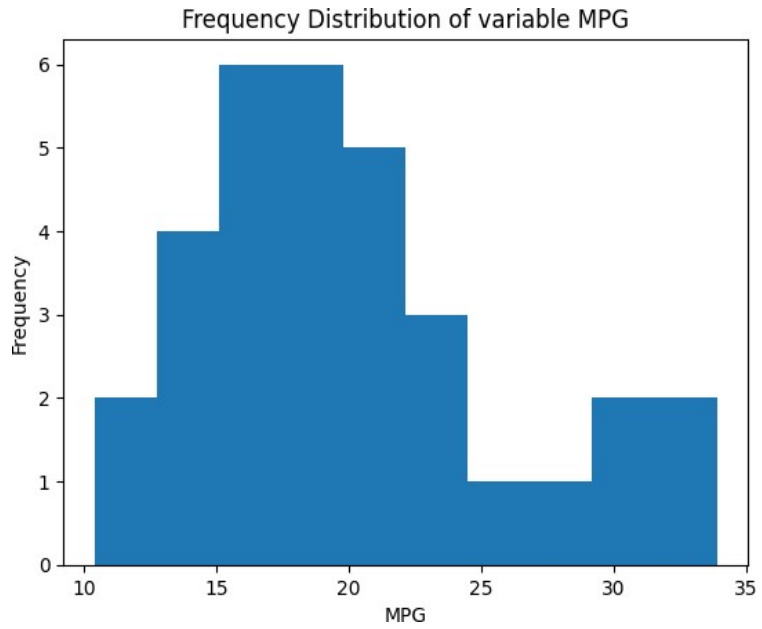
In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
cars_df=pd.read_csv("/content/mtcars(1).csv")
cars_df
```

Out[11]:

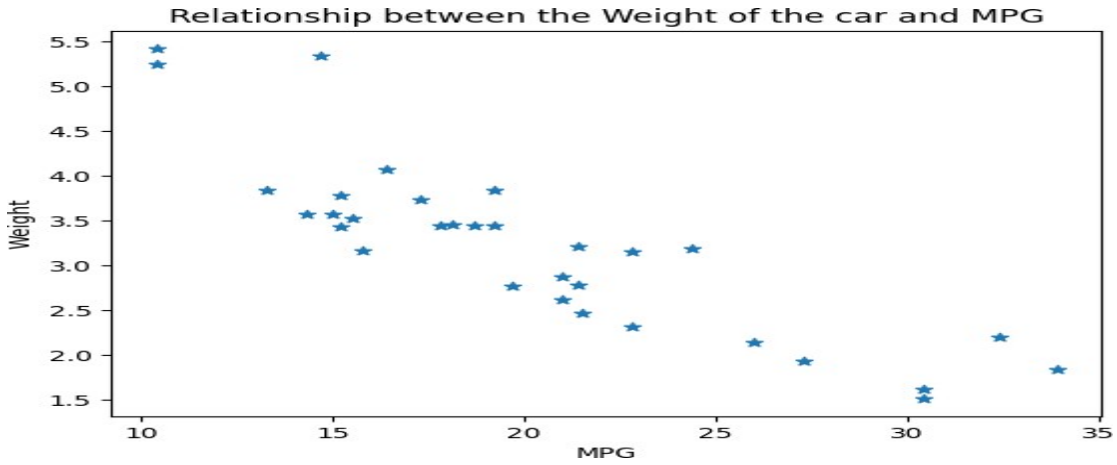
| | model | mpg | cyl | disp | hp | drat | wt | qsec | vs | am | gear | carb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Mazda RX4 | 21.0 | 6 | 160.0 | 110 | 3.90 | 2.620 | 16.46 | 0 | 1 | 4 | 4 |
| **1** | Mazda RX4 Wag | 21.0 | 6 | 160.0 | 110 | 3.90 | 2.875 | 17.02 | 0 | 1 | 4 | 4 |
| **8** | Merc 230 | 22.8 | 4 | 140.8 | 95 | 3.92 | 3.150 | 22.90 | 1 | 0 | 4 | 2 |
| **9** | Merc 280 | 19.2 | 6 | 167.6 | 123 | 3.92 | 3.440 | 18.30 | 1 | 0 | 4 | 4 |
| **10** | Merc 280C | 17.8 | 6 | 167.6 | 123 | 3.92 | 3.440 | 18.90 | 1 | 0 | 4 | 4 |

- ***Histogram to check the frequency distribution of the variable 'mpg'(Miles per gallon) andnote down the interval having the highest frequency***




*.Scatter plot to determine the relationship between the weight of the car and the mpg*

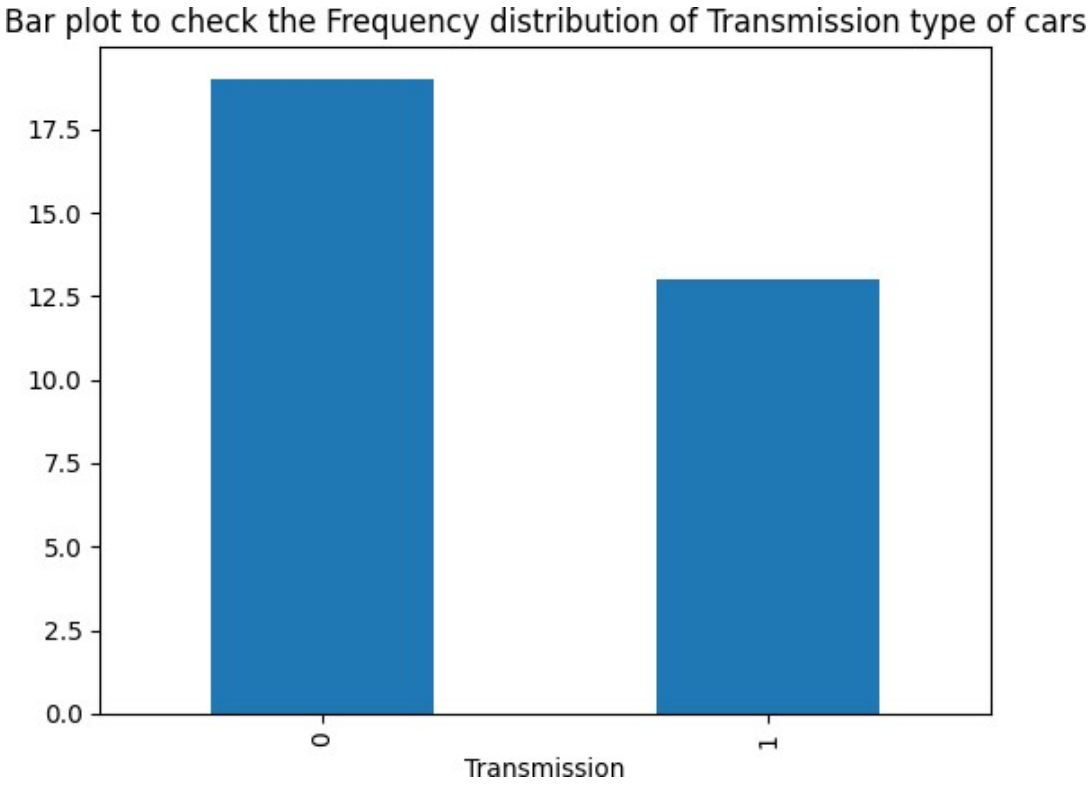
In [4]:
```
plt.scatter(cars_df['mpg'],cars_df['wt'],marker='*')
plt.xlabel("MPG")
plt.ylabel("Weight")
plt.title('RelationshipbetweentheWeightofthecarandMPG') plt.show()
```




***.Bar plot to check the frequency distribution of transmission type of cars***

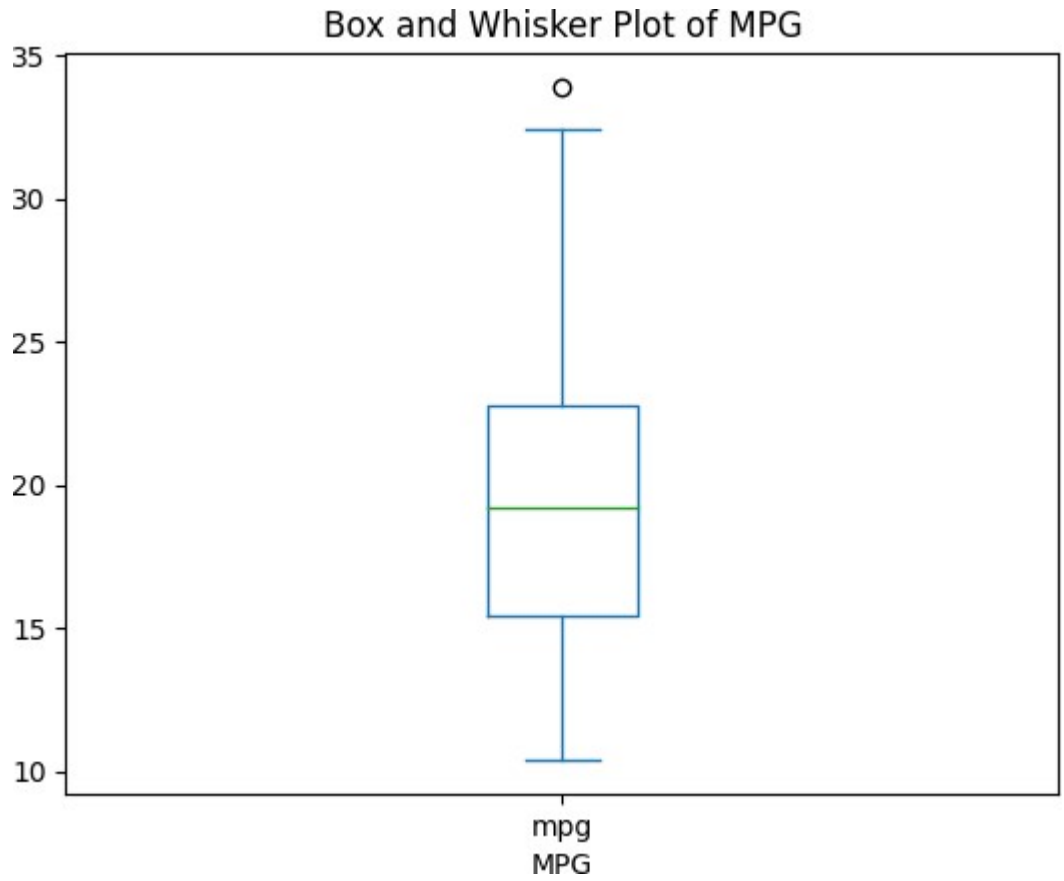
In [5]:
```
trans=cars_df['am'].value_counts()
print(trans)
trans.plot.bar()
plt.xlabel("Transmission")
plt.title('BarplottochecktheFrequencydistributionofTransmissiontyp
plt.show()
```

```
0    19
1    13
Name: am, dtype: int64
```



## Box and whisker plot to mpg and interpret the five number summary

In [7]:
```
cars_df['mpg'].plot(kind='box') plt.xlabel('MPG')
plt.title('BoxandWhiskerPlotofMPG') plt.show()
```




## Interpretfive-numbersummary

In [10]:
```
min_value = np.min(cars_df['mpg'])
q1=np.percentile(cars_df['mpg'],25) median
= np.median(cars_df['mpg'])
q3=np.percentile(cars_df['mpg'],75)
max_value = np.max(cars_df['mpg'])

print('MinimumValue:',min_value)
print('Q1:',q1)
print('Median:',median)
print('Q3:',q3)
print('Maximum Value:',max_value)
```

```
Minimum Value: 10.4
Q1: 15.425
Median: 19.2
Q3: 22.8
Maximum Value: 33.9
```

# EXP NO:8

For the given dataset perform the followingoperations:

In [1]:
```
# importing libraries
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
#Reading the data set
data=pd.read_csv("C:\\Users\\GPT-BANTWAL\\AI\\8.csv") data
```

Out[2]:

| | Number | Pencil | TextBooks | Drawing Sheet | TotalUnits | Profits |
|---|---|---|---|---|---|---|
| **0** | 1 | 300 | 250 | 100 | 800 | 8000 |
| **1** | 2 | 35 | 350 | 200 | 1000 | 9500 |
| **2** | 3 | 400 | 400 | 200 | 1320 | 10256 |
| **3** | 4 | 500 | 420 | 250 | 1510 | 12000 |
| **4** | 5 | 520 | 500 | 300 | 2000 | 18600 |

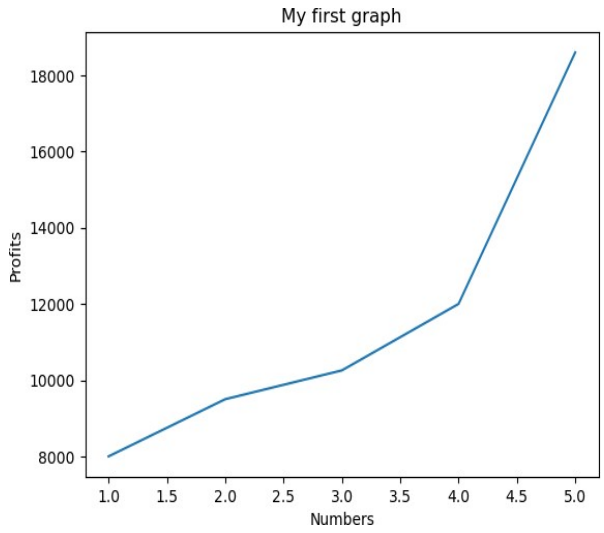
- ***Check Statistical info of the dataset***

In [3]:
```
data.info()
```

```
<class'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Number         5 non-null      int64
 1   Pencil         5 non-null      int64
 2   Text Books     5 non-null      int64
 3   Drawing Sheet  5 non-null      int64
 4   Total Units    5 non-null      int64
 5   Profits        5 non-null      int64
dtypes: int64(6)
memory usage: 372.0 bytes
```

- ***Plot a line chart/plot showing total profit on y- axis and numbers column on x-axis***

In [4]:
```
x = [1,2,3,4,5]
y = [8000,9500,10256,12000,18600]
plt.plot(x,y)
plt.xlabel('Numbers')
plt.ylabel('Profits')
plt.title('Myfirstgraph'
) plt.show()
```




### *Find the missing values*

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

Out[5]:
```
Number          0
Pencil          0
Text Books      0
Drawing Sheet   0
Total Units     0
Profits         0
dtype: int64
```

### *Find the sum of total Profits*

In [6]:
```
numbers=[8000,9500,10256,12000,18600]
total=sum(numbers)
print(total)
```

```
58356
```

### *Find the max value from drawing sheet column*

In [7]:
```
column = [100,200,200,250,300]
max_value=max(column)
max_value
```

Out[7]:
```
300
```

# EXP NO:9

**Perform the following operations on car manufacturing company dataset auto-mpg. csv. Write code given below in Pand /numpy.**

| | mpg | Cylinder | Displace-ment | HP | Weight | Accele-ration | Model year | Origin | Car-name |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 18 | 4 | 300 | 130 | 3504 | 12.0 | 70 | 1 | Honda |
| 1 | 15 | 8 | 325 | 165 | 3695 | 11.5 | 71 | 1 | Nexon |
| 2 | 18 | 8 | 318 | 160 | 3440 | 11.0 | 70 | 1 | Ford |
| 3 | 16 | 6 | 305 | 160 | 3450 | 12.0 | 75 | 1 | Indica |
| 4 | 17 | 8 | 307 | 150 | 3449 | 10.5 | 80 | 1 | Swift |

In [8]:
```
import pandas as pd
```

- ***Read data grom auto-mpg.csv***

In [9]:
```
#Reading the data set
data = pd.read_csv("C:\\Users\\GPT-BANTWAL\\AI\\9.csv")
data
```

Out[9]:

| | mpg | Cylinder | Displacement | HP | Weight | Acceleration | Model Year | Origin | Car name |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 18 | 4 | 300 | 130 | 3504 | 12.0 | 70 | 1 | Honda |
| **1** | 15 | 8 | 325 | 165 | 3695 | 11.5 | 71 | 1 | Nexon |
| **2** | 18 | 8 | 318 | 160 | 3440 | 11.0 | 70 | 1 | Ford |
| **3** | 16 | 6 | 305 | 160 | 3450 | 12.0 | 75 | 1 | Indica |
| **4** | 17 | 8 | 307 | 150 | 3449 | 10.5 | 80 | 1 | Swift |

- ***Give the code to get all cars with 8 cylinders***

In [10]:
```
cylinders = data[data['Cylinder'] ==8]
cylinders
```

Out[10]:

| | mpg | Cylinder | Displacement | HP | Weight | Acceleration | Model Year | Origin | Car name |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 15 | 8 | 325 | 165 | 3695 | 11.5 | 71 | 1 | Nexon |
| **2** | 18 | 8 | 318 | 160 | 3440 | 11.0 | 70 | 1 | Ford |
| **4** | 17 | 8 | 307 | 150 | 3449 | 10.5 | 80 | 1 | Swift |

- ***Get the number of cars manufactured in eachyear***

In [11]:
```
year = data.groupby("Model Year").size()
year
```

Out[11]:
```
Model Year
70    2
71    1
75    1
80    1
```

# EXP NO:10

## Write python code to imputation the missing values in the dataset

In [1]:
```
#Package imports
import pandas as pd
import missingno as msno
%matplotlib inline
```

In [2]:
```
#importing the required dataset
titanic_df = pd.read_csv("C:\\Users\\GPT BANTWAL\\titanic.csv")
titanic_df
```

Out[2]:

| | survived | pclass | sex | age | sibsp | parch | fare | embarked | class | who | adult_mal |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 | S | Third | man | Tru |
| **1** | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 | C | First | woman | Fals |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | . |
| **889** | 1 | 1 | male | 26.0 | 0 | 0 | 30.0000 | C | First | man | Tru |
| **890** | 0 | 3 | male | 32.0 | 0 | 0 | 7.7500 | Q | Third | man | Tru |

891 rows × 15 columns

In [3]:
```
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------------------------------------  -----------
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    object
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    object
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), float64(2), int64(4), object(7)
memory usage: 92.4+ KB
```

In [4]:
```
titanic_df.isnull()
```

Out[4]:

| | survived | pclass | sex | age | sibsp | parch | fare | embarked | class | who | adult_male | de |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | False | False | False | False | False | False | False | False | False | False | False | Tr |
| **1** | False | False | False | False | False | False | False | False | False | False | False | Fal |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **889** | False | False | False | False | False | False | False | False | False | False | False | Fal |
| **890** | False | False | False | False | False | False | False | False | False | False | False | Tr |

- **Deleting the entire row**

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

Out[5]:
```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

In [6]:
```
df = titanic_df.dropna(axis=0)
df.isnull().sum()
```

Out[6]:
```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 182 entries, 1 to 889
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------------------------------  -----
 0   survived     182 non-null    int64
 1   pclass       182 non-null    int64
 2   sex          182 non-null    object
 11  deck         182 non-null    object
 12  embark_town  182 non-null    object
 13  alive        182 non-null    object
 14  alone        182 non-null    bool
dtypes: bool(2), float64(2), int64(4), object(7)
memory usage: 20.3+ KB
```

- **Deleting the entire column**

```
In [8]: titanic_df.columns
```

```
Out[8]: Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'deck', 'embark_town',
       'alive', 'alone'],
      dtype='object')
```

```
In [9]: df = titanic_df.drop(['deck'],axis=1)
        df.isnull().sum()
```

```
Out[9]: survived         0
        pclass           0
        sex              0
        age            177
        sibsp            0
        parch            0
        fare             0
        embarked         2
        class            0
        who              0
        adult_male       0
        embark_town      2
        alive            0
        alone            0
        dtype: int64
```

**Imputing the Missing Value**

There are different ways of replacing the missing values.

- **Replacing With Arbitrary Value**

If you can make an educated guess about the missing value then you can replace it with some arbitrary value using the following code.

In [10]:
```
titanic_df['deck'].unique()
```

Out[10]:
```
array([nan, 'C', 'E', 'G', 'D', 'A', 'B', 'F'], dtype=object)
```

In [11]:
```
titanic_df['deck'] = titanic_df['deck'].fillna('C')
```

In [12]:
```
titanic_df['deck'].isnull().sum() #missing values replaced
```

Out[12]:
```
0
```

In [13]:
```
titanic_df
```

Out[13]:

| | survived | pclass | sex | age | sibsp | parch | fare | embarked | class | who | adult_mal |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 | S | Third | man | Tru |
| **1** | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 | C | First | woman | Fals |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **889** | 1 | 1 | male | 26.0 | 0 | 0 | 30.0000 | C | First | man | Tru |
| **890** | 0 | 3 | male | 32.0 | 0 | 0 | 7.7500 | Q | Third | man | Tru |

891 rows × 15 columns

- **Replacing With Mean**

  This is the most common method of imputing missing values of numeric columns.

In [15]:
```
mean = titanic_df['age'].mean()
print(mean)
titanic_df['age'] = titanic_df['age'].fillna(mean)
titanic_df['age']
```

```
29.69911764705882
```

Out[15]:
```
0      22.000000
1      38.000000
2      26.000000
         ...
889    26.000000
890    32.000000
Name: age, Length: 891, dtype: float64
```

- **Replacing With Mode**

In [17]:
```
titanic_df = pd.read_csv("C:\\Users\\GPT BANTWAL\\titanic.csv")
mode = titanic_df['deck'].mode()[0]
print(mode)
titanic_df['deck'] = titanic_df['deck'].fillna(mode)
```

```
In [18]: titanic_df['deck']
```

```
Out[18]: 0      C
         1      C
         2      C
         3      C
         4      C
               ..
         886    C
         887    B
         888    C
         889    C
         890    C
        Name: deck, Length: 891, dtype: object
```

- **Replacing With Median**

```
In [19]: titanic_df['age']= titanic_df['age'].fillna(titanic_df['age'].median())
         titanic_df['age']
```

```
Out[19]: 0      22.0
         1      38.0
         2      26.0
         3      35.0
         4      35.0
                ...
         886    27.0
         887    19.0
         888    28.0
         889    26.0
         890    32.0
        Name: age, Length: 891, dtype: float64
```

- **Forward and backward filling of missingvalues**

```
In [20]: titanic_df = pd.read_csv("C:\\Users\\GPT BANTWAL\\titanic.csv")
         titanic_df
```

Out[20]:

| | survived | pclass | sex | age | sibsp | parch | fare | embarked | class | who | adult_mal |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 | C | First | woman | Fals |
| **3** | 1 | 1 | female | 35.0 | 1 | 0 | 53.1000 | S | First | woman | Fals |
| **4** | 0 | 3 | male | 35.0 | 0 | 0 | 8.0500 | S | Third | man | Tru |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | . |
| **889** | 1 | 1 | male | 26.0 | 0 | 0 | 30.0000 | C | First | man | Tru |
| **890** | 0 | 3 | male | 32.0 | 0 | 0 | 7.7500 | Q | Third | man | Tru |

891 rows × 15 columns

In [21]:
```
new_df = titanic_df.fillna(method="ffill")
new_df
```

Out[21]:

| | survived | pclass | sex | age | sibsp | parch | fare | embarked | class | who | adult_mal |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 | S | Third | man | Tru |
| **1** | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 | C | First | woman | Fals |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | .. |
| **889** | 1 | 1 | male | 26.0 | 0 | 0 | 30.0000 | C | First | man | Tru |
| **890** | 0 | 3 | male | 32.0 | 0 | 0 | 7.7500 | Q | Third | man | Tru |

891 rows × 15 columns

In [22]:
```
new_df = titanic_df.fillna(method="ffill",limit=1)
new_df
```

Out[22]:

| | survived | pclass | sex | age | sibsp | parch | fare | embarked | class | who | adult_mal |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 | S | Third | man | Tru |
| **1** | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 | C | First | woman | Fals |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | .. |
| **889** | 1 | 1 | male | 26.0 | 0 | 0 | 30.0000 | C | First | man | Tru |
| **890** | 0 | 3 | male | 32.0 | 0 | 0 | 7.7500 | Q | Third | man | Tru |

891 rows × 15 columns

```
new_df = titanic_df.fillna(method="bfill")
new_df
```

| | survived | pclass | sex | age | sibsp | parch | fare | embarked | class | who | adult_mal |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 | S | Third | man | Tru |
| **1** | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 | C | First | woman | Fals |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | .. |
| **889** | 1 | 1 | male | 26.0 | 0 | 0 | 30.0000 | C | First | man | Tru |
| **890** | 0 | 3 | male | 32.0 | 0 | 0 | 7.7500 | Q | Third | man | Tru |

# EXP NO:11

Write python code to imputation the outliers in the dataset

In [2]:
```
#Importing the necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

In [3]:
```
titanic_df = pd.read_csv("C:\\Users\\GPT BANTWAL\\titanic.csv")
titanic_df
```

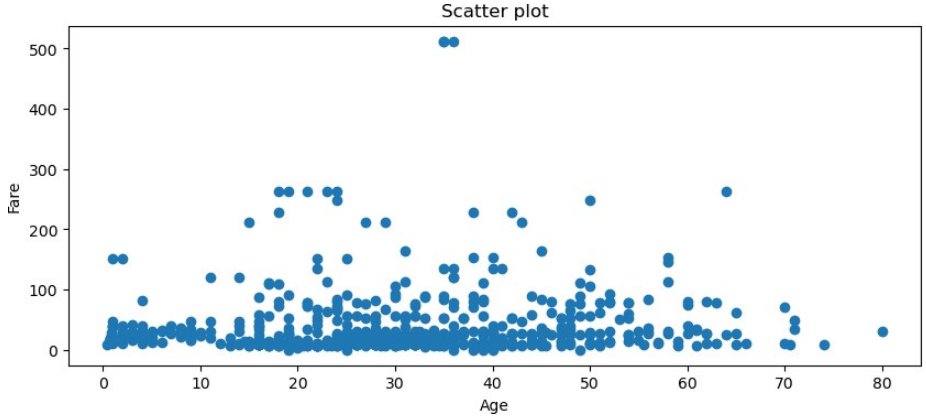
Out[3]:

| | survived | pclass | sex | age | sibsp | parch | fare | embarked | class | who | adult_mal |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 | S | Third | man | Tru |
| **1** | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 | C | First | woman | Fals |
| **2** | 1 | 3 | female | 26.0 | 0 | 0 | 7.9250 | S | Third | woman | Fals |
| **3** | 1 | 1 | female | 35.0 | 1 | 0 | 53.1000 | S | First | woman | Fals |
| **4** | 0 | 3 | male | 35.0 | 0 | 0 | 8.0500 | S | Third | man | Tru |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | . |
| **886** | 0 | 2 | male | 27.0 | 0 | 0 | 13.0000 | S | Second | man | Tru |
| **887** | 1 | 1 | female | 19.0 | 0 | 0 | 30.0000 | S | First | woman | Fals |
| **888** | 0 | 3 | female | NaN | 1 | 2 | 23.4500 | S | Third | woman | Fals |
| **889** | 1 | 1 | male | 26.0 | 0 | 0 | 30.0000 | C | First | man | Tru |
| **890** | 0 | 3 | male | 32.0 | 0 | 0 | 7.7500 | Q | Third | man | Tru |

891 rows × 15 columns

- ***Scatter plot to detect outliers***

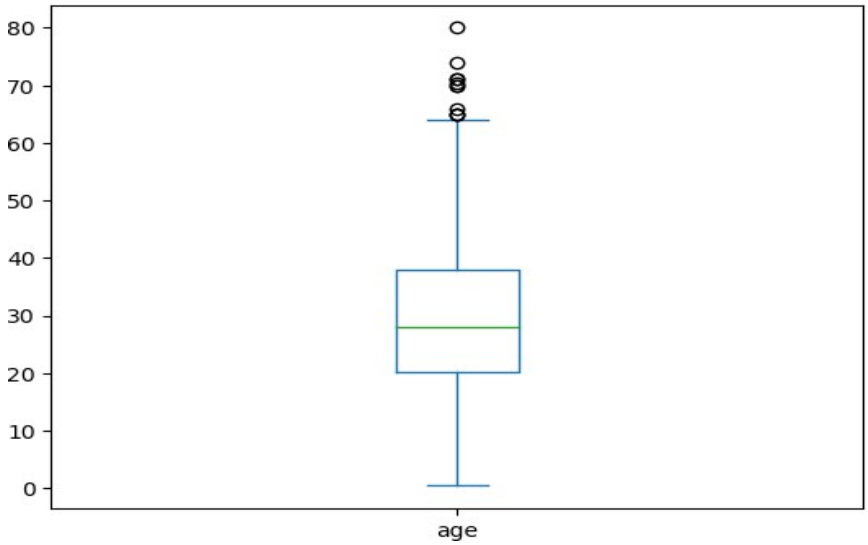
In [4]:
```
fig,ax = plt.subplots(figsize=(10,4))
ax.scatter(titanic_df['age'],titanic_df['fare'])
ax.set_xlabel('Age')
ax.set_ylabel('Fare')
plt.title("Scatter plot")
plt.show()
```



- ***Box plot to detect outliers***

In [5]:
```
titanic_df['age'].plot(kind='box')
```

Out[5]: <Axes: >




In [7]:
```
# finding the 1st quartile
q1 = titanic_df["age"].quantile(0.25)
# finding the 3rd quartile
q3 = titanic_df['age'].quantile(0.75)
# finding the iqr region
iqr = q3-q1
# finding upper and lower whiskers
upper_bound = q3+(1.5*iqr)
lower_bound = q1-(1.5*iqr)
```

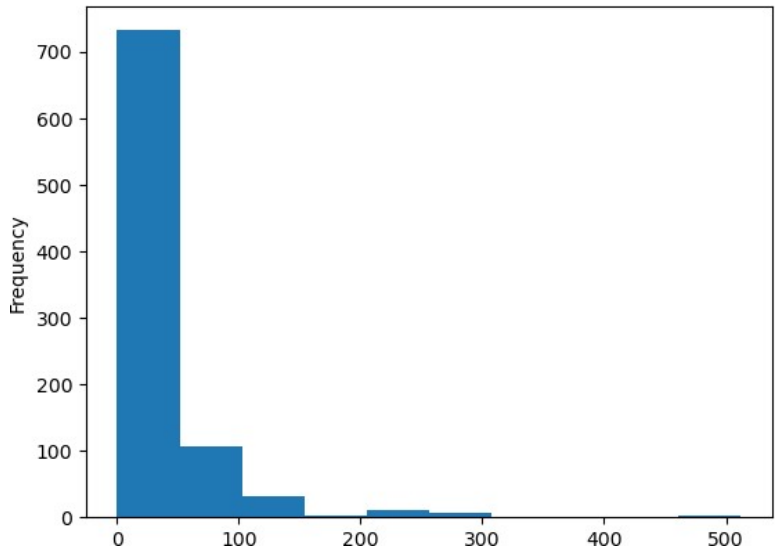
In [8]:
```
age_arr = titanic_df["age"]
outliers = age_arr[(age_arr <= lower_bound) | (age_arr >= upper_bound)]
print('The following are the outliers in the boxplot of age:\n',outliers)
```

```
The following are the outliers in the boxplot of
    age:
 33     66.0
54     65.0
96     71.0
116    70.5
280    65.0
456    65.0
493    71.0
630    80.0
672    70.0
745    70.0
851    74.0
Name: age, dtype: float64
```

- ***Histogram plot to detect outliers***

In [9]:
```
titanic_df['fare'].plot(kind='hist')
```

Out[9]:
```
<Axes: ylabel='Frequency'>
```




- ***Remove data objects with outliers***

In [10]:
```
upperIndex = titanic_df[titanic_df['age']>upper_bound].index
titanic_df.drop(upperIndex,inplace=True)
lowerIndex = titanic_df[titanic_df['age']<lower_bound].index
titanic_df.drop(lowerIndex,inplace=True)
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 880 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------------------------------------  -----------
 0   survived     880 non-null    int64
 1   pclass       880 non-null    int64
 2   sex          880 non-null    object
 3   age          703 non-null    float64
 4   sibsp        880 non-null    int64
 5   parch        880 non-null    int64
 6   fare         880 non-null    float64
 7   embarked     878 non-null    object
 8   class        880 non-null    object
 9   who          880 non-null    object
 10  adult_male   880 non-null    bool
 11  deck         198 non-null    object
 12  embark_town  878 non-null    object
 13  alive        880 non-null    object
 14  alone        880 non-null    bool
dtypes: bool(2), float64(2), int64(4), object(7)
memory usage: 98.0+ KB
```

- ***Replacing outliers with upper and lower cap:***

In [12]:
```
titanic_df = pd.read_csv("C:\\Users\\GPT BANTWAL\\titanic.csv")
```

In [13]:
```
#upper and lower cap
# Winzorization method
fare_arr = titanic_df["fare"]
upper_cap = np.percentile(fare_arr,1)
lower_cap = np.percentile(fare_arr,99)
outliers = fare_arr[(fare_arr < upper_cap) | (fare_arr > lower_cap)]
print('The following are the outliers in the boxplot of fare:\n',outliers)
```

```
The following are the outliers in the boxplot of fare:
 27     263.0000
88     263.0000
258    512.3292
311    262.3750
341    263.0000
438    263.0000
679    512.3292
737    512.3292
742    262.3750
Name: fare, dtype: float64
```

In [15]:
```
for i in titanic_df['fare']:
 if i<lower_bound :
    titanic_df['fare'] = titanic_df['fare'].replace(i,lower_cap)
 elif i>upper_bound :
     titanic_df['fare'] = titanic_df['fare'].replace(i,upper_cap)
```

In [16]:
```
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------------------------------------  -----------
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    object
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    object
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), float64(2), int64(4), object(7)
```

- ***Replacing outliers with Mean***

In [18]:
```
titanic_df = pd.read_csv("C:\\Users\\GPT BANTWAL\\titanic.csv")
m = np.mean(titanic_df['age'])
print('mean:',m)
for i in titanic_df['age']:
     if i<lower_bound or i>upper_bound :
         titanic_df['age'] = titanic_df['age'].replace(i,m)
```

```
mean: 29.69911764705882
```

- ***Replacing outliers with median***

In [20]:
```
titanic_df = pd.read_csv("C:\\Users\\GPT BANTWAL\\titanic.csv")
q1 = titanic_df["age"].quantile(0.25)
# finding the 3rd quartile
q3 = titanic_df['age'].quantile(0.75)
# finding the iqr region
iqr = q3-q1
# finding upper and lower whiskers
upper_bound = q3+(1.5*iqr)
lower_bound = q1-(1.5*iqr)
```

In [22]:
```
m = titanic_df['age'].median()
print(m)
for i in titanic_df['age']:
     if i<lower_bound or i>upper_bound :
         titanic_df['age'] = titanic_df['age'].replace(i,m)
```

```
Median=28.0
```

# EXP NO:12

**Assume a Boston housing dataset having two columns built_up_area (independent variable) and rent (dependent variable). Build linear regression model**

In [75]:
```
#importing libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

In [76]:
```
#Reading the dataset
df=pd.read_csv("C:\\Users\\GPTBANTWAL\\Downloads\\rent&biltuip.csv") df
```

Out[76]:

| | Built_up_area | Rent |
|---|---|---|
| **0** | 200 | 4000 |
| **1** | 350 | 5500 |
| **2** | 400 | 7000 |
| **3** | 450 | 8500 |
| **4** | 700 | 6500 |
| **5** | 750 | 7800 |
| **6** | 800 | 9000 |
| **7** | 1000 | 9500 |

In [77]:
```
df.info()
```

```
<class'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Built_up_area  20non-null      int64
 1   Rent           20 non-null     int64
dtypes: int64(2)
memory usage: 452.0 bytes
```

In [78]:
```
df.isnull().sum()
```

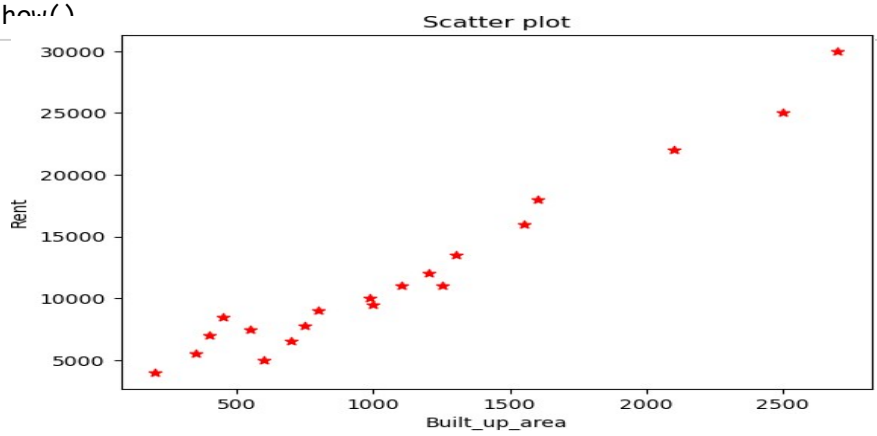
Out[78]:
```
Built_up_area      0
Rent               0
dtype: int64
```

In [79]:
```
df.head()
```

Out[79]:

| | Built_up_area | Rent |
|---|---|---|
| 0 | 200 | 4000 |
| 1 | 350 | 5500 |
| 2 | 400 | 7000 |
| 3 | 450 | 8500 |
| 4 | 700 | 6500 |

In [80]:
```
plt.scatter(df.Built_up_area,df.Rent,color='red',marker='*')
plt.xlabel('Built_up_area')
plt.ylabel('Rent')
plt.title('Scatterplot')
plt.show()
```




```
#droping the columns
x=df.drop('Rent',axis=1)
x
```

Out[81]:

| | Built_up_area |
|---|---|
| 0 | 200 |
| 1 | 350 |
| 2 | 400 |
| 3 | 450 |
| – | ---- |
| - | ---- |
| 17 | 1300 |
| 18 | 1550 |
| 19 | 2700 |

In [82]:
```
y=df.Rent
y
```

Out[82]:
```
0      4000
1      5500
2      7000
3      8500
4      6500
5      7800
6      9000
7      9500
8     12000
9     11000
10    10000
11     7500
12    11000
13    18000
```

- ***Splitting the dataset***

In [108]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.33,random_st
print("xtrain shape : ", xtrain.shape)
print("xtest shape : ", xtest.shape)
print("ytrainshape:",ytrain.shape)
print("ytest shape : ", ytest.shape)
```

```
xtrain shape :(13, 1)
xtestshape:(7,1)
ytrainshape:(13,)
ytest shape :(7,)
```

In [109]:
```
reg=linear_model.LinearRegression()
reg.fit(x,y)
y_pred=y_pred = reg.predict(xtest)
```

In [110]:
```
reg.predict([[650]])
```

```
C:\Users\GPTBANTWAL\anaconda3\Lib\site-packages\sklearn\base.py:464:UserWar
ning:Xdoesnothavevalidfeaturenames,butLinearRegressionwasfittedwi th feature
names
  warnings.warn( Out[110]:array([7492.

39323093])
```

In [111]:
```
reg.coef_
```

Out[111]:
```
array([9.7900215])
```

In [112]:
```
reg.intercept_
```

Out[112]:
```
1128.879253617697
```

In [113]:
```
z=9.7900215*650+1128.879253617697
z
```

Out[113]:
```
7492.393228617697
```

In [114]:
```
#R-Squared score of the model
train_score=reg.score(xtrain,ytrain)
test_score=reg.score(xtest,ytest)
print('TrainScore(R-Squared):',train_score)
print('Test Score (R-Squared)',test_score)
```

```
Train Score (R-Squared):  0.939368616808118
Test Score (R-Squared) 0.9776166105639195
```

In [115]:
```
#Mean squared error & mean absolute error of the model
mse=mean_squared_error(ytest,y_pred)
mae=mean_absolute_error(ytest,y_pred)
print("Mean Square Error : ", mse)
print("Mean Absolute Error : ", mae)
```

```
Mean Square Error : 1205277.000553029
Mean Absolute Error :  819.9778224058471
```

# EXP NO:13

## For Breast Cancer dataset build a Decision Tree machine learning model to predict or identify it and toperform the following operations.

In [1]:
```
#Import libraries
import pandas as pd
import matplotlib.pyplot as plt
df=pd.read_csv("C:\\Users\\GPT BANTWAL\\Downloads\\Breast_Cancer_data.csv")
df
```

Out[1]:

| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compact |
|---|---|---|---|---|---|---|---|---|
| **0** | 842302 | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | |
| **1** | 842517 | M | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | |
| **2** | 84300903 | M | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **566** | 926954 | M | 16.60 | 28.08 | 108.30 | 858.1 | 0.08455 | |
| **567** | 927241 | M | 20.60 | 29.33 | 140.10 | 1265.0 | 0.11780 | |
| **568** | 92751 | B | 7.76 | 24.54 | 47.92 | 181.0 | 0.05263 | |

569 rows × 33 columns

In [2]:
```
#Brief Overview
```

In [3]:
```
Df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 -   --  -----------
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
id                          0
diagnosis                   0
radius_mean                 0
texture_mean                0
concave points_worst        0
symmetry_worst              0
fractal_dimension_worst     0
Unnamed: 32               569
```

```
df.describe()
```

Out[5]:

| | id | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness |
|---|---|---|---|---|---|---|---|
| **count** | 5.690000e+02 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569. |
| **mean** | 3.037183e+07 | 14.127292 | 19.289649 | 91.969033 | 654.889104 | 0.096360 | 0. |
| **std** | 1.250206e+08 | 3.524049 | 4.301036 | 24.298981 | 351.914129 | 0.014064 | 0. |
| **min** | 8.670000e+03 | 6.981000 | 9.710000 | 43.790000 | 143.500000 | 0.052630 | 0. |
| **25%** | 8.692180e+05 | 11.700000 | 16.170000 | 75.170000 | 420.300000 | 0.086370 | 0. |
| **50%** | 9.060240e+05 | 13.370000 | 18.840000 | 86.240000 | 551.100000 | 0.095870 | 0. |
| **75%** | 8.813129e+06 | 15.780000 | 21.800000 | 104.100000 | 782.700000 | 0.105300 | 0. |
| **max** | 9.113205e+08 | 28.110000 | 39.280000 | 188.500000 | 2501.000000 | 0.163400 | 0. |

8 rows × 32 columns

In [6]:
```
df.column()
```

Out[6]:
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean','concave
      points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se','compactness_se',
       'concavity_se', 'concave points_se', 'symmetry_se', 'fractal_dimension_se', 'radius_worst',
       'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst','symmetry_worst',
       'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

In [11]:
```
#Preprocessing
```

In [12]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['diagnosis']=le.fit_transform(df['diagnosis'])
df['diagnosis']
```

Out[12]:
```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int64
```

In [13]:
```
x=df.drop(['diagnosis','Unnamed: 32'],axis=1)
x
```

Out[13]:

| | id | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean |
|---|---|---|---|---|---|---|---|
| **0** | 842302 | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 |
| **1** | 842517 | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 |
| **2** | 84300903 | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 |
| **3** | 84348301 | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 |
| **4** | 84358402 | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **564** | 926424 | 21.56 | 22.39 | 142.00 | 1479.0 | 0.11100 | 0.11590 |
| **565** | 926682 | 20.13 | 28.25 | 131.20 | 1261.0 | 0.09780 | 0.10340 |
| **566** | 926954 | 16.60 | 28.08 | 108.30 | 858.1 | 0.08455 | 0.10230 |
| **567** | 927241 | 20.60 | 29.33 | 140.10 | 1265.0 | 0.11780 | 0.27700 |
| **568** | 92751 | 7.76 | 24.54 | 47.92 | 181.0 | 0.05263 | 0.04362 |

569 rows × 31 columns

In [14]:
```
y=df['diagnosis']
y
```

Out[14]:
```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int64
```

In [ ]:
```
#Split the dataset
```

In [15]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.23,random_state=0)
```

In [16]:
```
print("x_train shape:",xtrain.shape)
print("x_test shape:",xtest.shape)
print("y_train shape:",ytrain.shape)
print("y_test shape:",ytest.shape)
```

```
x_train shape: (438, 31)
x_test shape: (131, 31)
y_train shape: (438,)
y_test shape: (131,)
```

In [17]:
```
#DecisionTree model()
```

In [18]:
```
from sklearn.tree import DecisionTreeClassifier
model=DecisionTreeClassifier(random_state=1)
model.fit(xtrain,ytrain)
```

Out[18]:
```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=1)
```

In [ ]:
```
#Find the accuarcy
```

In [26]:
```
train_accuracy=model.score(xtrain,ytrain)
train_accuracy
```

Out[26]:
```
1.0
```

In [27]:
```
test_accuracy=model.score(xtest,ytest)
test_accuracy
```

Out[27]:
```
0.916030534351145
```

In [28]:
```
#Data Prediction
```

In [29]:
```
train_predictions=model.predict(xtrain)
train_predictions
```

Out[29]:
```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, ∋, 0,
       0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, ∋, 1,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, ∋, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, ∋, 1,
       1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, ∋, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, ∋, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, ∋, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, ∋, 1,
       1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, ∋, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, ∋, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, ∋, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, ∋, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0],
      dtype=int64)
```

In [30]:
```
test_predictions=model.predict(xtest)
print(test_predictions)
```

```
[1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 0
 1 0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 0 0 0 1 0 1 1 1 1 0 0 1 1 1 1 1 0 0 1 0 1
 1 0 0 0 0 0 1 1 1 0 1 0 0 0 1 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0
 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1]
```

# EXP NO:14

## Natural language processing

In [82]:
```
import nltk
from nltk.tokenize import sent_tokenize
file=open("nlp.txt","r")
text=file.read()
print(text)
```

```
Natural language processing 1refers to the branch of computer @science andmore specifically, the
branch of 3" artificial intelligence" or AI, concerned with giving computers the ability to
understand text and spoken words in much the same way human beings can. NLP combines
computational linguistics with statistical, [machine learning, and deep learning models. Togethe
r, these technologies enable computers to process human language in the form of text or voice
data and to understand its full meaning, complete withthe speaker or writer's intent and
sentiment.
```

In [83]:
```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to C:\Users\GPT
[nltk_data]     BANTWAL\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[83]: True

### Tokenization

In [84]:
```
sentences=sent_tokenize(text)
```

In [85]:
```
print("number opf sentences:",len(sentences))
for i in range(len(sentences)):
    print("\n sentences",i+1,":\n",sentences[i])
```

```
number opf sentences: 3
sentences 1 :
 Natural language processing 1refers to the branch of computer @science an d more specifically,
the branch of 3" artificial intelligence" or AI, concerned with giving computers the ability to
understand text and spoken words in much the same way human beings can.

 sentences 2 :
 NLP combines computational linguistics with statistical, [machine learnin g, and deep learning
models.

 sentences 3 :
 Together, these technologies enable computers to process human language in the form of text or
voice data and to understand its full meaning, complete with the speaker or writer's intent and
sentiment.
```

### word tokenization

In [86]
```
from nltk.tokenize import word_tokenize
words=word_tokenize(text)
print("total number of words:",len(words))
print(words)
```

```
 total number of words: 99
['Natural', 'language', 'processing', '1refers', 'to', 'the', 'branch', 'o
f', 'computer', '@', 'science', 'and', 'more', 'specifically', ',', 'the',
'branch', 'of', '3', "''", 'artificial', 'intelligence', "''", 'or', 'AI',
',', 'concerned', 'with', 'giving', 'computers', 'the', 'ability', 'to',
'understand', 'text', 'and', 'spoken', 'words', 'in', 'much', 'the', 'same', 'way', 'human',
'beings', 'can', '.', 'NLP', 'combines', 'computational', 'linguistics', 'with', 'statistical', ',',
'[', 'machine', 'learning',',', 'and', 'deep', 'learning', 'models', '.', 'Together', ',', 'these',
'technologies', 'enable', 'computers', 'to', 'process', 'human', 'language', 'in', 'the', 'form',
'of', 'text', 'or', 'voice', 'data', 'and', 'to','understand', 'its', 'full', 'meaning', ',',
'complete', 'with', 'the', 'speaker', 'or', 'writer', "'s", 'intent', 'and', 'sentiment', '.']
```

### creating the frequency Dist

In [87]:
```
from nltk.probability import FreqDist
all_fdist=FreqDist(words)
all_fdist['to']
```

Out[87]: 4

### converting text to lower

In [88]:
```
text=text.lower()
print(text)
```

```
natural language processing 1refers to the branch of computer @science andmore specifically, the
branch of 3" artificial intelligence" or ai, concerned with giving computers the ability to
understand text and spoken words in much the same way human beings can. nlp combine computational
linguistics with statistical, [machine learning, and deep learning models. together, these
technologies enable computers to process human language in the form of text or voice data and to
understand its full meaning, complete withthe speaker or writer's intent and sentiment.
```

## removing the sepcial character

In [89]:
```
import re
text=re.sub('[^A-Za-z0-9]+',' ',text)
print(text)
```

```
natural language processing 1refers to the branch of computer science and more specifically the
branch of 3 artificial intelligence or ai concerned with giving computers the ability to
understand text and spoken words in much the same way human beings can nlp combines computational
linguistics with statistical machine learning and deep learning models together these
technologies enable computers to process human language in the form of textor voice data and to
understand its full meaning complete with the speakeror writer s intent and sentiment
```

### Removing words with Numbers

In [90]:
```
text=re.sub("\S*\d\S* ","",text).strip()
print(text)
```

```
natural language processing to the branch of computer science and more specifically the branch of
artificial intelligence or ai concerned with giving computers the ability to understand text and
spoken words in much the same way human beings can nlp combines computational linguistics with
statistical machine learning and deep learning models together these technologies enable
computers to process human language in the form of text or voice data and to understand its full
meaning complete with the speaker or writer s intent and sentiment
```

In [91]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to C:\Users\GPT
 [nltk_data]         BANTWAL\AppData\Roaming\nltk_data...
[nltk_data]       Package stopwords is already up-to-date!
```

Out[91]: True

### removing stop words

In [92]:
```
import nltk
words=word_tokenize(text)
stopwords=nltk.corpus.stopwords.words('english')
words_sw_removed=[]
for words in words:
    if words in stopwords:
        pass
    else:
        words_sw_removed.append(words)
print(word_tokenize(text))
```

```
['natural', 'language', 'processing', 'branch', 'computer', 'science', 'more', 'specifically',
'branch', 'artificial', 'intelligence', 'ai', 'concerned', 'giving', 'computers', 'ability',
'understand', 'text', 'spoken', 'words', 'much', 'same', 'way', 'human', 'beings', 'can', 'nlp',
'combines', 'computational', 'linguistics', 'with', 'statistical', 'machine', 'learning', 'deep',
'learning', 'models', 'together', 'these', 'technologies', 'enable', 'computers', 'process',
'human', 'languag e', 'text', 'voice', 'data','understand', 'its', 'full', 'meaning', 'complete',
'speaker', 'writer', 's', 'intent', 'sentiment']
```

### Spelling correction

In [93]:
```
import nltk
from nltk.metrics.distance import edit_distance
nltk.download('words')
from nltk.corpus import words
correct_words=words.words()
```

```
[nltk_data] Downloading package words to C:\Users\GPT
[nltk_data]        BANTWAL\AppData\Roaming\nltk_data...
[nltk_data]     Package words is already up-to-date!
```

In [94]:
```
incorrect_words=["happpy","amazaing","intelliengt"]
for word in incorrect_words:
    temp=[(edit_distance(word,w),w)for w in correct_words if w[0]==word[0]]
    print(sorted(temp,key=lambda val:val[0])[0][1])
```

```
happy
amazing
intelligent
```

## Normalization

### Stemming

In [95]:
```
from nltk.tokenize import word_tokenize
file=open("nlp.txt","r")
text=file.read()
text=text.lower()
import re
text=re.sub('[^A-Za-z0-9]+',' ',text)
text=re.sub("\S*\d\S* ","",text).strip()
print(text)
```

```
 natural language processing to the branch of computer science and more specifically the branch of artificial
intelligence or ai concerned with giving computers the ability to understand text and spoken words in much the
same way human beings can nlp combines computational linguistics with statistical machine learning and deep
learning models together these technologies enable computers to process human language in the form of text or
voice data and to understand its full meaning complete with the speaker or writer s intent and sentiment
```

In [96]:
```
word=word_tokenize(text,preserve_line=True)
print(word)
```

```
['natural', 'language', 'processing', 'to', 'the', 'branch', 'of', 'comput
er', 'science', 'and', 'more', 'specifically', 'the', 'branch', 'of', 'art
ificial', 'intelligence', 'or', 'ai', 'concerned', 'with', 'giving', 'comp
uters', 'the', 'ability', 'to', 'understand', 'text', 'and', 'spoken', 'wo
rds', 'in', 'much', 'the', 'same', 'way', 'human', 'beings', 'can', 'nlp','combines',
'computational', 'linguistics', 'with', 'statistical', 'machine', 'learning', 'and',
'deep', 'learning', 'models', 'together', 'these', 'technologies', 'enable',
'computers', 'to', 'process', 'human', 'languag e', 'in', 'the', 'form', 'of', 'text',
'or', 'voice', 'data', 'and', 'to','understand', 'its', 'full', 'meaning', 'complete',
'with', 'the', 'speake
r', 'or', 'writer', 's', 'intent', 'and', 'sentiment']
```

In [97]:

```
import nltk
words=word_tokenize(text)
stopwords=nltk.corpus.stopwords.words('english')
words_sw_removed=[]
for words in words:
    if words in stopwords:
        pass
    else:
        words_sw_removed.append(words)
print(word_tokenize(text))
```

```
['natural', 'language', 'processing', 'to', 'the', 'branch', 'of', 'computer', 'science', 'and',
'more', 'specifically', 'the', 'branch', 'of', 'artificial', 'intelligence', 'or', 'ai',
'concerned', 'with', 'giving', 'computers', 'the', 'ability', 'to', 'understand', 'text', 'and',
'spoken', 'words', 'in', 'much', 'the', 'same', 'way', 'human', 'beings', 'can', 'nlp','combines',
'computational', 'linguistics', 'with', 'statistical', 'machine', 'learning', 'and', 'deep',
'learning', 'models', 'together', 'these', 'technologies', 'enable', 'computers', 'to', 'process',
'human', 'languag e', 'in', 'the', 'form', 'of', 'text', 'or', 'voice', 'data', 'and', 'to',
'understand', 'its', 'full', 'meaning', 'complete', 'with', 'the', 'speaker', 'or', 'writer', 's',
'intent', 'and', 'sentiment']
```

In [98]:

```
import nltk
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

ps = PorterStemmer()
text=open("nlp.txt","r")
text=text.read()
words = word_tokenize(text)

for w in words:
    print(w, " : ", ps.stem(w))
```

```
Natural       :    natur
language       :    languag
processing    :    process1refers
              :    1refer
to    :    to
the     :    the
branch     : branchof :
      of
computer :      comput@
     :    @
science      : sciencand
        :    and
more     :    more
specifically          :    specif
,    :    ,
the     :    the
branch     : branchof :
      of
3    :    3
''    :    ''
artificial         :    artifici
intelligence          :    intellig''
      :    ''
or    :    or
AI    :    ai
,    :    ,
concerned:    concern with
         :    with
giving       :    give
computers    :    computthe
        :    the
ability    :  abil to
       :    to
understand    :    understandtext
         :    text
and     :    and
spoken       :    spoken
words      : word
in    :    in
much      :    much
the     : the
same      :    same
way     : way
```

# lemmataization

In [99]:
```
import nltk
from nltk.stem.wordnet import WordNetLemmatizer
Lemmatizer= WordNetLemmatizer()
lem_sent=[Lemmatizer.lemmatize(words_sent)for words_sent in words]
print(lem_sent)
```

```
['Natural', 'language', 'processing', '1refers', 'to', 'the', 'branch', 'o
f', 'computer', '@', 'science', 'and', 'more', 'specifically', ',', 'the',
'branch', 'of', '3', "'", 'artificial', 'intelligence', "'", 'or', 'AI',
',', 'concerned', 'with', 'giving', 'computer', 'the', 'ability', 'to', 'understand', 'text', 'and',
'spoken', 'word', 'in', 'much', 'the', 'same','way', 'human', 'being', 'can', '.', 'NLP', 'combine',
'computational', 'linguistics', 'with', 'statistical', ',', '[', 'machine', 'learning', ',',
'and', 'deep', 'learning', 'model', '.', 'Together', ',', 'these', 'techno
logy', 'enable', 'computer', 'to', 'process', 'human', 'language', 'in',
'the', 'form', 'of', 'text', 'or', 'voice', 'data', 'and', 'to', 'understand', 'it', 'full',
'meaning', ',', 'complete', 'with', 'the', 'speaker','or', 'writer', "'s", 'intent', 'and',
'sentiment', '.']
```

## N-grams

In [100]:
```
from nltk.util import ngrams
sentence="Natural language processing 1refers to the branch of computer @sc
bigram=ngrams(sentence.split(),2)
for item in bigram:
    print(item)
```

```
('Natural', 'language')

('language', 'processing')
('processing', '1refers')
('1refers', 'to')
('to', 'the')
('the', 'branch')
('branch', 'of')
('of', 'computer')
('computer', '@science')
('@science', 'and')
('and', 'more')
('more', 'specifically,')
('specifically,', 'the')
('the', 'branch')
('branch', 'of')
('of', '3')
('3', 'artificial')
('artificial', 'intelligence')
('intelligence', 'or')
('or', 'AI,')
('AI,', 'concerned')
('concerned', 'with')
('with', 'giving')
 ('in', 'much')
('much', 'the')
('the', 'same')
('same', 'way')

   ----------------
 ('or', "writer's")
("writer's", 'intent')
('intent', 'and')
('and', 'sentiment.')
```

## stop words

In [102]:
```
from nltk.tokenize import word_tokenize
word=word_tokenize(text)
print(word)
```

```
['Natural', 'language', 'processing', '1refers', 'to', 'the', 'branch', 'of', 'computer', '@',
'science', 'and', 'more', 'specifically', ',', 'the',branch', 'of', '3', "'", 'artificial',
'intelligence', "'", 'or', 'AI',',', 'concerned', 'with', 'giving', 'computers', 'the', 'ability',
'to','understand', 'text', 'and', 'spoken', 'words', 'in', 'much', 'the', 'same', 'way', 'human',
'beings', 'can', '.', 'NLP', 'combines', 'computational', 'linguistics', 'with', 'statistical', ',',
'[', 'machine', 'learning',',', 'and', 'deep', 'learning', 'models', '.', 'Together', ',', 'these',
'technologies', 'enable', 'computers', 'to', 'process', 'human', 'language', 'in', 'the', 'form',
'of', 'text', 'or', 'voice', 'data', 'and', 'to','understand', 'its', 'full', 'meaning', ',',
'complete', 'with', 'the', 'speaker', 'or', 'writer', "'s", 'intent', 'and', 'sentiment', '.'
```

In [103]:
```
import nltk
words=word_tokenize(text)
stopwords=nltk.corpus.stopwords.words('english')
words_sw_removed=[]
for words in words:
    if words in stopwords:
        pass
    else:
        words_sw_removed.append(words)
print(words_sw_removed)
```

```
['Natural', 'language', 'processing', '1refers', 'branch', 'computer',
'@', 'science', 'specifically', ',', 'branch', '3', "'", 'artificial', 'intelligence',
"'", 'AI', ',', 'concerned', 'giving', 'computers', 'ability', 'understand', 'text',
'spoken', 'words', 'much', 'way', 'human', 'beings', '.', 'NLP'combines','computational',
'linguistics', 'statistical', ',', '[', 'machine', 'learning', ',', 'deep', 'learning',
'models','.', 'Together', ',', 'technologies', 'enable', 'computers', 'process', 'human',
'language', 'form', 'text', 'voice', 'data', 'understand', 'full','meaning', ',',
'complete', 'speaker', 'writer', "'s", 'intent', 'sentiment', '.']
```

## after performing stop words

In [105]:
```
from nltk.util import ngrams
sentence="Natural language processing 1refers to the branch of computer @sc
bigram=ngrams(words_sw_removed,2)
for item in bigram:
    print(item)
```

```
('Natural', 'language')
('language', 'processing')
('processing', '1refers')
('1refers', 'branch')
('branch', 'computer')
('computer', '@')
('@', 'science') ('science',
'specifically')
('specifically', ',')
(',', 'branch')
('branch', '3')
('3', "'")
("'", 'artificial')
('artificial', 'intelligence')
('intelligence', "'")
("'", 'AI')
('AI', ',')
(',', 'concerned')
)
('statistical', ',')
(',', '[')
('[', 'machine')
('machine', 'learning')
```

```
om nltk.util import ngrams
sentence="Natural language processing 1refers to the branch of computer @sc
bigram=ngrams(words_sw_removed,3)
for item in bigram:
    print(item)
```

```
('Natural', 'language', 'processing') ('language', 'processing',
'1refers') ('processing', '1refers', 'branch')
('1refers', 'branch', 'computer')
('branch', 'computer', '@')
('computer', '@', 'science')
('@', 'science', 'specifically')
('science', 'specifically', ',')
('specifically', ',', 'branch')
(',', 'branch', '3')
('branch', '3', "'")
('3', "'", 'artificial')
("'", 'artificial', 'intelligence')
('artificial', 'intelligence', "'")
('intelligence', "'", 'AI')
("'", 'AI', ',')
('AI', ',', 'concerned')
(',', 'concerned', 'giving')
('concerned', 'giving', 'computers')('giving','computers',
'ability')
('computers', 'ability', 'understand')('ability','understand',
'text')
```

### Replacing punctuions by a single space

In [107]:
```
import re
text=re.sub('[^A-Za-z0-9]+',' ',text)
print(text)
```

```
Natural language processing 1refers to the branch of computer science and more specifically the
branch of 3 artificial intelligence or AI concerned with giving computers the ability to
understand text and spoken words in much the same way human beings can NLP combines computational
linguistics with statistical machine learning and deep learning models Together these t
echnologies enable computers to process human language in the form of textor voice data and to
understand its full meaning complete with the speakeror writer s intent and sentiment
```

### after performing puntuations

In [109]:
```
from nltk.util import ngrams
bigram=ngrams(text.split(),2)
for item in bigram:
    print(item)
```

```
('Natural', 'language')
('language', 'processing')
('processing', '1refers')
('1refers', 'to')
('to', 'the')
('the', 'branch')
('branch', 'of')
('of', 'computer')
('computer', 'science')
('science', 'and')
('and', 'more')
('more', 'specifically')
('specifically', 'the')
('the', 'branch')
('branch', 'of')
('of', '3')
('3', 'artificial')
('artificial', 'intelligence')
('intelligence', 'or')
```

```
('or', 'AI')
```

In [110]:
```
from nltk.util import ngrams
bigram=ngrams(text.split(),3)
for item in bigram:
    print(item)
```

```
('Natural', 'language', 'processing')
  ('to', 'the', 'branch')
 ('the', 'branch', 'of')
 ('branch', 'of', 'computer')
 ('of', 'computer', 'science')
 ('computer', 'science', 'and')
 ('science', 'and', 'more')
 ('and', 'more', 'specifically')
 ('more', 'specifically', 'the')
 ('specifically', 'the', 'branch')
 ('the', 'branch', 'of')
 ('branch', 'of', '3')
 ('of', '3', 'artificial')
 ('3', 'artificial', 'intelligence')
 ('artificial', 'intelligence', 'or')
 ('spoken', 'words', 'in')
```

# *CNN MODEL BREAST CANCER*

In [48]:
```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv1D, Flatten,Dense,Dropout
from tensorflow.keras.optimizers import Adam
```

In [49]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [50]:
```
from sklearn import datasets,metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [51]:
```
cancerData = datasets.load_breast_cancer()
```

In [52]:
```
X = pd.DataFrame(data = cancerData.data, columns=cancerData.feature_names )
X.head()
```

Out[52]:

| | mean radius | mean texture | mean perimeter | mean area | mean smoothness | mean compactness | mean concavity | mean concave points | mean symmetry | mean fractal dimension | ... | worst radius te |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.3001 | 0.14710 | 0.2419 | 0.07871 | ... | 25.38 |
| **1** | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.0869 | 0.07017 | 0.1812 | 0.05667 | ... | 24.99 |
| **2** | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.1974 | 0.12790 | 0.2069 | 0.05999 | ... | 23.57 |
| **3** | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.2414 | 0.10520 | 0.2597 | 0.09744 | ... | 14.91 |
| **4** | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.1980 | 0.10430 | 0.1809 | 0.05883 | ... | 22.54 |

5 rows × 30 columns

In [53]:
```
y = cancerData.target
```

In [54]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.1,stratify=y)
```

In [55]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [56]:
```
X_train = X_train.reshape(512,30,1)
X_test = X_test.reshape(57,30,1)
```

In [57]:
```
model = Sequential()
model.add(Conv1D(filters=16,kernel_size=2,activation='relu',input_shape=(30,1)))
model.add(Dropout(0.2))
model.add(Conv1D(32,2,activation='relu'))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(32,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1,activation='sigmoid'))
```

In [58]:
```
model.compile(optimizer=Adam(learning_rate=0.0001),loss='binary_crossentropy',metrics=['accuracy'])
```

```
history = model.fit(X_train,y_train,epochs=35,verbose=1,validation_data=(X_test,y_test))
```

```
16/16 [==============================] - 0s 13ms/step - loss: 0.0861 - accuracy: 0.9727 -
val_loss: 0.0582 - val_accuracy: 0.9825
```

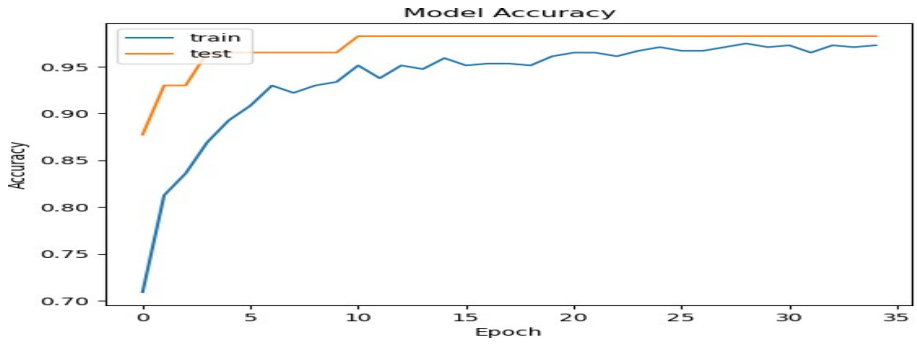
In [68]:
```
load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

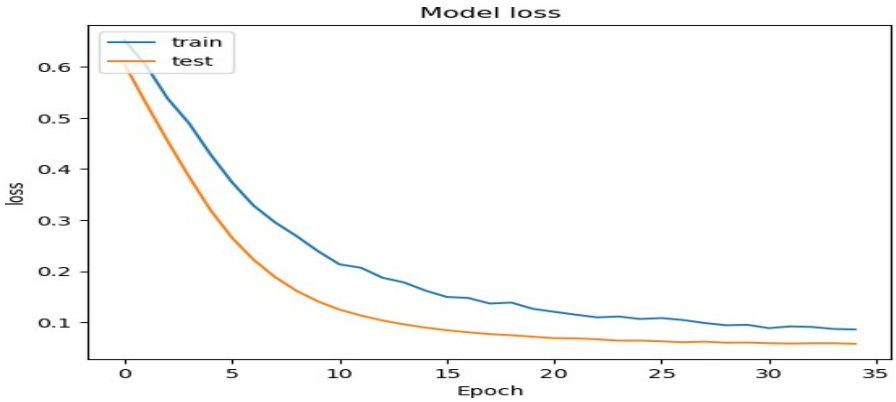
In [69]:
```
tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6006 (pid 5344), started 25 days, 1:21:32 ago. (Use '!kill
5344' to kill it.
```

In [70]:
```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
```




In
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
[p7l1t].:xlabel('Epoch')
plt.legend(['train','test'],loc = 'upper left')
plt.show()
```



# CNN model using mnist Dataset

In [1 j:
```
import tensorflow as tf
```

In [2j:
```
from keras.models import Sequential
from keras.layers import Flatten,Dense,Dropout,Activation
from keras.optimizers import Adam
```

In [5]:
```
mnist=tf.keras.datasets.mnist
(x_train,y_train),(x_test,y_test)=mnist.load_data()
(x_train,x_test)=(x_train/255.0,x_test/255.0)
```

In :
```
model=tf.keras.models.Sequential([ tf.keras.layers.Flatten(
    input_shape=(28,28)),
    tf.keras.layers.Dense(512,activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10,activation='softmax')
```

In [9 ]:
```
model.compile(optimizer='SGD', loss='sparse  categorical  crossentropy', metrics=['accuracy'])
```

In [10]:
```
tf_callbacks=tf.keras.callbacks.TensorBoard(log_dir='logs/fit',histogram_freq=1)
```

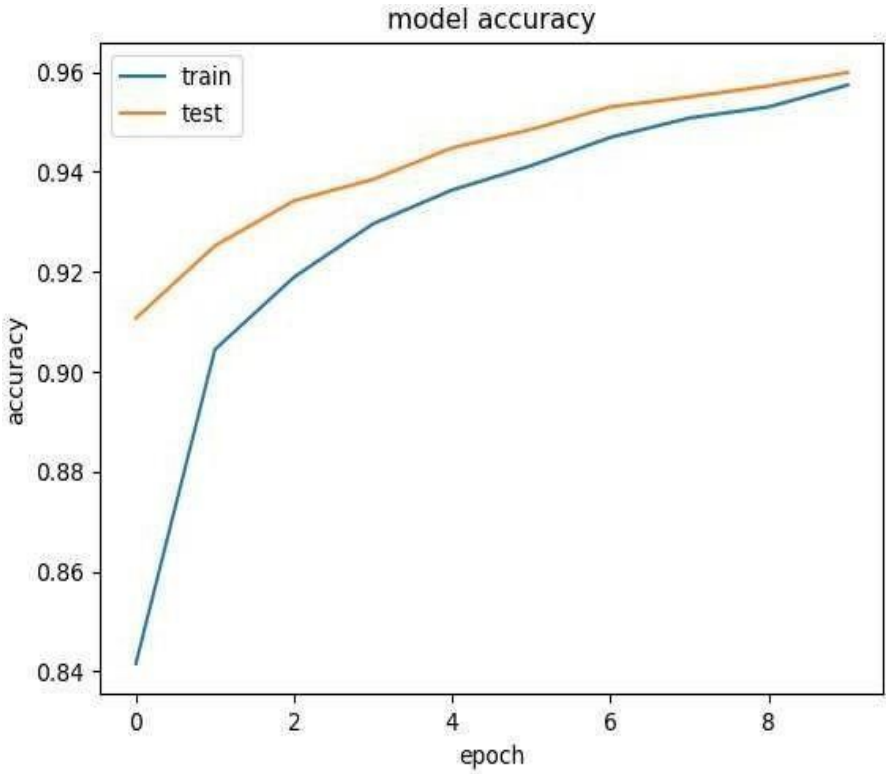
In [11 :
```
history=model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=10,callbacks=tf_callbacks)
```

```
Epoch 1/10
1875/1875 [==============================]  - 26s 13ms/step - loss: 0.6246 - accuracy: 0.8415 - val
 loss: 0.3379 - val_accuracy: 0.9107
Epoch 2/10
1875/1875 [==============================]  - 24s 13ms/step - loss: 0.3397 - accuracy: 0.9044 - val
 loss: 0.2761 - val_accuracy: 0.9252
Epoch 3/10
1875/1875 [==============================]  - 24s 13ms/step - loss: 0.2874 - accuracy: 0.9189 - val
 loss: 0.2399 - val_accuracy: 0.9342
Epoch 4/10
1875/1875 [==============================]  - 24s 13ms/step - loss: 0.2519 - accuracy: 0.9296 - val
 loss: 0.2167 - val_accuracy: 0.9385
Epoch 5/10
1875/1875 [==============================]  - 24s 13ms/step - loss: 0.2272 - accuracy: 0.9364 - val
 loss: 0.1962 - val_accuracy: 0.9448
Epoch 6/10
1875/1875 [==============================]  - 25s 13ms/step - loss: 0.2075 - accuracy: 0.9413 - val
 loss: 0.1810 - val_accuracy: 0.9485
Epoch 7/10
1875/1875 [==============================]  - 25s 13ms/step - loss: 0.1897 - accuracy: 0.9469 - val
 loss: 0.1683 - val_accuracy: 0.9530
Epoch 8/10
1875/1875 [==============================]  - 24s 13ms/step - loss: 0.1778 - accuracy: 0.9508 - val
 loss: 0.1582 - val_accuracy: 0.9550
Epoch 9/10
1875/1875 [==============================]    25s 13ms/step   loss: 0.1657   accuracy: 0.9530   val
 loss: 0.1492 - val_accuracy: 0.9572
Epoch 10/10
1875/1875 [==============================]  - 24s 13ms/step - loss: 0.1552 - accuracy: 0.9574 - val
 loss: 0.1406 - val_accuracy: 0.9599
```

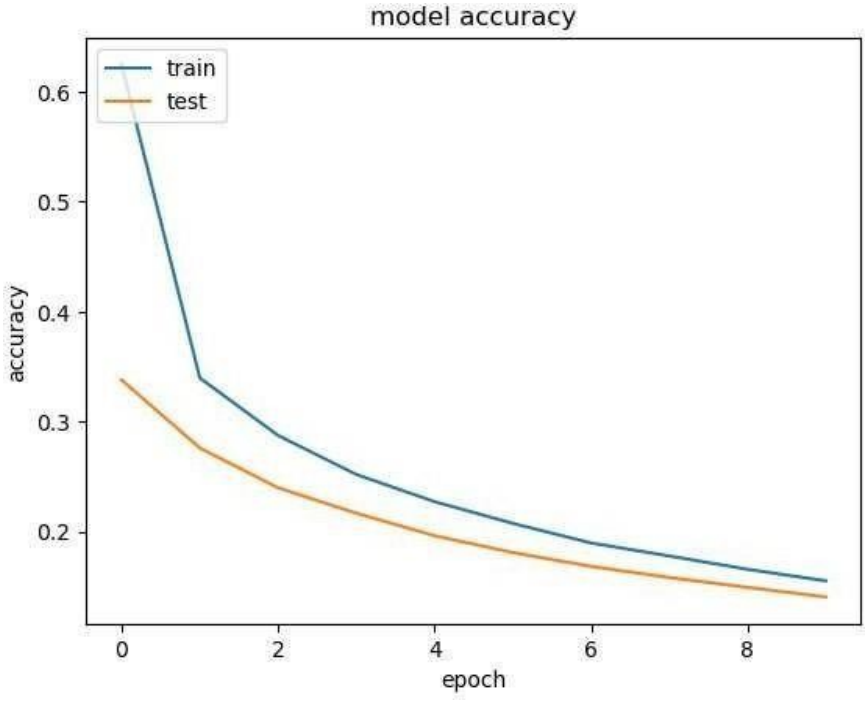
In[12]
```
%reload_ext tensorboard
```

```
tensorboard --logdir logs/fit
```

In [14]:
```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'],loc='upper left')
plt.show()
```




```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'],loc='upper left')
plt.show()
```



# CNN model using IRIS Dataset

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

In [2]:
```
df=pd.read_csv("C:\\Users\\cheth\\Downloads\\IRIS.csv")
df
```

Out[2]:

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| | | | | | |
| **145** | 6.7 | 3.0 | 5.2 | 2.3 | Iris-virginica |
| **146** | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| **147** | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| **148** | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| **149** | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

150 rows x 5 columns

In [3]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['species']=le.fit_transform(df['species'])
df
```

Out[3]:

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| | 4.9 | 3.0 | 1.4 | 0.2 | 0 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | 0 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | 0 |
| | | | | | |
| **145** | 6.7 | 3.0 | 5.2 | 2.3 | 2 |
| **146** | 6.3 | 2.5 | 5.0 | 1.9 | 2 |
| **147** | 6.5 | 3.0 | 5.2 | 2.0 | 2 |
| **148** | 6.2 | 3.4 | 5.4 | 2.3 | 2 |
| **149** | 5.9 | 3.0 | 5.1 | 1.8 | 2 |

150 rows x 5 columns

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
sepal_length  0
sepal_width   0
petal_length  0
petal_width   0
species       0
dtype: int64
```

In [5]:
```
x=df.drop('species',axis=1)
```

In [6]:
```
y_=df['species']
```

In [7]:
```
import numpy as np

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder

from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
```

In [8]:
```
yl=np.array(y_)
```

In [9]:
```
y2 = yl.reshape(-1, 1)
```

In [10]:
```
encoder = OneHotEncoder(sparse=False)
y = encoder.fit_transform(y2)
```

```
C:\Users\cheth\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:868: FutureWarning:
'sparse· was renamed to 'sparse_output' in version 1.2 and will be removed in 1.4. 'sparse_output'
is ignored unless you leave 'sparse' to its default value.
  warnings.warn(
```

In [11]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
print('xtrain:',x_train.shape)
print('xtest:',x_test.shape)
print('ytain:',y_train.shape)
print('ytest:',y_test.shape)
```

```
xtrain: (120, 4)
xtest: (30, 4)
ytain: (120, 3)
ytest: (30, 3)
```

In [12]:
```
model = Sequential()
model.add(Dense(10, input_shape=(4,), activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

In [13]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

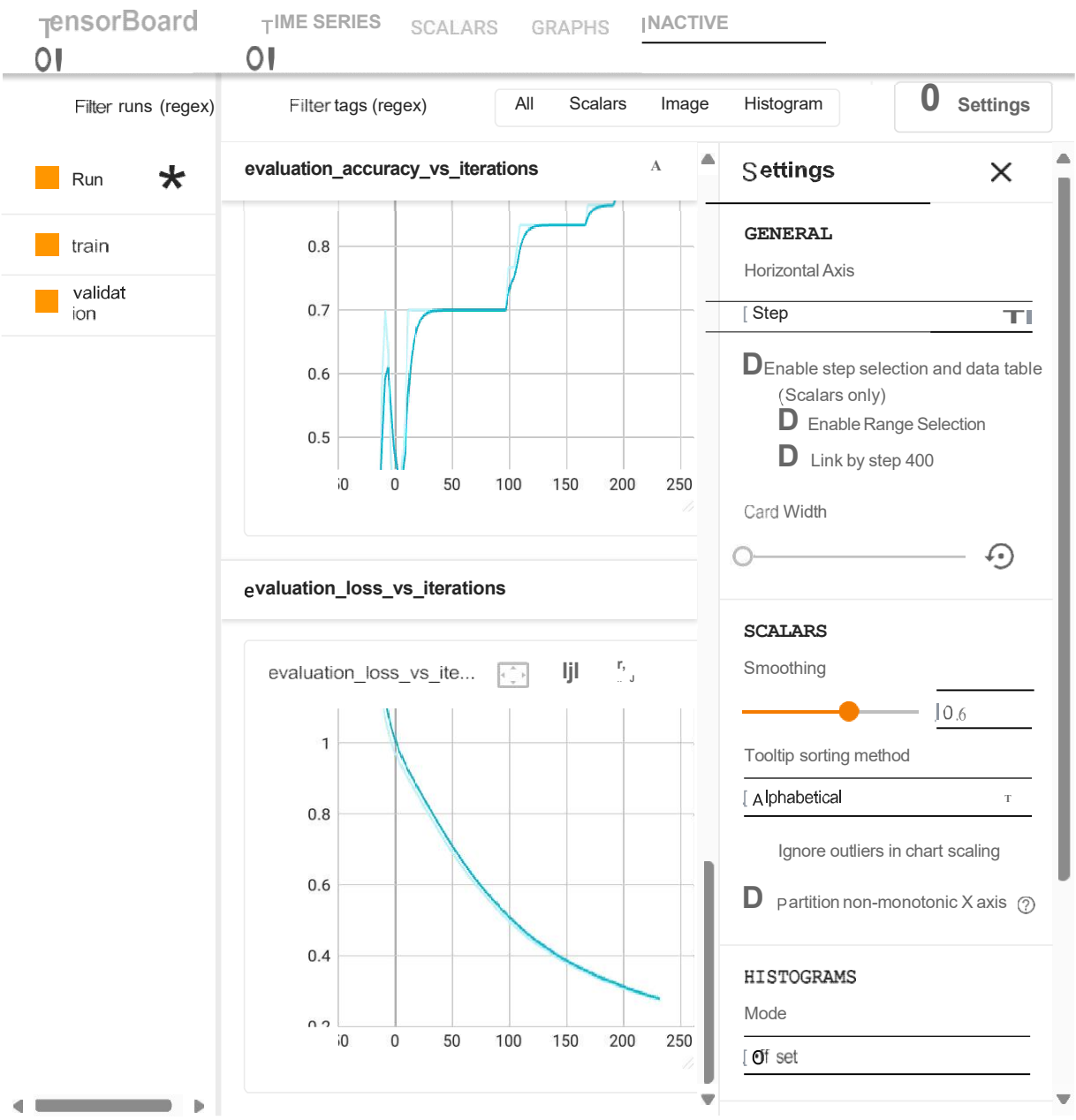
In [14]:
```
import tensorflow as tf
```

In [15]:
```
tf_callbacks=tf.keras.callbacks.TensorBoard(log_dir='logs/fit',histogram_freq=1)
```

In [18]:
```
history=model.fit(x_train,y_train,validation_
data=(x_test,y_test),epochs=50,callbacks=tf_c
allbacks)Epoch 1/50
4/4 [==============================]  - 0s 62ms/step  - loss: 0.4960 - accuracy: 0.7583 - val_loss:
0.4700 - val_accuracy: 0.8333
Epoch 2/50
4/4 [==============================]  - 0s 35ms/step  - loss: 0.4889 - accuracy: 0.7833 - val_loss:
0.4631 - val_accuracy: 0.8333
Epoch 3/50
4/4 [==============================]  - 0s 67ms/step - loss: 0.2757 - accuracy: 0.9833 - val_loss:
0.2797 - val_accuracy: 0.9000
Epoch 49/50
4/4 [==============================]  - 0s 42ms/step - loss: 0.2731 - accuracy: 0.9750 - val_loss:
0.2798 - val_accuracy: 0.9000
Epoch 50/50
4/4 [==============================]  - 0s 40ms/step - loss: 0.2693 - accuracy: 0.9667 - val_loss:
0.2759 - val_accuracy: 0.9000
```

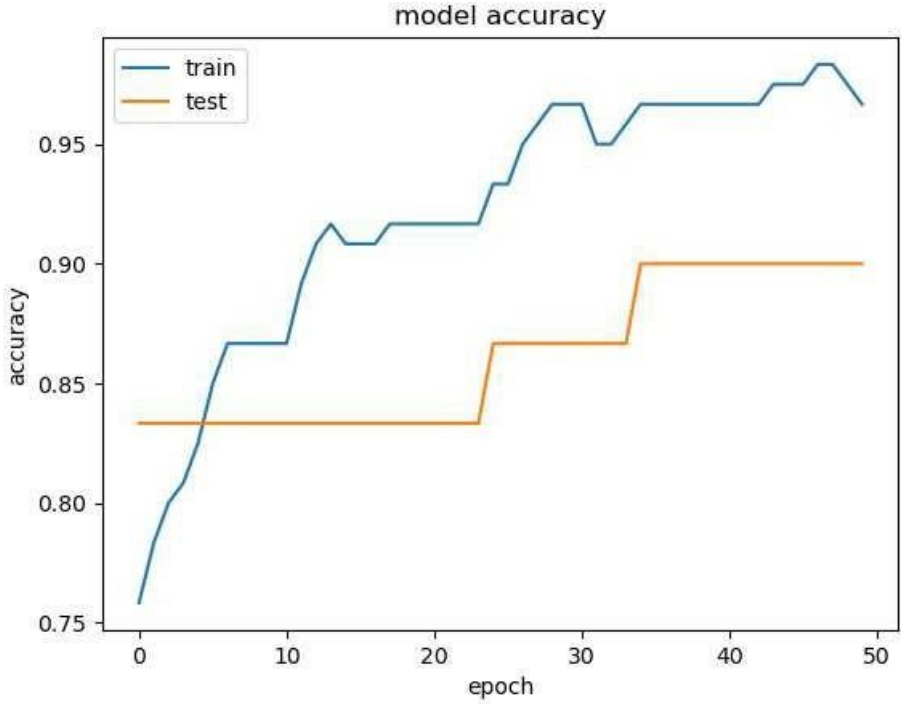
In [19]:
```
%reload_ext tensorboard
```

In [20]:
```
tensorboard --logdir logs/fit
```



In [21]:

```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
p1t . plot (history. history [ 'val  accuracy ' l }
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'],loc='upper left')
plt.show()
```




In [22]:

```
plt.plot(history.history['loss'])
plt.pit(history.history'val loss']
plt.title('modelloss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['trdin','test'],loc='upper  left')
plt.show()
```