DATA SCIENCE JOURNAL

Name : Kamal vasa Class : TYBSc CS

RollNo : TCS2324087

SIES College of Arts, Science & Commerce

RISE WITH EDUCATION

NAAC REACCREDITED - 'A' GRADE

ISO 9001 : 2008

S.I.E.S College of Arts, Science and Commerce(Autonomous)

Sion(W), Mumbai – 400 022.

CERTIFICATE

This is to certify that Mr. Kamal vasa. Roll No. TCS2324087 has successfully completed the necessary course of experiments in the subject of Data Science during the academic year 2023 – 2024 complying with the requirements of University of Mumbai, for the course of TYBSc Computer Science [Semester-VI].

Prof. In-Charge

Prof. MAYA NAIR

Examination date:

Examiner's Signature & Date:

Head of the Department

Prof. Manoj Singh

College Seal

INDEX

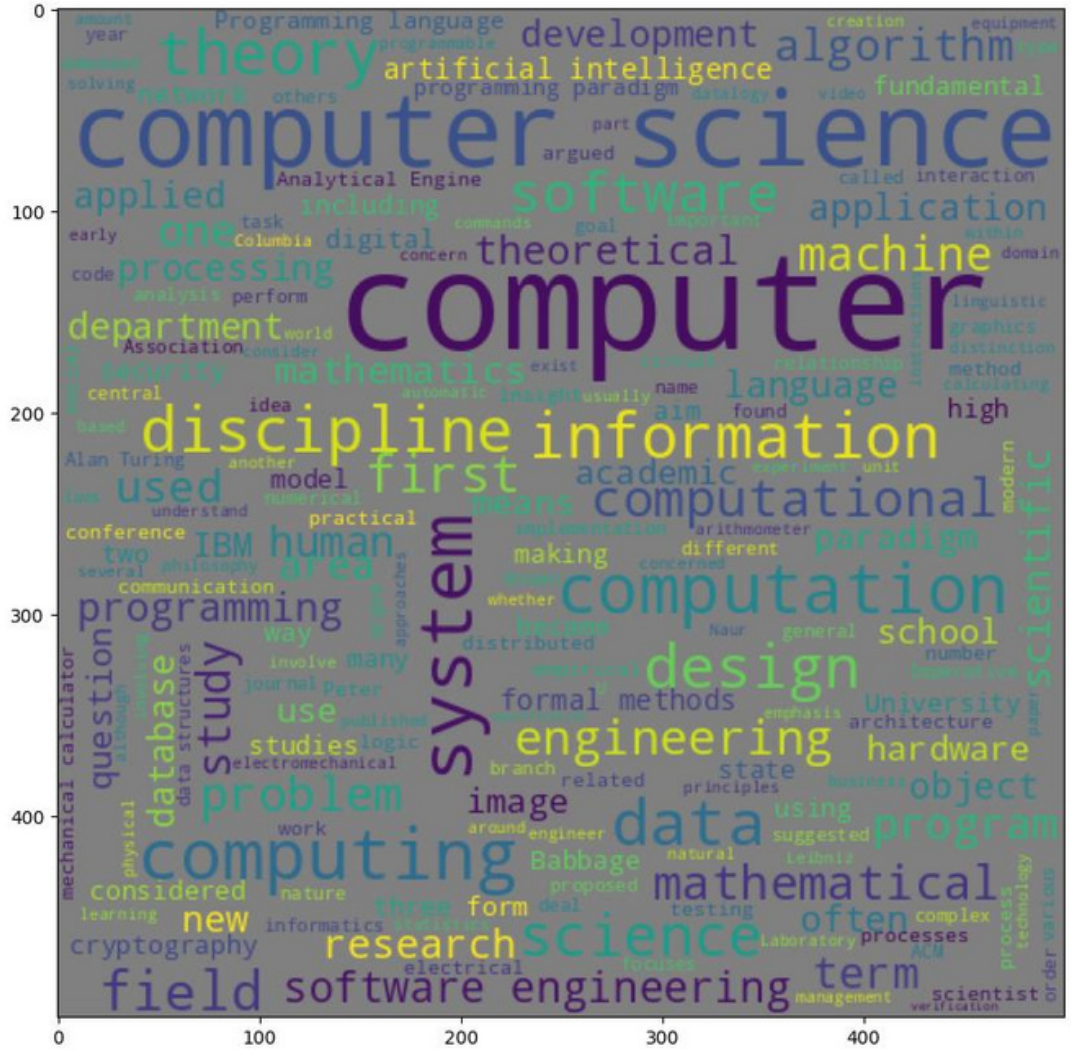
Practical 1:

Aim: Write a python program to plot word cloud for a Wikipedia page of any topic.

Code:

```
from wordcloud import WordCloud,STOPWORDS
import matplotlib.pyplot as plt
import wikipedia as wp

result=wp.page("Computer Science")
final_result=result.content

def plot_wordcloud(wc):
    plt.axis("off")
    plt.figure(figsize=(10,10)) #width,height
    plt.imshow(wc)
    plt.show()
wc=WordCloud(width=500,height=500,background_color="grey",random_state=10,stopwords=STOPWORDS).generate(final_result)
wc.to_file("cs.png") #stored in file ie cwd
plot_wordcloud(wc)
```

Output:



Practical 2:

Web scraping Web scrapingis the process of collecting and parsing raw data from the Web.

Aim: Write a python program to perform Web Scrapping

01.Html scrapping- use Beautiful

Soup Code:

```
#html scrapping
import pandas as pd
from bs4 import BeautifulSoup #scrapping
from urllib.request import urlopen
url="https://en.wikipedia.org/wiki/List_of_Asian_countries_by_area"
page=urlopen(url)
html_page=page.read().decode("utf-8")
#print(html_page)
soup=BeautifulSoup(html_page,"html.parser")
table=soup.find("table") #find tag
#print(table)
SrNo=[]
Country=[]
Area=[]
rows=table.find("tbody").find_all("tr")
for row in rows:
    cells=row.find_all("td")
    if(cells):
        SrNo.append(cells[0].get_text().strip("\n"))#strip extra characters
        Country.append(cells[1].get_text().strip("\xa0").strip("\n").strip("\[2]*"))#strip extra characters
        Area.append(cells[3].get_text().strip("\n").replace(",",""))#strip extra characters
countries_df=pd.DataFrame()
countries_df["SrNo"]=SrNo
countries_df["Country"]=Country
countries_df["Area"]=Area
print(countries_df.head(10))
```

Output:

	SrNo	Country	Area
0	1	Russia	13083100 (5051400)
1	2	China	9596961 (3705407)
2	3	India	3287263 (1269219)
3	4	Kazakhstan	2600000 (1000000)
4	5	Saudi Arabia	2149690 (830000)
5	6	Iran	1648195 (636372)
6	7	Mongolia	1564110 (603910)
7	8	Indonesia	1488509 (574717)
8	9	Pakistan	881913 (340509)
9	10	Turkey	759805 (293362)

02.json scrapping

Code:

```
import pandas as pd
import urllib,json
url="https://jsonplaceholder.typicode.com/users"
response=urllib.request.urlopen(url)
data=json.loads(response.read()) #json -> python data structure
#print(data)
id1=[]
username=[]
email=[]
for item in data:
    if "id" in item.keys(): #check if id is present in dictionary
        id1.append(item["id"])
    else:
        id1.append("NA")
    if "username" in item.keys(): #check if id is present in dictionary
        username.append(item["username"])
    else:
        username.append("NA")
    if "email" in item.keys(): #check if id is present in dictionary
        email.append(item["email"])
    else:
        email.append("NA")
print(id1)
print(username)
print(email)
user_accounts=pd.DataFrame()
user_accounts["USER ID"]=id1
user_accounts["USERNAME"]=username
user_accounts["EMAIL"]=email
user_accounts.head(10)
```

Output:

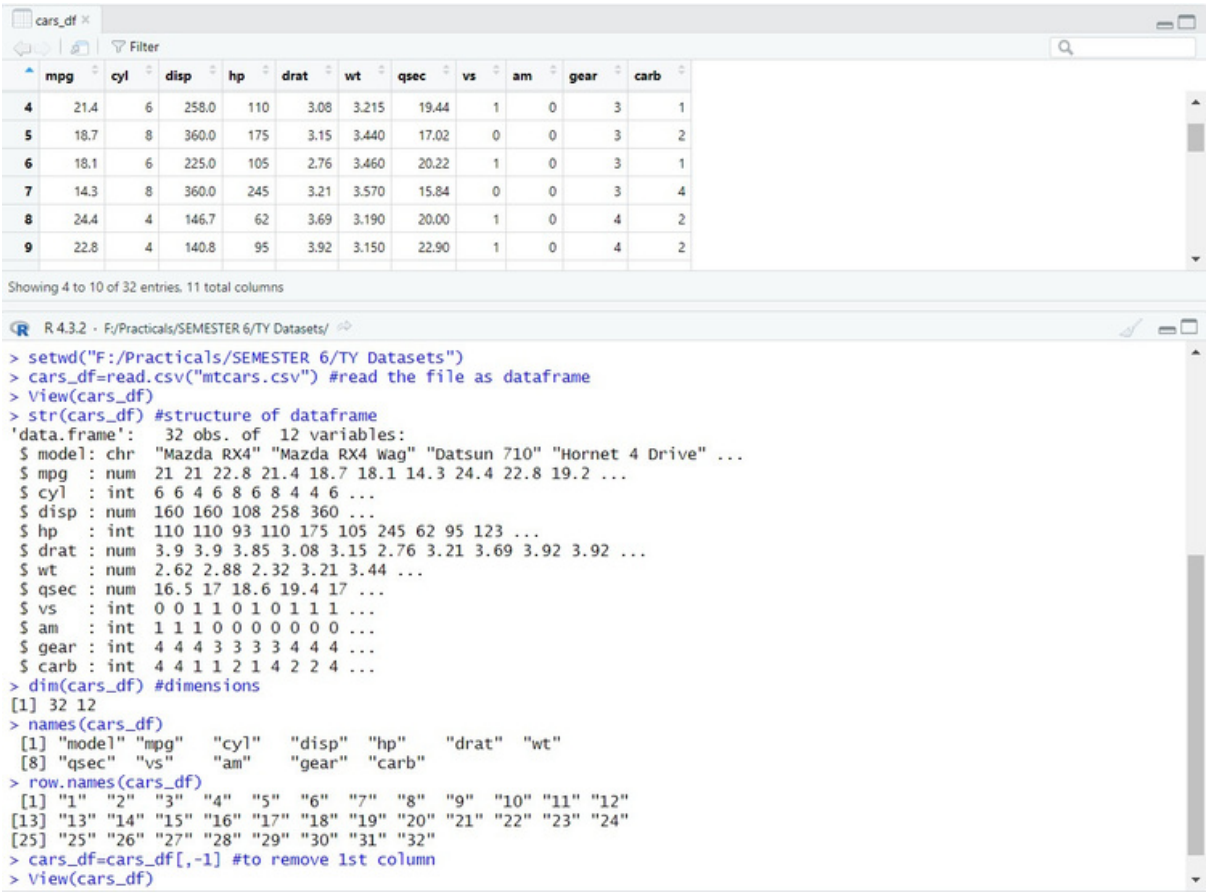
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
['Bret', 'Antonette', 'Samantha', 'Karianne', 'Kamren', 'Leopoldo_Corkery', 'Elwyn.Skiles', 'Maxime_Nienow', 'Delphine', 'Moria
h.Stanton']
['Sincere@april.biz', 'Shanna@melissa.tv', 'Nathan@yesenia.net', 'Julianne.OConner@kory.org', 'Lucio_Hettinger@annie.ca', 'Karl
ey_Dach@jasper.info', 'Telly.Hoeger@billy.biz', 'Sherwood@rosamond.me', 'Chaim_McDermott@dana.io', 'Rey.Padberg@karina.biz']
```

Out[5]:

	USER ID	USERNAME	EMAIL
0	1	Bret	Sincere@april.biz
1	2	Antonette	Shanna@melissa.tv
2	3	Samantha	Nathan@yesenia.net
3	4	Karianne	Julianne.OConner@kory.org
4	5	Kamren	Lucio_Hettinger@annie.ca
5	6	Leopoldo_Corkery	Karley_Dach@jasper.info
6	7	Elwyn.Skiles	Telly.Hoeger@billy.biz
7	8	Maxime_Nienow	Sherwood@rosamond.me
8	9	Delphine	Chaim_McDermott@dana.io
9	10	Moriah.Stanton	Rey.Padberg@karina.biz

Practical 3:

Aim: Exploratory Data Analysis of mtcars.csv Dataset in R (Use functions of dplyr like select, filter, mutate , rename, arrange, group by, summarize and data visualizations) segreatum



cars_df

Filter

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
4	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
5	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
6	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
7	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
8	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
9	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2

Showing 4 to 10 of 32 entries. 11 total columns

R 4.3.2 · F:/Practicals/SEMESTER 6/TY Datasets/

```
> setwd("F:/Practicals/SEMESTER 6/TY Datasets")
> cars_df=read.csv("mtcars.csv") #read the file as dataframe
> View(cars_df)
> str(cars_df) #structure of dataframe
'data.frame':   32 obs. of  12 variables:
 $ model: chr  "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" ...
 $ mpg  : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl  : int  6 6 4 6 8 6 8 4 4 6 ...
 $ disp : num  160 160 108 258 360 ...
 $ hp   : int  110 110 93 110 175 105 245 62 95 123 ...
 $ drat : num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt   : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec : num  16.5 17 18.6 19.4 17 ...
 $ vs   : int  0 0 1 1 0 1 0 1 1 1 ...
 $ am   : int  1 1 1 0 0 0 0 0 0 0 ...
 $ gear : int  4 4 4 3 3 3 3 4 4 4 ...
 $ carb : int  4 4 1 1 2 1 4 2 2 4 ...
> dim(cars_df) #dimensions
[1] 32 12
> names(cars_df)
 [1] "model" "mpg"   "cyl"   "disp"  "hp"    "drat"  "wt"
 [8] "qsec"  "vs"    "am"    "gear"  "carb"
> row.names(cars_df)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12"
[13] "13" "14" "15" "16" "17" "18" "19" "20" "21" "22" "23" "24"
[25] "25" "26" "27" "28" "29" "30" "31" "32"
> cars_df=cars_df[,-1] #to remove 1st column
> View(cars_df)
```

Select function:

```
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> #select: used for extracting specific columns
> df1=select(cars_df,mpg:hp)
> df1=cars_df%>%select(mpg:hp) #  METHOD 2
> View(df1)
```

cars_df × df1 ×

Filter

	mpg	cyl	disp	hp
1	21.0	6	160.0	110
2	21.0	6	160.0	110
3	22.8	4	108.0	93
4	21.4	6	258.0	110
5	18.7	8	360.0	175
6	18.1	6	225.0	105
7	14.3	8	360.0	245
8	24.4	4	146.7	62
9	22.8	4	140.8	95
10	19.2	6	167.6	123

Filter:

```
> #filter: used to extract specific rows
> #exracting records where gear=4 and columns to be displayed are mpg,
disp, wt, gear
> df1=cars_df%>%filter(gear==4)%>%select(c(mpg,disp,wt,gear))
> View(df1)
```

cars_df × df1 ×

Filter

	mpg	disp	wt	gear
1	21.0	160.0	2.620	4
2	21.0	160.0	2.875	4
3	22.8	108.0	2.320	4
4	24.4	146.7	3.190	4
5	22.8	140.8	3.150	4
6	19.2	167.6	3.440	4
7	17.8	167.6	3.440	4
8	32.4	78.7	2.200	4
9	30.4	75.7	1.615	4
10	33.9	71.1	1.835	4
11	27.3	79.0	1.935	4
12	21.4	121.0	2.780	4

```
> #Extract records where cy=4 or mpg>20 and columns required are mpg, cyl
> df1=cars_df %>% filter(cyl==4 | mpg>20) %>% select(c(mpg,cyl))
> View(df1)
>
```

cars_df × df1 ×

Filter

	mpg	cyl
1	21.0	6
2	21.0	6
3	22.8	4
4	21.4	6
5	24.4	4
6	22.8	4
7	32.4	4
8	30.4	4
9	33.9	4
10	21.5	4
11	27.3	4
12	26.0	4
13	30.4	4
14	21.4	4

```
> #Extract the records where the mpg<20 and carb=3
> df1=cars_df%>% filter(mpg<20 & carb==3) %>% select(c(mpg,carb))
> View(df1)
>
```

cars_df × df1 ×

Filter

	mpg	carb
1	16.4	3
2	17.3	3
3	15.2	3

Arrange:

```
> #Arrange : used for sorting the data as per secific columns
> df1=cars_df%>%arrange(cyl,desc(mpg),desc(wt)) #sorting cyl first and when multiple ent
ries are seen , mpg is sorted in descending order
> View(df1)
>
```

cars_df × df1 ×

Filter

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
1	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
2	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
3	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
4	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
5	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
6	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
7	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
8	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
9	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
10	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
11	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2
12	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
13	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
14	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
15	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6

Remane

```
> #Rename function: to change the name of the column
> df1=cars_df%>%rename(MilesperGallon=mpg, cylinders=cyl, Displacement=disp)
> View(df1)
>
```

cars_df × df1 ×

Filter

	MilesperGallon	cylinders	Displacement	hp	drat	wt	qsec	vs	am	gear	carb
1	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
2	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
3	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
4	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
5	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
6	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
7	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
8	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
9	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
10	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
11	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
12	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
13	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
14	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3

Mutate:

```
> #Mutate: used for creating new columns on the basis of existing columns
> df1=cars_df%>%mutate(power=hp*wt)
> View(df1)
>
>
```

cars_df × df1 ×

Filter

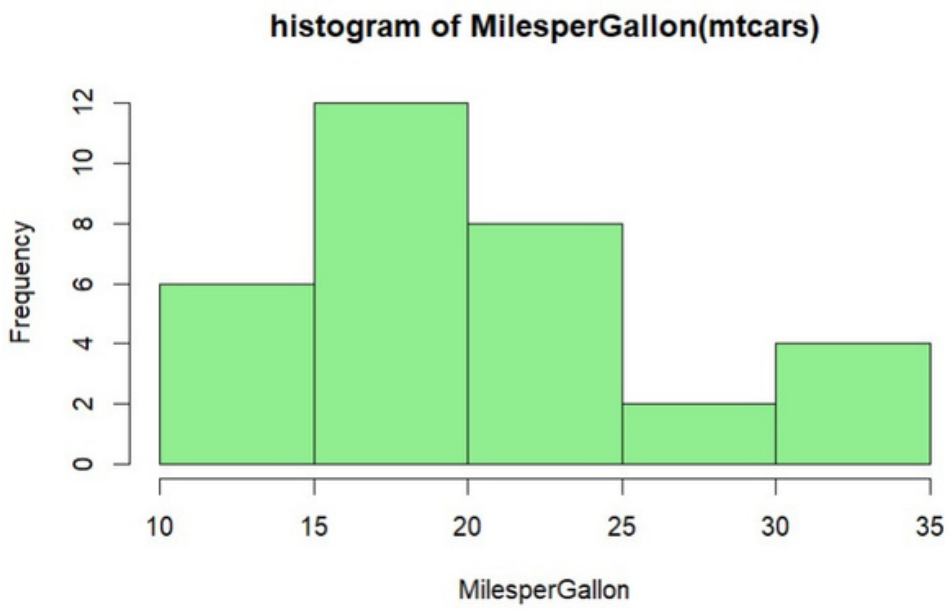
	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	power
1	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	288.200
2	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	316.250
3	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	215.760
4	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	353.650
5	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2	602.000
6	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1	363.300
7	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4	874.650
8	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2	197.780
9	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2	299.250
10	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4	423.120
11	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4	423.120
12	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3	732.600
13	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3	671.400
14	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3	680.400
15	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4	1076.250

Group by

```
> #group_by and summarise- segregating data as per categorical variables and summarizing
> cars_df$gear=as.factor(cars_df$gear)
> str(df1)
'data.frame':   32 obs. of  12 variables:
 $ mpg  : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl  : int  6 6 4 6 8 6 8 4 4 6 ...
 $ disp : num  160 160 108 258 360 ...
 $ hp   : int  110 110 93 110 175 105 245 62 95 123 ...
 $ drat : num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt   : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec : num  16.5 17 18.6 19.4 17 ...
 $ vs   : int  0 0 1 1 0 1 0 1 1 1 ...
 $ am   : int  1 1 1 0 0 0 0 0 0 0 ...
 $ gear : int  4 4 4 3 3 3 3 4 4 4 ...
 $ carb : int  4 4 1 1 2 1 4 2 2 4 ...
 $ power: num  288 316 216 354 602 ...
> summary_df=df1%>%group_by(df1$gear)%>%summarise(no=n(),mean_mpg=mean(mpg))
> summary_df
# A tibble: 3 × 3
  `df1$gear`    no mean_mpg
       <int> <int>    <dbl>
1          3    15     16.1
2          4    12     24.5
3          5     5     21.4
> |
```

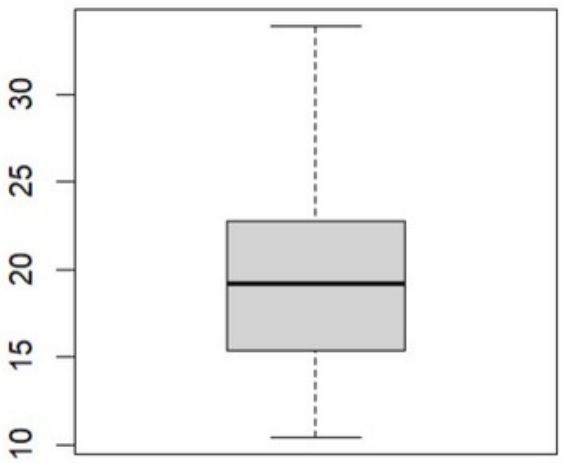
Data Visualization

```
> #DATA VISUALIZATIONS
> #histogram
> hist(df1$mpg)
> hist(df1$mpg,main="histogram of MilesperGallon(mtcars)",col="lightgreen", xlab="MilesperGallon")
>
```



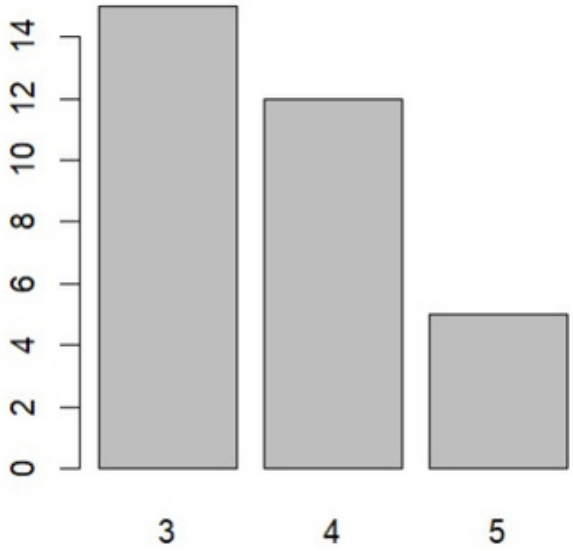
Boxplot:

```
> #box plot:- visualize the summary function
> summary(df1$mpg)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.40   15.43   19.20   20.09   22.80   33.90
> boxplot(df1$mpg)
>
```



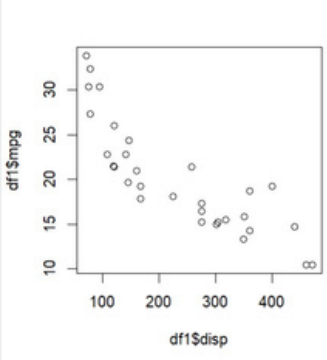
Bar plot:

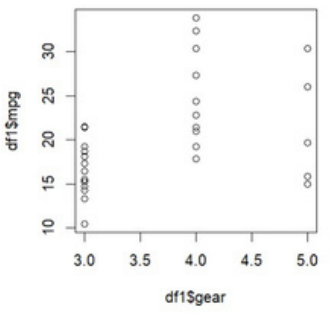
```
> #barplot:- categorical variable representation table
> table(df1$gear)

 3  4  5
15 12  5
> barplot(table(df1$gear))
>
```

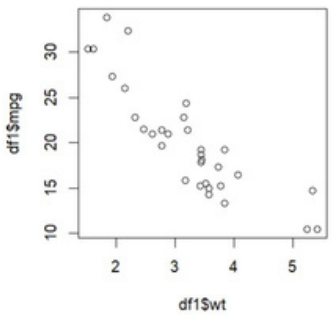


Scatter plot

```
> #scatter plot-plots relationshipbetween two variables
> plot(df1$mpg~df1$disp)  #negative trend
> plot(df1$mpg~df1$gear)  #no relation
> plot(df1$mpg~df1$wt)  #negative trend
>
```







Practical 4:

Aim: Exploratory data analysis in Python using Titanic Dataset –
It is one of the most popular datasets used for understanding machine learning basics. It contains information of all the passengers aboard the RMS Titanic, which unfortunately was shipwrecked. This dataset can be used to predict whether a given passenger survived or not.

Code:

```
!pip install Seaborn
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
#preprocessing
%matplotlib inline

titanic=pd.read_csv("train.csv")
titanic.head()

titanic.info() #structure
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
titanic.describe() #statistical info about column
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

```
titanic.isnull().sum()
```

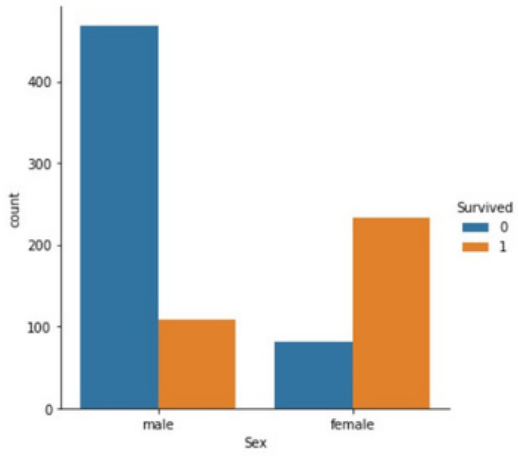
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
titanic_cleaned=titanic.drop(['PassengerId','Name','Ticket','Cabin'],axis=1)
titanic_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 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 
dtypes: float64(2), int64(4), object(2)
memory usage: 55.8+ KB
```

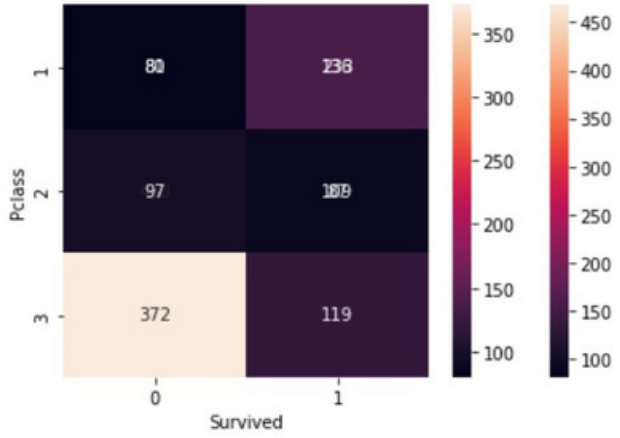
```
#plotting count of passengers who survived on basis of sex using seaborn
sns.catplot(x='Sex',hue='Survived',kind='count',data=titanic)#y->kind

titanic_cleaned.groupby(['Sex','Survived'])['Survived'].count()
```

```
Sex     Survived
female  0            81
        1           233
male    0           468
        1           109
Name: Survived, dtype: int64
```



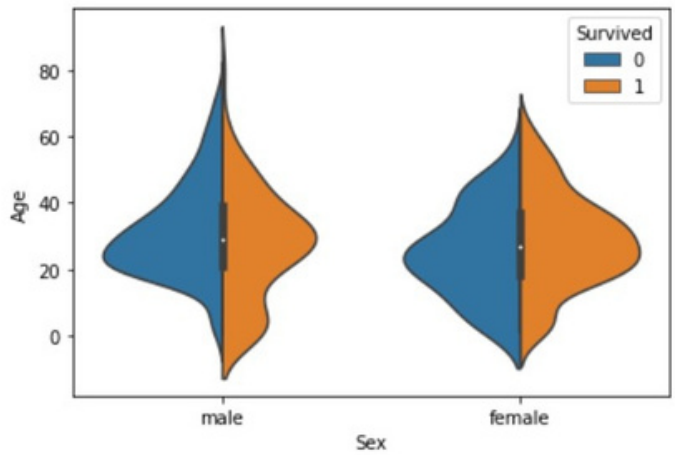
```
grp1=titanic_cleaned.groupby(['Sex','Survived'])
gender_survived=grp1.size().unstack()
sns.heatmap(gender_survived,annot=True,fmt="d") # fmt-format d-> integer annot-annotation

grp1=titanic_cleaned.groupby(['Pclass','Survived'])
gender_survived=grp1.size().unstack()
sns.heatmap(gender_survived,annot=True,fmt="d") # fmt-format d-> integer annot-annotation
```

```
<AxesSubplot:xlabel='Survived', ylabel='Pclass'>
```




```
#violin-> for visualisation the distribution of a numerical variable across various classes
sns.violinplot(x="Sex",y="Age",hue="Survived",data=titanic_cleaned, split=True)

#imputing to handle age problem
print("Oldest person on board: ",titanic_cleaned['Age'].max())
print("Oldest person on board: ",titanic_cleaned['Age'].min())
print("Oldest person on board: ",titanic_cleaned['Age'].mean())
```

```
Oldest person on board:  80.0
Oldest person on board:  0.42
Oldest person on board:  29.69911764705882
```



``

```
def impute(cols):
    Age=cols[0]
    Pclass=cols[1]
    if pd.isnull(Age):
        if Pclass==1:
            return 34
        elif Pclass==2:
            return 29
        else:
            return 24
    else:
        return Age
titanic_cleaned['Age']=titanic_cleaned[["Age","Pclass"]].apply(impute,axis=1) #apply userdefined fnc on given set of data
titanic_cleaned.isnull().sum()
```

```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    2
dtype: int64
```

```
#correlation
titanic_cleaned.corr(method='pearson')
```

	Survived	Pclass	Age	SibSp	Parch	Fare
Survived	1.000000	-0.338481	-0.048737	-0.035322	0.081629	0.257307
Pclass	-0.338481	1.000000	-0.398027	0.083081	0.018443	-0.549500
Age	-0.048737	-0.398027	1.000000	-0.242196	-0.168427	0.119877
SibSp	-0.035322	0.083081	-0.242196	1.000000	0.414838	0.159651
Parch	0.081629	0.018443	-0.168427	0.414838	1.000000	0.216225
Fare	0.257307	-0.549500	0.119877	0.159651	0.216225	1.000000

Practical 5:

Aim: Exploratory data analysis in Python using Titanic Dataset

1) Write a python program to build a regression model that could predict the salary of an employee from the given experience and visualize univariate linear regression on it.

Code:

```
from sklearn import datasets
import numpy as np
x,y,coef=datasets.make_regression(n_samples=100,n_features=1,n_informative=1,noise=10,coef=True,random_state=0)
#x->experience y->salary
x=np.interp(x,(x.min(),x.max()),(0,20))  #interpolation
print(len(x))
print(x)
```

```
100
[[ 9.09621765]
 [14.63742853]
 [12.25580785]
 [ 7.21515957]
 [ 6.90562848]
 [12.42799856]
 [ 6.53450315]
 [12.36358975]
```

```
y=np.interp(y,(y.min(),y.max()),(20000,150000))
print(len(y))
print(y)
```

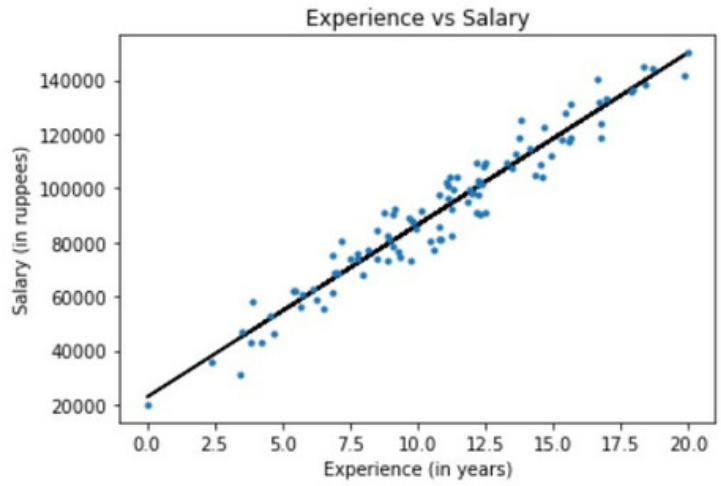
```
100
[ 78311.16075377 103897.6645258   97836.26101499  80550.25638039
  68555.820963   108021.44227128  55778.0199934  101586.97979347
 103966.61856971  76826.00913959  73657.03907056  96439.33831133
  43282.85644907  73119.73495559 109692.0380975  128125.74670244
  87499.26503386 136438.82955292 140414.06203468  75920.22641562
 122765.94046351 138676.79599883  90840.21480164  99453.36502726
 118663.17132396 125247.52951645 144470.99004202  98454.6493064
  92321.3919241  133162.35931048  61723.07434352  77095.35501897
  59042.68149761 109559.00643186  77206.62874325 109743.44545302
 103902.53136675  82585.66146856  81088.97054957  62200.35300958
 111971.74647069 101515.0451792   47090.60230288 141613.36480828
  99370.060872    72953.14343772 131312.34257614  68957.25418311
 135509.14233685  90658.86260334  75147.59074288  46071.12989863
  91128.91843553 105126.42548023 118217.9745179  102729.79197702
  90792.28592931  31126.14409027 131792.50158722  87540.54488213
  68478.3452966   94708.41722992 117182.6660135   36048.82831918
  20000.          52790.64656392  80294.55925299  82720.34224854
 118540.87080567 144911.51213921 150000.          73881.48926853
  81199.99099033 123939.57314323  74226.99120491  43047.41708325
 100812.29148621 108995.7100776  118842.34793498  97468.27208886
 114754.41658701 107396.93022062  85405.03533063  62846.45214219
  91710.86790453  74737.37663911  81016.26833261 112452.52450198
 101930.1138918   84709.84734624  62164.05969477  88875.11018096
  68266.46136864  92525.83267833  86080.90676148  60661.28647136
  90385.8756142   97977.26250181  57982.95420136  56337.82217384]
```

```
import matplotlib.pyplot as plt
plt.plot(x,y,'.',label="Training data")
plt.xlabel("Experience (in years)")
plt.ylabel("Salary (in ruppees)")
plt.title("Experience vs Salary")
```

```
Text(0.5, 1.0, 'Experience vs Salary')
```


```
from sklearn.linear_model import LinearRegression
reg_model=LinearRegression()
reg_model.fit(x,y)
y_pred=reg_model.predict(x) #pass X so that it will predict Y
plt.plot(x,y_pred,color="black")
plt.plot(x,y,'.',label="Training data")
plt.xlabel("Experience (in years)")
plt.ylabel("Salary (in ruppees)")
plt.title("Experience vs Salary")
```

```
Text(0.5, 1.0, 'Experience vs Salary')
```



```
import pandas as pd
data={'Experience':np.round(x.flatten()),'Salary':np.round(y)}
df=pd.DataFrame(data)
df.head(10)
```

	Experience	Salary
0	9.0	78311.0
1	15.0	103898.0
2	12.0	97836.0
3	7.0	80550.0
4	7.0	68556.0
5	12.0	108021.0
6	7.0	55778.0
7	12.0	101587.0
8	11.0	103967.0
9	9.0	76826.0

```
x1=[[13.0]]
y1=reg_model.predict(x1)
print(np.round(y1,2))
```

```
[105534.49]
```

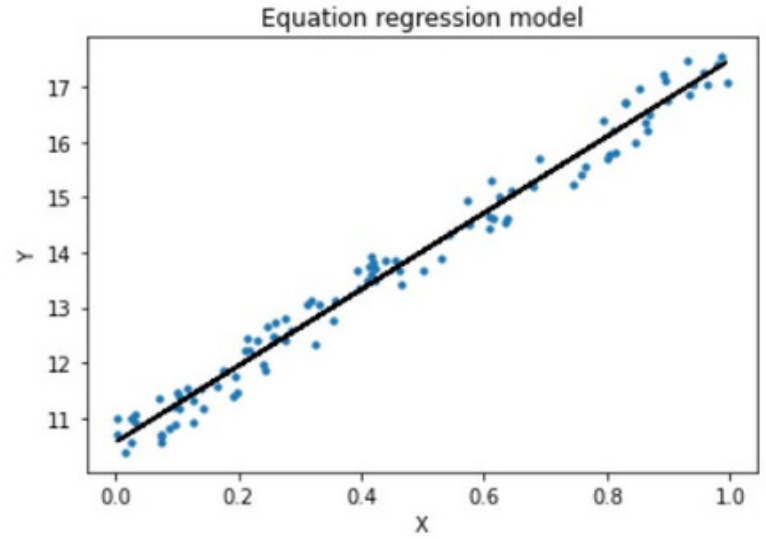
2) Write a python program to simulate linear model
Y=10+7*x+e for random 100 samples and visualize univariate linear regression on it.

Code:

```
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

reg_model1=LinearRegression()
x=np.random.rand(100,1)
yintercept=10
slope=7
error=np.random.rand(100,1)
y=yintercept+ (slope*x) + error
#print(y)

reg_model1.fit(x,y)
y_pred=reg_model1.predict(x) #pass X so that it will predict Y

plt.scatter(x,y,s=10)
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Equation regression model")
plt.plot(x,y_pred,color="black")
```

Output:

```
[<matplotlib.lines.Line2D at 0x2538f818910>]
```



Practical 6:

Aim: Write a python program to implement multiple linear regression on the Dataset Boston.csv

```
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
boston =pd.read_csv("Boston.csv")
boston.head()  #shows 5 lines in python and 6 is R
```

	Unnamed: 0	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

```
boston.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  506 non-null    int64
 1   crim        506 non-null    float64
 2   zn          506 non-null    float64
 3   indus       506 non-null    float64
 4   chas        506 non-null    int64
 5   nox         506 non-null    float64
 6   rm          506 non-null    float64
 7   age         506 non-null    float64
 8   dis         506 non-null    float64
 9   rad         506 non-null    int64
 10  tax         506 non-null    int64
 11  ptratio     506 non-null    float64
 12  black       506 non-null    float64
 13  lstat       506 non-null    float64
 14  medv        506 non-null    float64
dtypes: float64(11), int64(4)
memory usage: 59.4 KB
```

```
boston = boston.drop(columns="Unnamed: 0")
boston.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   crim     506 non-null    float64
 1   zn       506 non-null    float64
 2   indus    506 non-null    float64
 3   chas     506 non-null    int64  
 4   nox      506 non-null    float64
 5   rm       506 non-null    float64
 6   age      506 non-null    float64
 7   dis      506 non-null    float64
 8   rad      506 non-null    int64  
 9   tax      506 non-null    int64  
 10  ptratio  506 non-null    float64
 11  black    506 non-null    float64
 12  lstat    506 non-null    float64
 13  medv     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

```
#index loc(iloc does splitting of existing dataframe)(before comma colon indicating rows i.e all selected)
#After comma colon for columns indicating columns from 0,12
boston_x=pd.DataFrame(boston.iloc[:,:13])
boston_y=pd.DataFrame(boston.iloc[:,-1])
boston_y.head()
```

	medv
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2

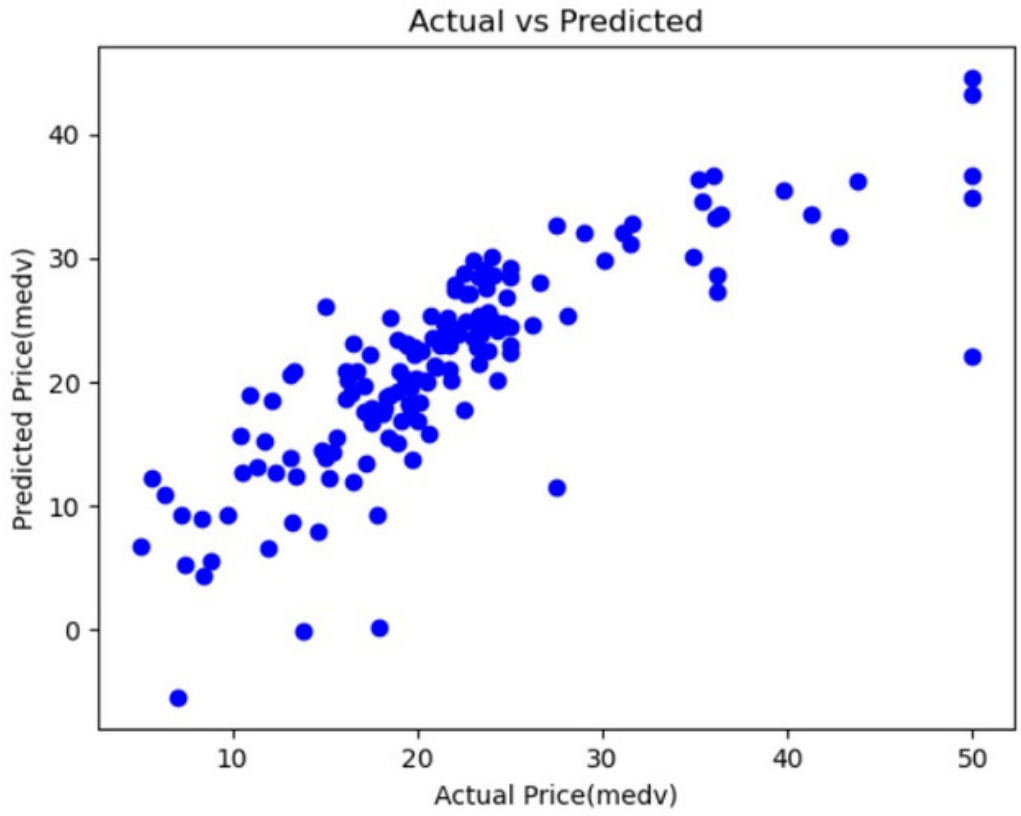
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(boston_x,boston_y,test_size=0.3)
#3rd param is percentage of test data to be included (for 30% we can write as 0.3)
#Thus train data is 70%
print("xtrain Shape",X_train.shape)
print("ytrain Shape",Y_train.shape)
print("xtest Shape",X_test.shape)
print("ytest Shape",Y_test.shape)
```

```
xtrain Shape (354, 13)
ytrain Shape (354, 1)
xtest Shape (152, 13)
ytest Shape (152, 1)
```

```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X_train,Y_train)
Y_pred_linear = regression.predict(X_test)
Y_pred_df = pd.DataFrame(Y_pred_linear,columns=["Predicted"])
Y_pred_df.head()
```

	Predicted
0	22.751459
1	12.651082
2	28.540267
3	29.228829
4	25.230344

```
plt.scatter(Y_test,Y_pred_linear,c="blue")
plt.xlabel("Actual Price(medv)")
plt.ylabel("Predicted Price(medv)")
plt.title("Actual vs Predicted")
plt.show()
```



Practical 7:

Aim: K Nearest Neighbor classification Algorithm
Write a python program to implement KNN algorithm to predict breast cancer using breast cancer wisconsin dataset .

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

bc_df=load_breast_cancer()
x=pd.DataFrame(bc_df.data,columns=bc_df.feature_names)
x.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374

5 rows × 30 columns

```
x=x[["mean area","mean compactness"]]
x.head()
```

	mean area	mean compactness
0	1001.0	0.27760
1	1326.0	0.07864
2	1203.0	0.15990
3	386.1	0.28390
4	1297.0	0.13280

```
y=pd.Categorical.from_codes(bc_df.target,bc_df.target_names)#categorical variable from data
print(y)
```

```
['malignant', 'malignant', 'malignant', 'malignant', 'malignant', ..., 'malignant', 'malignant', 'malignant', 'malignant', 'ben
ign']
Length: 569
Categories (2, object): ['malignant', 'benign']
```

```
#benign=0 -> malignant  #`COnverting two categorical variable into one
#benign=1 -> benign

y=pd.get_dummies(y,drop_first=True) #dropping malignant
print(y)
```

```
     benign
0         0
1         0
2         0
3         0
4         0
..      ...
564       0
565       0
566       0
567       0
568       1

[569 rows x 1 columns]
```

```
X_train,X_test,Y_train,Y_test=train_test_split(x,y,random_state=1) #Skipping ratio will give bydefault 70-30
#recreate the same randomness everytime we run the function
knn=KNeighborsClassifier(n_neighbors=5,metric="euclidean")
knn.fit(X_train,Y_train)
```

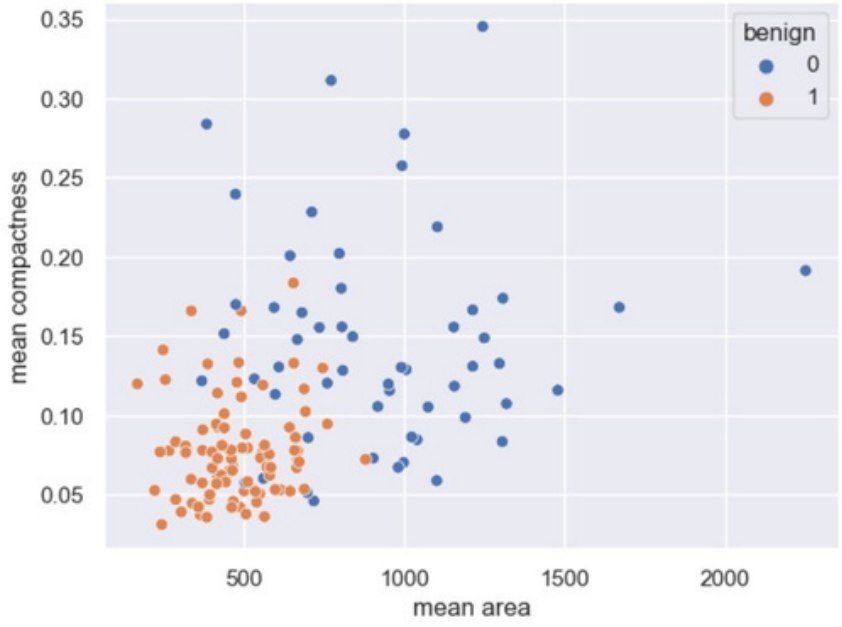
```
C:\ProgramData\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:215: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```

```
KNeighborsClassifier(metric='euclidean')
```

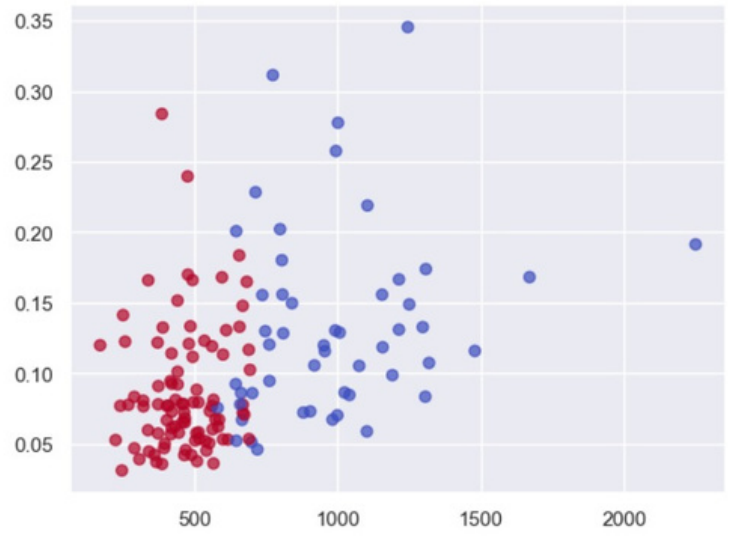
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
sns.set() #scatterplot
sns.scatterplot(x="mean area",y="mean compactness",hue="benign",data=X_test.join(Y_test,how="outer"))
```

```
<Axes: xlabel='mean area', ylabel='mean compactness'>
```



```
y_pred=knn.predict(X_test)
plt.scatter(X_test["mean area"],X_test["mean compactness"],c=y_pred,cmap="coolwarm",alpha=0.7) #size of graph
plt.show()
```

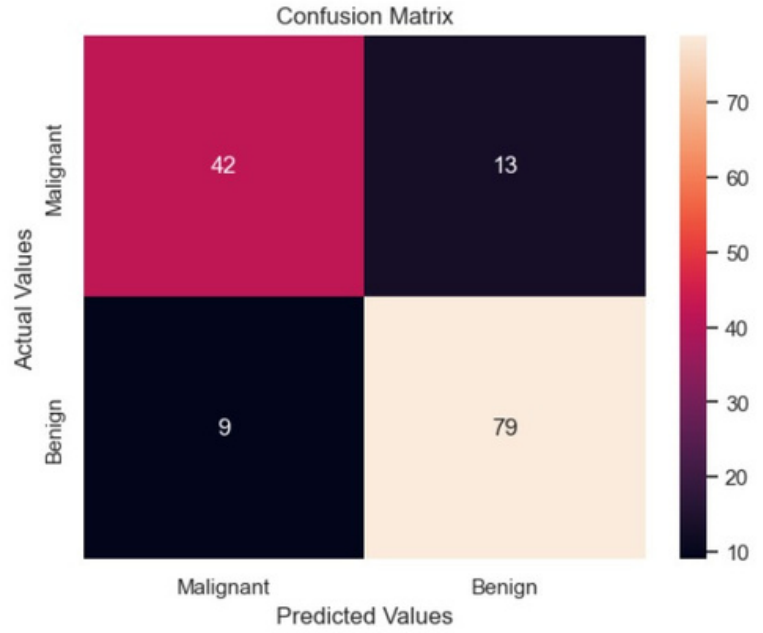



```
#Y_test--> actual  Y_pred-->Predicted
cf=confusion_matrix(Y_test,y_pred)
print(cf)
```

```
[[42 13]
 [ 9 79]]
```

```
labels=["True Negative","False Positive","False Negative","True Positive"]
labels=np.asarray(labels).reshape(2,2)
categories=["Zero","One"]
ax=plt.subplot()
sns.heatmap(cf,annot=True,ax=ax)
ax.set_xlabel("Predicted Values")
ax.set_ylabel("Actual Values")
ax.set_title("Confusion Matrix")
ax.xaxis.set_ticklabels(["Malignant","Benign"])
ax.yaxis.set_ticklabels(["Malignant","Benign"])
```

```
[Text(0, 0.5, 'Malignant'), Text(0, 1.5, 'Benign')]
```



```
tp,fn,fp,tn = confusion_matrix(Y_test,y_pred,labels=[1,0]).reshape(-1)
print("Values of tp fn fp tn" ,tp, fn, fp, tn)
#fn is better than fp
```

```
Values of tp fn fp tn 79 9 13 42
```

```
accuracy=(tp+tn)/(tp+fp+fn+tn)
print(accuracy)
```

```
0.8461538461538461
```

```
Precision=tp/(fp+tp)
print(Precision)
```

```
0.8586956521739131
```

```
Recall=tp/(tp+fn)
print(Recall)
```

```
0.8977272727272727
```

```
from sklearn.metrics import f1_score
f1_score(Y_test,y_pred)
```

```
0.8777777777777778
```

```
from sklearn.metrics import roc_auc_score
roc_auc_score(Y_test,y_pred)
```

```
0.8306818181818182
```

Practical 8:

AIM: Introduction to NOSQL using MongoDB

1.Create a database Institution ,Create a Collection Staff and Insert ten documents in it with fields: empid,empname,salary and designation.

• Display all documents in Staff and display only empid and designation.db

•

```
C:\Windows\System32\cmd.exe - mongo
> use Institution
switched to db Institution
> db.staff.insertMany( [
...  { empid: 101, empname: "Bharat", salary: 80000, designation: "Engineer" },
...  { empid: 102, empname: "Kavita", salary: 105000, designation: "Accountant" },
...  { empid: 103, empname: "Suchita", salary: 65000, designation: "Accountant" },
...  { empid: 104, empname: "Suraj", salary: 60000, designation: "Manager" },
...  { empid: 105, empname: "Pratik", salary:120000, designation: "Developer" },
...  { empid: 106, empname: "Gayatri", salary: 85000, designation: "Manager" },
...  { empid: 107, empname: "Supriya", salary: 75000, designation: "HR" },
...  { empid: 108, empname: "Atharva", salary: 115000, designation: "Accountant" },
...  { empid: 109, empname: "Omkar", salary: 67000, designation: "DGM" },
...  { empid: 110, empname: "Sahil", salary: 60000, designation: "Manager" },
... ] );
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("65eeb425c65fc88629c9b3f7"),
                ObjectId("65eeb425c65fc88629c9b3f8"),
                ObjectId("65eeb425c65fc88629c9b3f9"),
                ObjectId("65eeb425c65fc88629c9b3fa"),
                ObjectId("65eeb425c65fc88629c9b3fb"),
                ObjectId("65eeb425c65fc88629c9b3fc"),
                ObjectId("65eeb425c65fc88629c9b3fd"),
                ObjectId("65eeb425c65fc88629c9b3fe"),
                ObjectId("65eeb425c65fc88629c9b3ff"),
                ObjectId("65eeb425c65fc88629c9b400")
        ]
}
```

- Sort the documents in descending order of Salary

```
> db.staff.find({},{_id: 0,empid: 1,salary: 1}).sort({salary:-1})
{ "empid" : 105, "salary" : 120000 }
{ "empid" : 108, "salary" : 115000 }
{ "empid" : 102, "salary" : 105000 }
{ "empid" : 106, "salary" : 85000 }
{ "empid" : 101, "salary" : 80000 }
{ "empid" : 107, "salary" : 75000 }
{ "empid" : 109, "salary" : 67000 }
{ "empid" : 103, "salary" : 65000 }
{ "empid" : 104, "salary" : 60000 }
{ "empid" : 110, "salary" : 60000 }
>
```

• Display employee with designation with "Manager" or salary greater than Rs. 50,000/-.

```
> db.staff.find({ $or:[ {designation: "Manager"}, {salary: {$gt:50000}}]},{_id:0,salary:1,designation:1})
{ "salary" : 80000, "designation" : "Engineer" }
{ "salary" : 60000, "designation" : "Manager" }
{ "salary" : 85000, "designation" : "Manager" }
{ "salary" : 75000, "designation" : "HR" }
{ "salary" : 67000, "designation" : "DGM" }
{ "salary" : 60000, "designation" : "Manager" }
>
```

• Update the salary of all employees with designation as "Accountant" to Rs.45000.

```
> db.staff.updateMany({designation:"Accountant"}, {$set: {salary:45000}} );
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
> db.staff.find({designation:"Accountant"})
{ "_id" : ObjectId("65eeb425c65fc88629c9b3f8"), "empid" : 102, "empname" : "Kavita", "salary" : 45000, "designation" : "Accountant" }
{ "_id" : ObjectId("65eeb425c65fc88629c9b3f9"), "empid" : 103, "empname" : "Suchita", "salary" : 45000, "designation" : "Accountant" }
{ "_id" : ObjectId("65eeb425c65fc88629c9b3fe"), "empid" : 108, "empname" : "Atharva", "salary" : 45000, "designation" : "Accountant" }
>
```

• Remove the documents of employees whose salary is greater than Rs100000.

```
> db.staff.remove({salary: {$gt:100000}})
WriteResult({ "nRemoved" : 1 })
> db.staff.find()
{ "_id" : ObjectId("65eeb425c65fc88629c9b3f7"), "empid" : 101, "empname" : "Bharat", "salary" : 80000, "designation" : "Engineer" }
{ "_id" : ObjectId("65eeb425c65fc88629c9b3f8"), "empid" : 102, "empname" : "Kavita", "salary" : 45000, "designation" : "Accountant" }
{ "_id" : ObjectId("65eeb425c65fc88629c9b3f9"), "empid" : 103, "empname" : "Suchita", "salary" : 45000, "designation" : "Accountant" }
{ "_id" : ObjectId("65eeb425c65fc88629c9b3fa"), "empid" : 104, "empname" : "Suraj", "salary" : 60000, "designation" : "Manager" }
{ "_id" : ObjectId("65eeb425c65fc88629c9b3fc"), "empid" : 106, "empname" : "Gayatri", "salary" : 85000, "designation" : "Manager" }
{ "_id" : ObjectId("65eeb425c65fc88629c9b3fd"), "empid" : 107, "empname" : "Supriya", "salary" : 75000, "designation" : "HR" }
{ "_id" : ObjectId("65eeb425c65fc88629c9b3fe"), "empid" : 108, "empname" : "Atharva", "salary" : 45000, "designation" : "Accountant" }
{ "_id" : ObjectId("65eeb425c65fc88629c9b3ff"), "empid" : 109, "empname" : "Omkar", "salary" : 67000, "designation" : "DGM" }
{ "_id" : ObjectId("65eeb425c65fc88629c9b400"), "empid" : 110, "empname" : "Sahil", "salary" : 60000, "designation" : "Manager" }
>
```

2. Create a database Institution . Create a Collection Student and Insert ten documents in it with fields: RollNo, Name, Class and TotalMarks (out of 500).

• Display all documents in Student.

```
> db.students.insertMany( [
...  { rollno: 01, name: "Rishi", class: "MSC", totalmarks: 390 },
...  { rollno: 02, name: "Vaishnavi", class: "TYCS", totalmarks: 496 },
...  { rollno: 03, name: "Purva", class: "TYCS", totalmarks: 480 },
...  { rollno: 04, name: "Sakshi", class: "TYCS", totalmarks: 490 },
...  { rollno: 05, name: "Atharva", class: "TYCS", totalmarks: 380 },
...  { rollno: 06, name: "Vinitha", class: "MSC", totalmarks: 355 },
...  { rollno: 07, name: "Saloni", class: "TYCS", totalmarks: 385 },
...  { rollno: 08, name: "Shifa", class: "MSC", totalmarks: 360 },
...  { rollno: 09, name: "Om", class: "TYCS", totalmarks: 368 },
...  { rollno: 10, name: "Kimaya", class: "MSC", totalmarks: 395 },
... ] );
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("65eebe1ac65fc88629c9b415"),
                ObjectId("65eebe1ac65fc88629c9b416"),
                ObjectId("65eebe1ac65fc88629c9b417"),
                ObjectId("65eebe1ac65fc88629c9b418"),
                ObjectId("65eebe1ac65fc88629c9b419"),
                ObjectId("65eebe1ac65fc88629c9b41a"),
                ObjectId("65eebe1ac65fc88629c9b41b"),
                ObjectId("65eebe1ac65fc88629c9b41c"),
                ObjectId("65eebe1ac65fc88629c9b41d"),
                ObjectId("65eebe1ac65fc88629c9b41e")
        ]
}
> db.students.find()
{ "_id" : ObjectId("65eebe1ac65fc88629c9b415"), "rollno" : 1, "name" : "Rishi", "class" : "MSC", "totalmarks" : 390 }
{ "_id" : ObjectId("65eebe1ac65fc88629c9b416"), "rollno" : 2, "name" : "Vaishnavi", "class" : "TYCS", "totalmarks" : 496 }
{ "_id" : ObjectId("65eebe1ac65fc88629c9b417"), "rollno" : 3, "name" : "Purva", "class" : "TYCS", "totalmarks" : 480 }
{ "_id" : ObjectId("65eebe1ac65fc88629c9b418"), "rollno" : 4, "name" : "Sakshi", "class" : "TYCS", "totalmarks" : 490 }
{ "_id" : ObjectId("65eebe1ac65fc88629c9b419"), "rollno" : 5, "name" : "Atharva", "class" : "TYCS", "totalmarks" : 380 }
{ "_id" : ObjectId("65eebe1ac65fc88629c9b41a"), "rollno" : 6, "name" : "Vinitha", "class" : "MSC", "totalmarks" : 355 }
{ "_id" : ObjectId("65eebe1ac65fc88629c9b41b"), "rollno" : 7, "name" : "Saloni", "class" : "TYCS", "totalmarks" : 385 }
{ "_id" : ObjectId("65eebe1ac65fc88629c9b41c"), "rollno" : 8, "name" : "Shifa", "class" : "MSC", "totalmarks" : 360 }
{ "_id" : ObjectId("65eebe1ac65fc88629c9b41d"), "rollno" : 9, "name" : "Om", "class" : "TYCS", "totalmarks" : 368 }
{ "_id" : ObjectId("65eebe1ac65fc88629c9b41e"), "rollno" : 10, "name" : "Kimaya", "class" : "MSC", "totalmarks" : 395 }
>
```

- Sort the documents in descending order of TotalMarks.

```
> db.students.find({},{_id: 0,name: 1,totalmarks: 1}).sort({totalmarks:-1})
{ "name" : "Vaishnavi", "totalmarks" : 496 }
{ "name" : "Sakshi", "totalmarks" : 490 }
{ "name" : "Purva", "totalmarks" : 480 }
{ "name" : "Kimaya", "totalmarks" : 395 }
{ "name" : "Rishi", "totalmarks" : 390 }
{ "name" : "Saloni", "totalmarks" : 385 }
{ "name" : "Atharva", "totalmarks" : 380 }
{ "name" : "Om", "totalmarks" : 368 }
{ "name" : "Shifa", "totalmarks" : 360 }
{ "name" : "Vinitha", "totalmarks" : 355 }
>
```

- Display students of class "MSc" or marks greater than 400.

```
> db.students.find({ $or:[ {class: "MSC"}, {totalmarks: {$gt:400}}]},{_id:0})
{ "rollno" : 1, "name" : "Rishi", "class" : "MSC", "totalmarks" : 390 }
{ "rollno" : 2, "name" : "Vaishnavi", "class" : "TYCS", "totalmarks" : 496 }
{ "rollno" : 3, "name" : "Purva", "class" : "TYCS", "totalmarks" : 480 }
{ "rollno" : 4, "name" : "Sakshi", "class" : "TYCS", "totalmarks" : 490 }
{ "rollno" : 6, "name" : "Vinitha", "class" : "MSC", "totalmarks" : 355 }
{ "rollno" : 8, "name" : "Shifa", "class" : "MSC", "totalmarks" : 360 }
{ "rollno" : 10, "name" : "Kimaya", "class" : "MSC", "totalmarks" : 395 }
>
```

- Remove all the documents with TotalMarks<200

```
> db.students.remove({ totalmarks: {$lt: 200}})
WriteResult({ "nRemoved" : 4 })
> db.students.find()
{ "_id" : ObjectId("65eebe1ac65fc88629c9b416"), "rollno" : 2, "name" : "Vaishnavi", "class" : "TYCS", "totalmarks" : 496 }
{ "_id" : ObjectId("65eebe1ac65fc88629c9b417"), "rollno" : 3, "name" : "Purva", "class" : "TYCS", "totalmarks" : 480 }
{ "_id" : ObjectId("65eebe1ac65fc88629c9b418"), "rollno" : 4, "name" : "Sakshi", "class" : "TYCS", "totalmarks" : 490 }
{ "_id" : ObjectId("65eebe1ac65fc88629c9b419"), "rollno" : 5, "name" : "Atharva", "class" : "TYCS", "totalmarks" : 380 }
{ "_id" : ObjectId("65eebe1ac65fc88629c9b41b"), "rollno" : 7, "name" : "Saloni", "class" : "TYCS", "totalmarks" : 385 }
{ "_id" : ObjectId("65eebe1ac65fc88629c9b41d"), "rollno" : 9, "name" : "Om", "class" : "TYCS", "totalmarks" : 368 }
>
```