



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. **CHAUHAN PANKAJ YAMUNAPRASAD**

Roll No **TCS2324007** 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 **T.Y. BSc. Computer Science [Semester-VI]**.

Prof. In-Charge
Prof. Maya Nair

Examination Date:

Examiner's Signature & Date:

HOD's Signature & Date:
Dr. Manoj Singh

College Seal
And
Date

Index Page

Sr. No	Description	Page No	Date	Faculty Signature
1	Write a python program a plot word cloud for a Wikipedia page of any topic	5	16/11/23	
2	Write a python program to perform web scrapping 1) html scrapping – use beautiful soup 2) json scrapping	8	23/11/23	
3	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) mtcars.csv: Motor Trend Car Road Tests-The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973--74 models). Format A data frame with 32 observations on 12 (numeric) variables. [, 1] mpg Miles/(US) gallon [, 2] cyl Number of cylinders	11	06/01/24	

	<p>[, 3] disp Displacement (cu.in.)</p> <p>[, 4] hp Gross horsepower</p> <p>[, 5] drat Rear axle ratio</p> <p>[, 6] wt Weight (1000 lbs)</p> <p>[, 7] qsec 1/4 mile time</p> <p>[, 8] vs Engine (0 = V-shaped, 1 = straight)</p> <p>[, 9] am Transmission (0 = automatic, 1 = manual)</p> <p>[,10] gear Number of forward gears</p> <p>[,11] Carb Number of carburetors</p> <p>Motor Trend is a magazine about the automobile industry. Looking at a data set of a collection of cars,</p> <p>they are interested in exploring the relationship between a set of variables and miles per gallon (MPG)</p> <p>(outcome).</p>			
4	Exploratory data analysis in python using Titanic dataset	24	11/01/24	
5	<p>Exploratory data analysis in Python using Titanic Dataset</p> <p>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.</p> <p>2) Write a python program to simulate linear model</p>	29	25/01/24	

	$Y=10+7*x+e$ for random 100 samples and visualize univariate linear regression on it.			
6	Write a python program to implement multiple linear regression on the Dataset Boston.csv	34	07/02/24	
7	Write a python program to implement KNN algorithm to predict breast cancer using breast cancer Wisconsin dataset	38	23/02/24	
8	Introduction to NOSQL using MongoDB	42	23/02/24	

Practical no 1

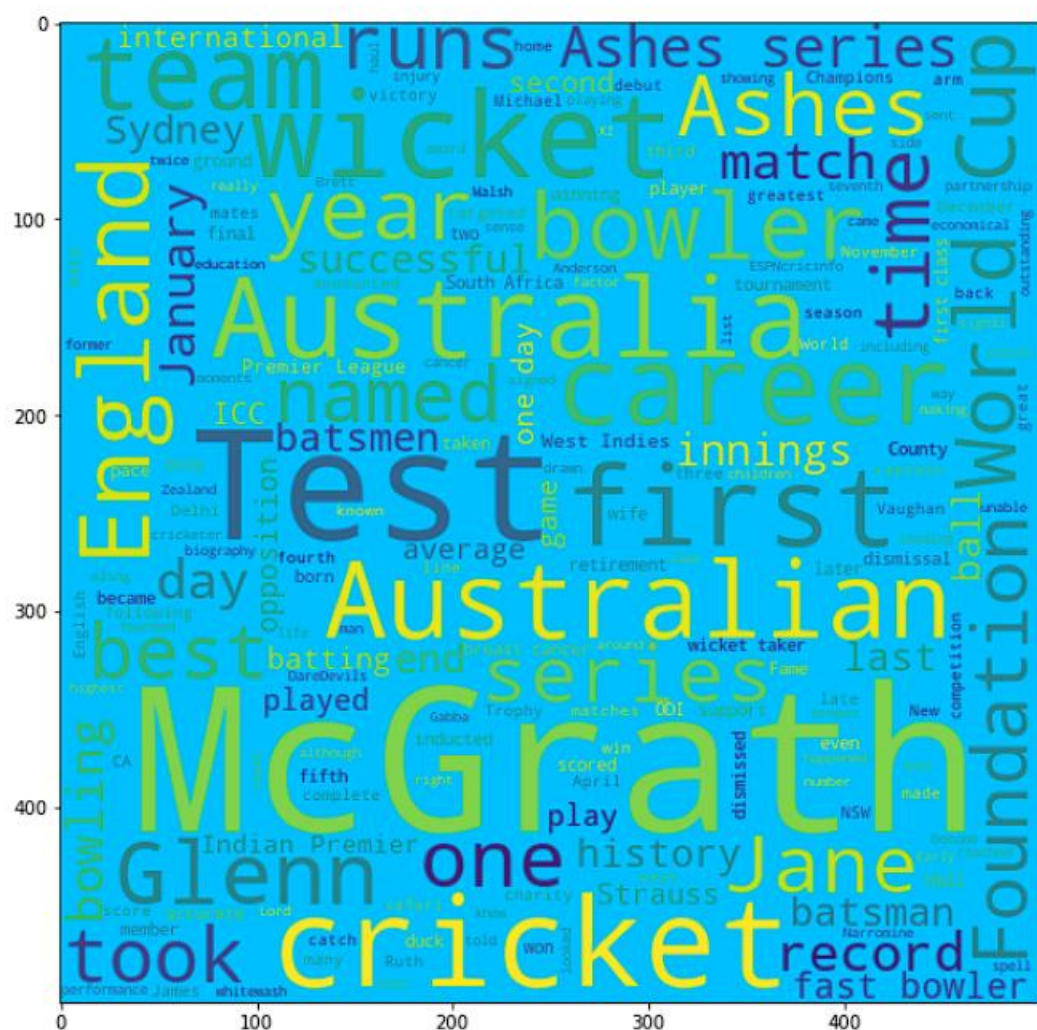
Aim: Write a python program to plot word cloud for a Wikipedia page of any topic

Code:

```
1 from wordcloud import WordCloud, STOPWORDS
2 import matplotlib.pyplot as plt
3 import wikipedia as wp
4
5 result = wp.page("glenn mcgrath") # accessing the wikipedia page
6 final_result = result.content
7 def plot_wordcloud(wc):
8     plt.axis("off")
9     plt.figure(figsize=(10,10))
10    plt.imshow(wc)
11    plt.show()
12 wc=WordCloud(width=500, height=500, background_color="deepskyblue", random_state=10,
13             stopwords=STOPWORDS).generate(final_result)
14 wc.to_file("ComSc.png")
15 plot_wordcloud(wc)
```

Output:

Computer science is the study of computation, information, and automation. Computer science spans theoretical disciplines (such as algorithms, theory of computation, and information theory) to applied disciplines (including the design and implementation of hardware and software). Though more often considered an academic discipline, computer science is closely related to computer programming. Algorithms and data structures are central to computer science. The theory of computation concerns abstract models of computation and general classes of problems that can be solved using them. The fields of cryptography and computer security involve studying the means for secure communication and for preventing security vulnerabilities. Computer graphics and computational geometry address the generation of images. Programming language theory considers different ways to describe computational processes, and database theory concerns the management of repositories of data. Human-computer interaction investigates the interfaces through which humans and computers interact, and software engineering focuses on the design and principles behind developing software. Areas such as operating systems, networks and embedded systems investigate the principles and design behind complex systems. Computer architecture describes the construction of computer components and computer-operated equipment. Artificial intelligence and machine learning aim to synthesize goal-oriented processes such as problem-solving, decision-making, environmental adaptation, planning and learning found in humans and animals. Within artificial intelligence, computer vision aims to understand and process image and video data, while natural language processing aims to understand and process textual and linguistic data. The fundamental concern of computer science is determining what can and cannot be automated. The Turing Award is generally recognized as the highest distinction in computer science.

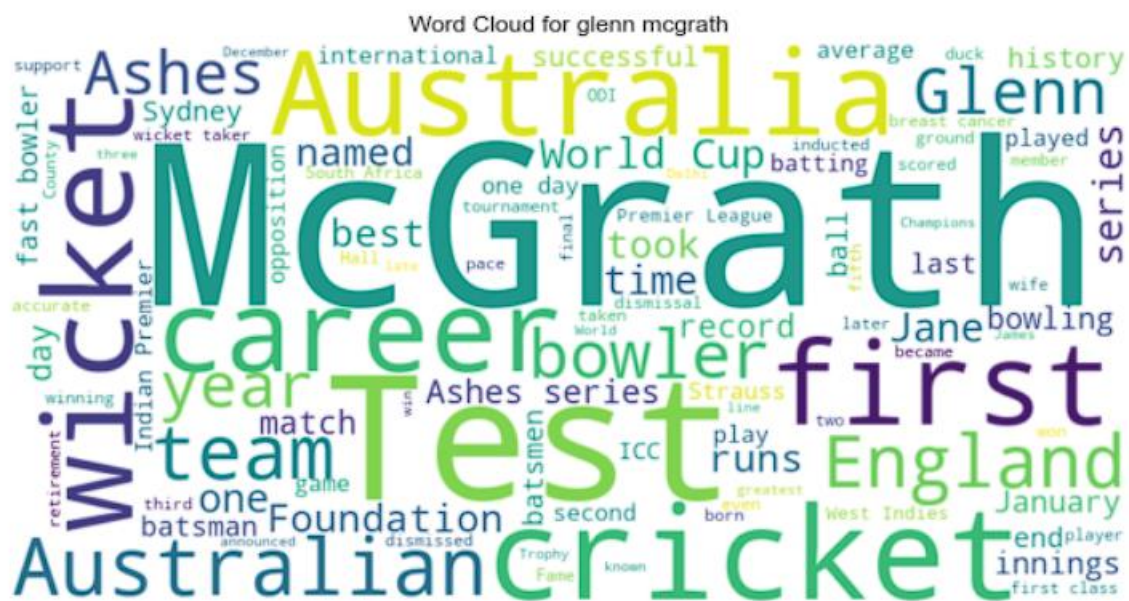


Code:

```
1 import wikipedia
2 from wordcloud import WordCloud
3 import matplotlib.pyplot as plt
4
5 def generate_word_cloud(topic, num_words=100):
6     # Get content from the Wikipedia page
7     page_content = wikipedia.page(topic).content
8
9     # Generate word cloud
10    wordcloud = WordCloud(width=800, height=400, max_words=num_words, background_color='white').generate(page_content)
11
12    # Display the word cloud using matplotlib
13    plt.figure(figsize=(10, 8))
14    plt.imshow(wordcloud, interpolation='bilinear')
15    plt.axis('off')
16    plt.title(f"Word Cloud for {topic}")
17    plt.show()
18
19 if __name__ == "__main__":
20     # Enter the topic for the Wikipedia page
21     topic = input("Enter the topic for the Wikipedia page: ")
22     generate_word_cloud(topic)
```

Output:

Enter the topic for the Wikipedia page: glenn mcgrath



Practical no 2

Aim: Write a python program to perform Web Scapping

Web scrapping: web scrapping is the process of collecting and parsing raw data from the web.

01. HTML scrapping – use beautiful soup library

Code:

```
1 import pandas as pd
2 from bs4 import BeautifulSoup
3 from urllib.request import urlopen
4
5
6 url="https://en.wikipedia.org/wiki/List_of_Asian_countries_by_area"
7 page=urlopen(url)
8 html_page=page.read().decode("utf-8")
9 soup=BeautifulSoup(html_page,"html.parser")
10 table=soup.find("table")
11 SrNo=[]
12 Country=[]
13 Area=[]
14 rows=table.find("tbody").find_all("tr")
15 for row in rows:
16     cells=row.find_all("td")
17     if(cells):
18         SrNo.append(cells[0].get_text().strip("\n"))
19         Country.append(cells[1].get_text().strip("\xa0").strip("\n"))
20         Area.append(cells[3].get_text().strip("\n").replace(",",""))
21 country_df=pd.DataFrame()
22 country_df["ID"]=SrNo
23 country_df["Country"]=Country
24 country_df["Area"]=Area
25
26 print(country_df.head(10))
27 #print(SrNo)
28 #print(Country)
29 #print(Area)
```

Output:

	ID	Country	Area
0	1	Russi	13083100 (5051400)
1	2	Chin	9596961 (3705407)
2	3	Indi	3287263 (1269219)
3	4	Kazakhstan	2600000 (1000000)
4	5	Saudi Arabi	2149690 (830000)
5	6	Iran	1648195 (636372)
6	7	Mongoli	1564110 (603910)
7	8	Indonesi	1488509 (574717)
8	9	Pakistan	881913 (340509)
9	10	Turkey	759805 (293362)

02. JSON Scrapping

Code:

```

1 import pandas as pd
2 import urllib.request
3 import json
4
5 url="https://jsonplaceholder.typicode.com/users"
6 response=urllib.request.urlopen(url)
7 data=json.loads(response.read())
8
9 id=[]
10 username=[]
11 email=[]
12
13 for item in data:
14     if "id" in item.keys():
15         id.append(item["id"])
16     else:
17         id.append("NA")
18     if "username" in item.keys():
19         username.append(item["username"])
20     else:
21         username.append("NA")
22     if "email" in item.keys():
23         email.append(item["email"])
24     else:
25         email.append("NA")
26
27 user_df=pd.DataFrame()
28 user_df["User ID"]=id
29 user_df["User Name"]=username
30 user_df["Email Address"]=email
31
32 print(user_df.head(10))

```

Output:

	User ID	User Name	Email Address
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 no 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)

mtcars.csv:

Motor Trend Car Road Tests-The data was extracted from the 1974 Motor Trend US magazine, and

comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles

(1973--74 models).

Format

A data frame with 32 observations on 12 (numeric) variables.

[, 1] mpg Miles/(US) gallon

[, 2] cyl Number of cylinders

[, 3] disp Displacement (cu.in.)

[, 4] hp Gross horsepower

[, 5] drat Rear axle ratio

[, 6] wt Weight (1000 lbs)

[, 7] qsec 1/4 mile time

[, 8] vs Engine (0 = V-shaped, 1 = straight)

[, 9] am Transmission (0 = automatic, 1 = manual)

[,10] gear Number of forward gears

[,11] Carb Number of carburetors

Motor Trend is a magazine about the automobile industry. Looking at a data set of a collection of cars,

they are interested in exploring the relationship between a set of variables and miles per gallon (MPG)

(outcome).

Code:

```
1 getwd()
2 cars_df=read.csv("mtcars.csv")
3 View(cars_df)
4 str(cars_df)
5 dim(cars_df)
6 names(cars_df)
7 row.names(cars_df)
8 row.names(cars_df)=cars_df$model
9 cars_df=cars_df[,-1]
10 View(cars_df)
11
12 library(dplyr)
13
14 # select function - for extracting specific columns
15 # df1 = select(cars_df,mpg:hp)
16 df1=cars_df %>% select(mpg:hp) #pipe of dplyr it will take out content of one column to the output of other
17 View(df1)
18 df1 = cars_df %>% select(c(mpg,disp,wt,gear))
19 View(df1)
20
21 #Filter function - for extracting specific rows or observation
22 #extract record where gears=4 and columns to be displayed are mpg, disp, wt and gears.
23 df1 = cars_df %>% filter(gear==4) %>% select(c(mpg,disp,wt,gear))
24 View(df1)
25
26 # extract record where cyl=4 or mpg>20 and columns are required are mpg,cyl
27 df1 = cars_df %>% filter(cyl==4 | mpg > 20) %>% select(c(mpg,cyl))
28 View(df1)
29
30 #extract records where mpg<20 and carb = 3 and columns needed are mpg and carb
31 df1 = cars_df %>% filter(mpg < 20 & carb == 3) %>% select(c(mpg,carb))
32 View(df1)
33
34 # Arrange function -Sort as per specific columns
35 df1 =cars_df %>% arrange(cyl,desc(mpg))
36 View(df1)
```

```

37
38 #Rename function - change names of one or more column
39 df1 = cars_df %>% rename(MilesPerGallon=mpg,Cylinders=cyl,Displacement=disp)
40 view(df1)
41
42 #Mutate function - creating new columns on the basis of existing column
43 df1 = cars_df %>% mutate(Power=hp*wt)
44 view(df1)
45
46 #Group_by and summaries - segregating data as per categorical variable and summarizing
47 df1$gear = as.factor(df1$gear)
48 str(df1)
49
50 summary_df = df1%>% group_by(df1$gear) %>% summarise(no=n(), mean_mpg=mean(mpg), mean_wt=mean(wt))
51 summary_df
52
53 summary_df = df1%>% group_by(df1$cylinders) %>% summarise(no=n(), mean_mpg=mean(mpg), mean_wt=mean(wt))
54 summary_df
55
56
57 #Data Visualization
58 #histogram - for single column frequency
59 hist(df1$mpg, main="Histogram of MilesPerGallon(mtcars)", col="lightgreen", xlab="Miles Per Gallon")
60 #box plot - diagrammatic representation of summary
61 summary(df1$mpg)
62 boxplot(df1$mpg)
63
64 #bar plot - categorical variable representation'
65 table(df1$gear)
66 barplot(table(df1$gear))
67
68 #scatter plot - plot() - plots relationship between two variable
69 plot(df1$mpg~df1$disp)
70 plot(df1$mpg~df1$cyl)
71 plot(df1$mpg~df1$wt)
72

```

Output:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
1	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
2	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
3	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
4	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
5	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
6	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
7	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
8	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
9	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
10	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
11	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
12	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
13	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
14	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
15	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
16	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
17	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
18	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
19	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
20	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1

```

> str(cars_df)
'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)
[1] 32 12

> names(cars_df)
[1] "model" "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear" "carb"

> row.names(cars_df)
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12" "13" "14" "15" "16" "17" "18" "19" "20" "21"
[22] "22" "23" "24" "25" "26" "27" "28" "29" "30" "31" "32"

```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1

	mpg	cyl	disp	hp
Mazda RX4	21.0	6	160.0	110
Mazda RX4 Wag	21.0	6	160.0	110
Datsun 710	22.8	4	108.0	93
Hornet 4 Drive	21.4	6	258.0	110
Hornet Sportabout	18.7	8	360.0	175
Valiant	18.1	6	225.0	105
Duster 360	14.3	8	360.0	245
Merc 240D	24.4	4	146.7	62
Merc 230	22.8	4	140.8	95
Merc 280	19.2	6	167.6	123
Merc 280C	17.8	6	167.6	123
Merc 450SE	16.4	8	275.8	180
Merc 450SL	17.3	8	275.8	180
Merc 450SLC	15.2	8	275.8	180
Cadillac Fleetwood	10.4	8	472.0	205
Lincoln Continental	10.4	8	460.0	215
Chrysler Imperial	14.7	8	440.0	230
Fiat 128	32.4	4	78.7	66
Honda Civic	30.4	4	75.7	52
Toyota Corolla	33.9	4	71.1	65

	mpg	disp	wt	gear
Mazda RX4	21.0	160.0	2.620	4
Mazda RX4 Wag	21.0	160.0	2.875	4
Datsun 710	22.8	108.0	2.320	4
Hornet 4 Drive	21.4	258.0	3.215	3
Hornet Sportabout	18.7	360.0	3.440	3
Valiant	18.1	225.0	3.460	3
Duster 360	14.3	360.0	3.570	3
Merc 240D	24.4	146.7	3.190	4
Merc 230	22.8	140.8	3.150	4
Merc 280	19.2	167.6	3.440	4
Merc 280C	17.8	167.6	3.440	4
Merc 450SE	16.4	275.8	4.070	3
Merc 450SL	17.3	275.8	3.730	3
Merc 450SLC	15.2	275.8	3.780	3
Cadillac Fleetwood	10.4	472.0	5.250	3
Lincoln Continental	10.4	460.0	5.424	3
Chrysler Imperial	14.7	440.0	5.345	3
Fiat 128	32.4	78.7	2.200	4
Honda Civic	30.4	75.7	1.615	4
Toyota Corolla	33.9	71.1	1.835	4

	<div>mpg</div>	<div>disp</div>	<div>wt</div>	<div>gear</div>
Mazda RX4	21.0	160.0	2.620	4
Mazda RX4 Wag	21.0	160.0	2.875	4
Datsun 710	22.8	108.0	2.320	4
Merc 240D	24.4	146.7	3.190	4
Merc 230	22.8	140.8	3.150	4
Merc 280	19.2	167.6	3.440	4
Merc 280C	17.8	167.6	3.440	4
Fiat 128	32.4	78.7	2.200	4
Honda Civic	30.4	75.7	1.615	4
Toyota Corolla	33.9	71.1	1.835	4
Fiat X1-9	27.3	79.0	1.935	4
Volvo 142E	21.4	121.0	2.780	4

	<div>mpg</div>	<div>cyl</div>
Mazda RX4	21.0	6
Mazda RX4 Wag	21.0	6
Datsun 710	22.8	4
Hornet 4 Drive	21.4	6
Merc 240D	24.4	4
Merc 230	22.8	4
Fiat 128	32.4	4
Honda Civic	30.4	4
Toyota Corolla	33.9	4
Toyota Corona	21.5	4
Fiat X1-9	27.3	4
Porsche 914-2	26.0	4
Lotus Europa	30.4	4
Volvo 142E	21.4	4

	mpg	cyl
Mazda RX4	21.0	6
Mazda RX4 Wag	21.0	6
Datsun 710	22.8	4
Hornet 4 Drive	21.4	6
Merc 240D	24.4	4
Merc 230	22.8	4
Fiat 128	32.4	4
Honda Civic	30.4	4
Toyota Corolla	33.9	4
Toyota Corona	21.5	4
Fiat X1-9	27.3	4
Porsche 914-2	26.0	4
Lotus Europa	30.4	4
Volvo 142E	21.4	4

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4

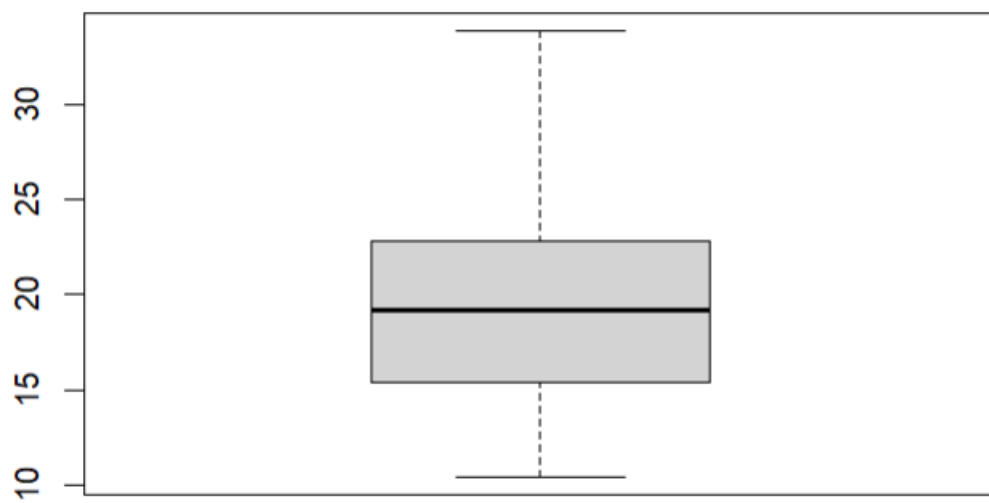
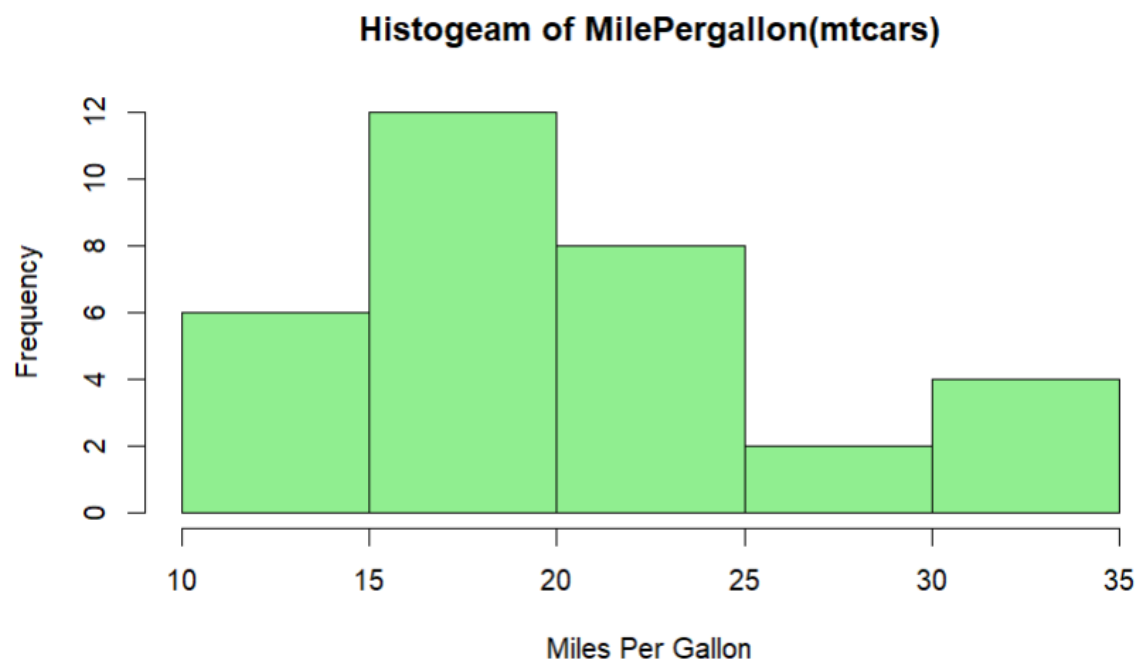
	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4

```
> str(df1)
'data.frame': 32 obs. of 11 variables:
 $ mpg : num 33.9 32.4 30.4 30.4 27.3 26 24.4 22.8 22.8 21.5 ...
 $ cyl : int 4 4 4 4 4 4 4 4 4 4 ...
 $ disp: num 71.1 78.7 75.7 95.1 79 ...
 $ hp : int 65 66 52 113 66 91 62 93 95 97 ...
 $ drat: num 4.22 4.08 4.93 3.77 4.08 4.43 3.69 3.85 3.92 3.7 ...
 $ wt : num 1.83 2.2 1.61 1.51 1.94 ...
 $ qsec: num 19.9 19.5 18.5 16.9 18.9 ...
 $ vs : int 1 1 1 1 1 0 1 1 1 1 ...
 $ am : int 1 1 1 1 1 1 0 1 0 0 ...
 $ gear: int 4 4 4 5 4 5 4 4 4 3 ...
 $ carb: int 1 1 2 2 1 2 2 1 2 1 ...
```

```
> summary_df = df1%>% group_by(df1$gear) %>% summarise(no=n(), mean_mpg=mean(mpg), mean_wt=mean(wt))
> summary_df
# A tibble: 3 × 4
  `df1$gear`    no mean_mpg mean_wt
  <int> <int>    <dbl>    <dbl>
1         3     15     16.1     3.89
2         4     12     24.5     2.62
3         5      5     21.4     2.63
> |
```

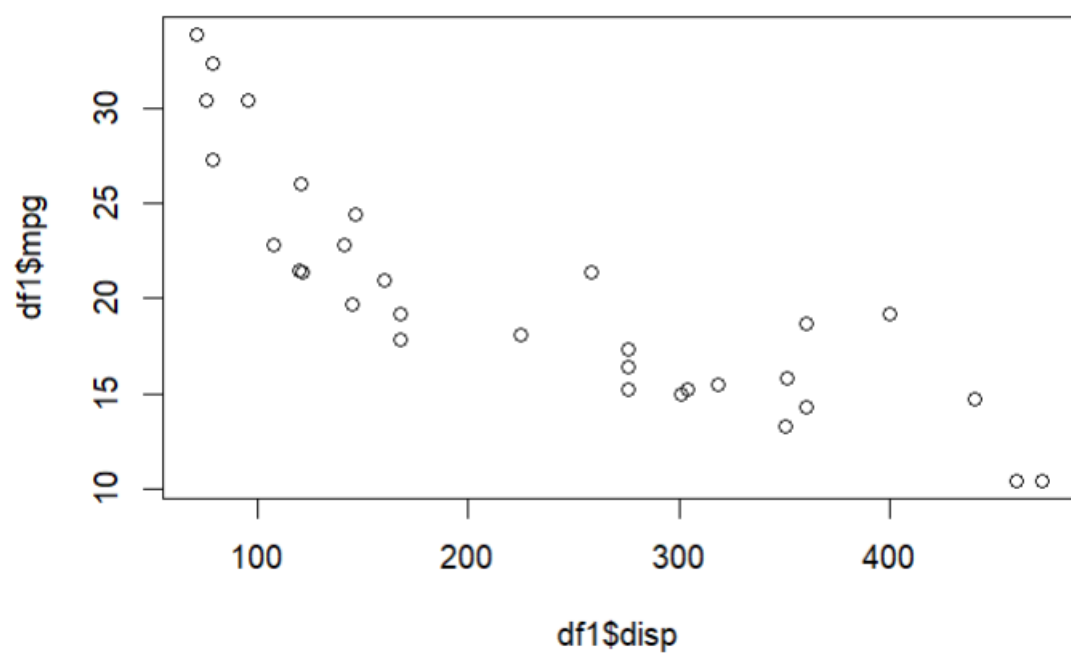
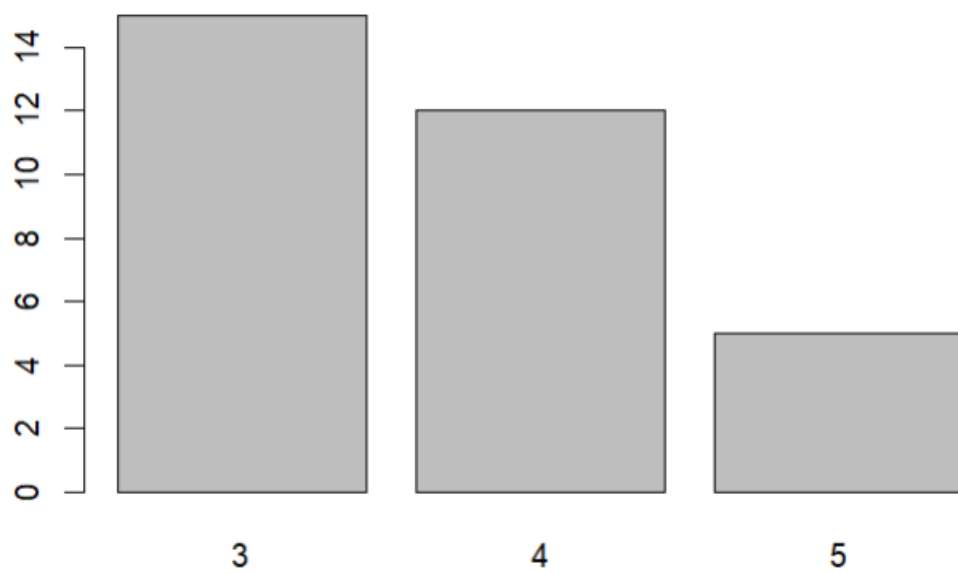
```
> summary_df = df1%>% group_by(df1$cylinders) %>% summarise(no=n(), mean_mpg=mean(mpg), mean_wt=mean(wt))
> summary_df
# A tibble: 1 × 3
  no mean_mpg mean_wt
  <int>    <dbl>    <dbl>
1    32     20.1     3.22
```

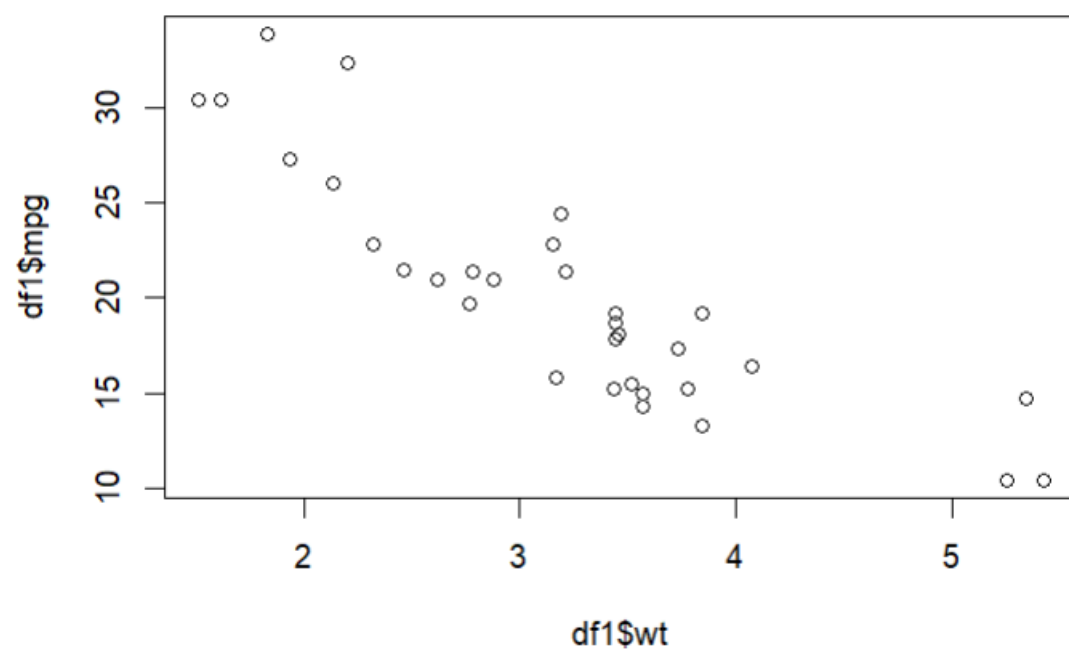
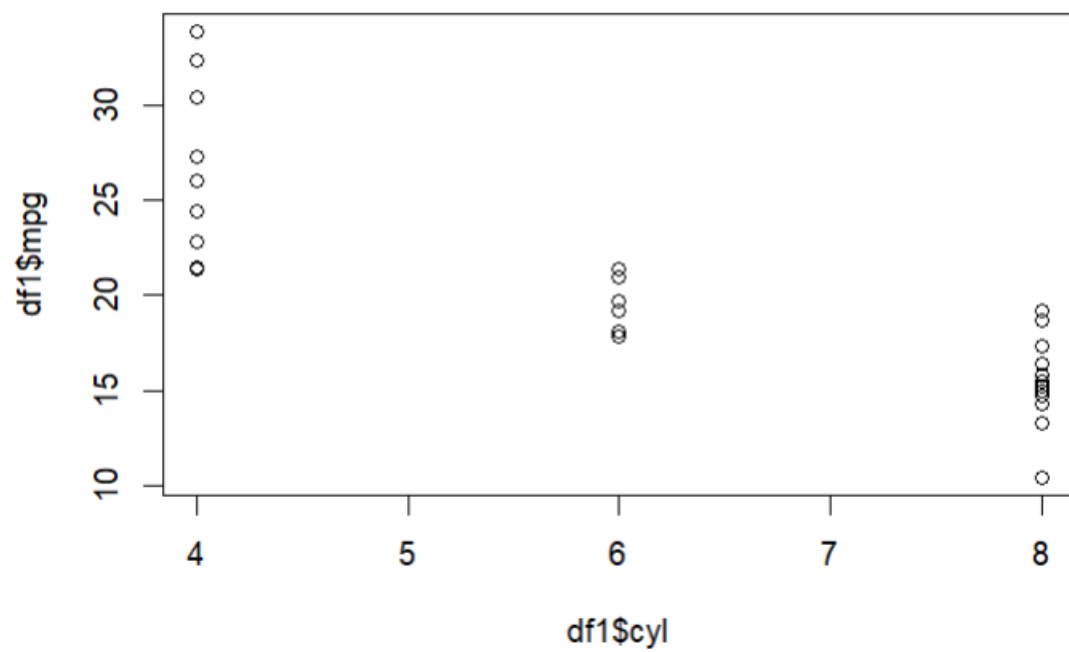
```
> #box plot - diagrammatic representation of summary
> summary(df1$mpg)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
10.40  15.43   19.20   20.09  22.80   33.90
> |
```



```
> table(df1$gear)
```

```
 3  4  5  
15 12  5
```





Practical no 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.

Data Dictionary

Variable	Definition	Key
Survival	Survival	0 = No, 1 = Yes
Pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
Sex	Sex	
Age	Age in years	
Sibsp	# of siblings / spouses aboard the Titanic	
Parch	# of parents / children aboard the Titanic	
Ticket	Ticket number	
Fare	Passenger fare	
Cabin	Cabin number	
Embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Seaborn:

It is a python library used to statistically visualize data. Seaborn, built over Matplotlib, provides a better interface and ease of usage. It can be installed using the following command,

pip3 install seaborn

Features: The titanic dataset has roughly the following types of features:

- Categorical/Nominal: Variables that can be divided into multiple categories but having no order or priority.

Eg. Embarked (C = Cherbourg; Q = Queenstown; S = Southampton)

- Binary: A subtype of categorical features, where the variable has only two categories.

Eg. Sex (Male/Female)

- Ordinal: They are similar to categorical features but they have an order(i.e can be sorted).

Eg. Pclass (1, 2, 3)

- Continuous: They can take up any value between the minimum and maximum values in a column.

Eg. Age, Fare

- Count: They represent the count of a variable.

Eg. SibSp, Parch

Useless: They don't contribute to the final outcome of an ML model. Here, *PassengerId*, *Name*, *Cabin* and *Ticket* might fall into this category.

Code:

```
1 import pandas as pd
2 titanic = pd.read_csv("train.csv")
3 titanic.head()
4 titanic.info()
5
6 titanic.describe()
7 titanic.isnull().sum()
8 titanic_cleaned = titanic.drop(['PassengerId', 'Name', 'Ticket', 'Fare', 'Cabin'], axis=1)
9 titanic_cleaned.info()
10
11 import seaborn as sns
12 import matplotlib.pyplot as plt
13 %matplotlib inline
14 sns.catplot(x="Sex", hue="Survived", kind="count", data=titanic_cleaned)
15
16 titanic_cleaned.groupby(['Sex', 'Survived'])['Survived'].count()
17
18 group1=titanic_cleaned.groupby(['Sex', 'Survived'])
19 sns.heatmap(group1, annot=True, fmt='d')
20
21 sns.violinplot(x="Sex", y="Age", hue="Survived", data=titanic_cleaned, split=True)
22 print("Oldest Person on Board:", titanic_cleaned['Age'].max())
23 print("Youngest Person on Board:", titanic_cleaned['Age'].min())
24 print("Average age of Person on Board:", titanic_cleaned['Age'].mean())
25
26 titanic_cleaned.isnull().sum()
27 def impute(cols):
28     Age = cols[0]
29     Pclass = cols[1]
30     if pd.isnull(Age):
31         if Pclass==1:
32             return 38
33         elif Pclass==2:
34             return 29
35         else:
36             return 24
37     else:
38         return Age
39 titanic_cleaned['Age'] = titanic_cleaned[['Age', 'Pclass']].apply(impute, axis=1)
40 titanic_cleaned.isnull().sum()
41 titanic_cleaned.corr(method='pearson')
42
43 sns.heatmap(titanic_cleaned.corr(method='pearson'), annot=True, vmax=1)
44
45 import numpy as np
46 from sklearn import datasets
47 x,y,coef=datasets.make_regression(n_samples=100, n_features=1, n_informative=1, noise=10,coef=True, random_state = 0)
48 x=np.interp(x,(x.min(),x.max()),(0,20))
49 print(len(x))
50 print(x)
51 y=np.interp(y,(y.min(),y.max()),(20000,150000))
52 print(len(y))
53 print(y)
```

Output:

```

<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

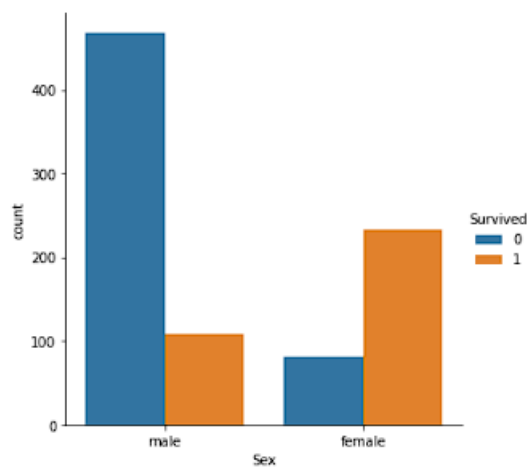
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 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   Embarked          889 non-null   object  
dtypes: float64(1), int64(4), object(2)
memory usage: 48.9+ KB

```

```
<seaborn.axisgrid.FacetGrid at 0x1cd8e899040>
```



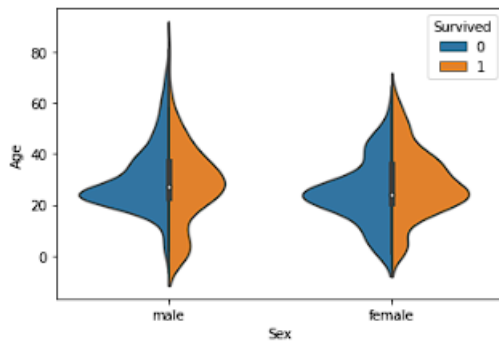
```

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

```

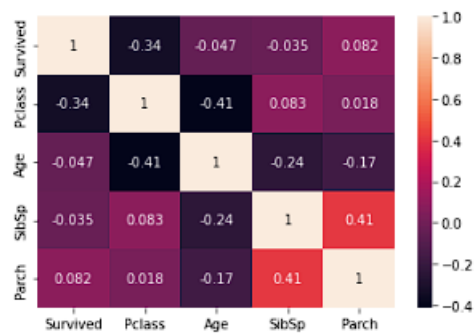
	Survived	
	0	1
Sex		
female	81	233
male	468	109

Oldest Person on Board: 80.0
 Youngest Person on Board: 0.42
 Average age of Person on Board: 29.100078563411895



	Survived	Pclass	Age	SibSp	Parch
Survived	1.000000	-0.338481	-0.046746	-0.035322	0.081629
Pclass	-0.338481	1.000000	-0.411805	0.083081	0.018443
Age	-0.046746	-0.411805	1.000000	-0.243877	-0.171917
SibSp	-0.035322	0.083081	-0.243877	1.000000	0.414838
Parch	0.081629	0.018443	-0.171917	0.414838	1.000000

<AxesSubplot:>



```
100
[[ 9.09621765]
 [14.63742853]
 [12.25580785]
 [ 7.21515957]
 [ 6.90562848]
 [12.42799856]
 [ 6.53450315]
 [12.36358975]
 [11.45101022]
 [ 9.29527704]
 [ 8.46897323]
 [11.11359701]
 [ 4.21646281]
 [ 8.92109838]
 [13.29785748]
 [15.47570863]
 [ 9.84113925]
 [17.99332461]
```

```

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

```

Practical no 5

Aim: Exploratory data analysis in python using titanic dataset

The following figure illustrates simple linear regression:

The package scikit-learn is a widely used Python library for machine learning, built on top of NumPy and some other packages, It provides the means for preprocessing data, reducing dimensionality, implementing regression, classification, clustering, and more. Like NumPy, scikit-learn is also open source.

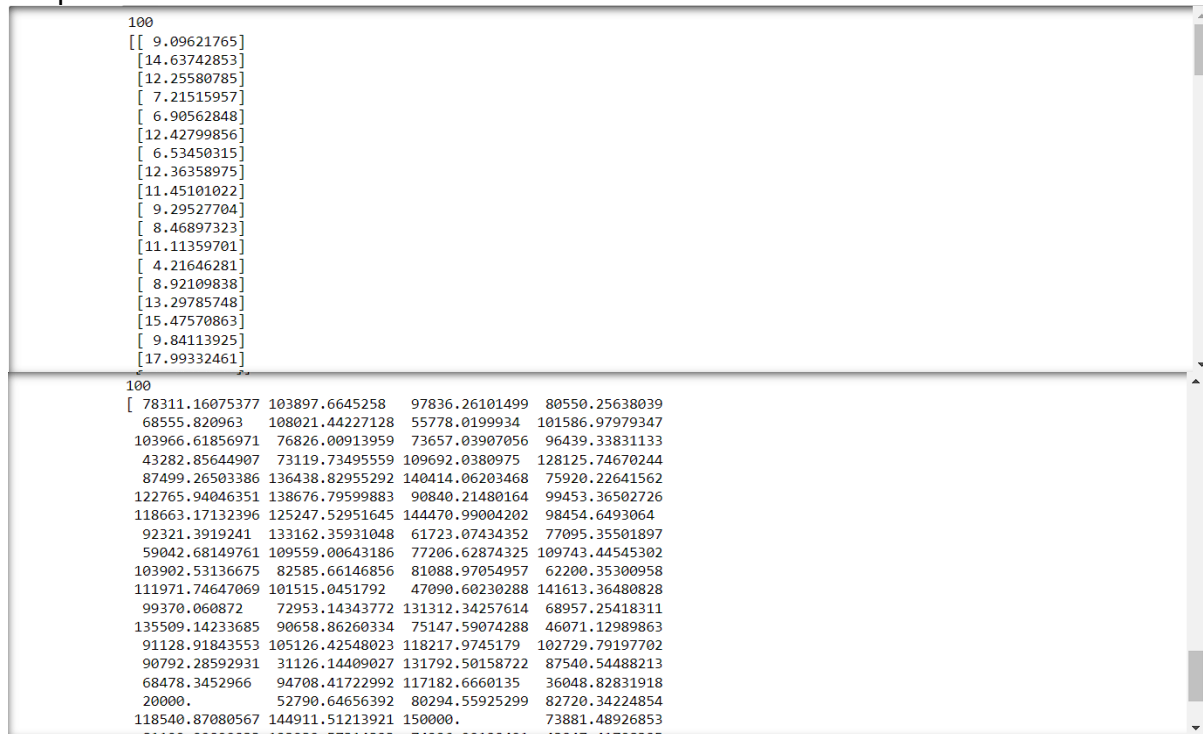
It is used as sklearn in python

- 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:

```
1 import numpy as np
2 from sklearn import datasets
3 x,y,coef=datasets.make_regression(n_samples=100, n_features=1, n_informative=1, noise=10,coef=True, random_state = 0)
4 x=np.interp(x,(x.min(),x.max()),(0,20))
5 print(len(x))
6 print(x)
7 y=np.interp(y,(y.min(),y.max()),(20000,150000))
8 print(len(y))
9 print(y)
```

Output:



```
100
[[ 9.09621765]
 [14.63742853]
 [12.25580785]
 [ 7.21515957]
 [ 6.90562848]
 [12.42799856]
 [ 6.53450315]
 [12.36358975]
 [11.45101022]
 [ 9.29527704]
 [ 8.46897323]
 [11.11359701]
 [ 4.21646281]
 [ 8.92109838]
 [13.29785748]
 [15.47570863]
 [ 9.84113925]
 [17.99332461]]

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
91100.00000000 122030.57314322  74336.00330401  42047.41708235]
```

Code:

```
1 import matplotlib.pyplot as plt
2 plt.plot(x,y,'.',label="training data")
3 plt.xlabel("Years of Experience")
4 plt.ylabel("Salary")
5 plt.title("Experience vs Salary")
```

Output:

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

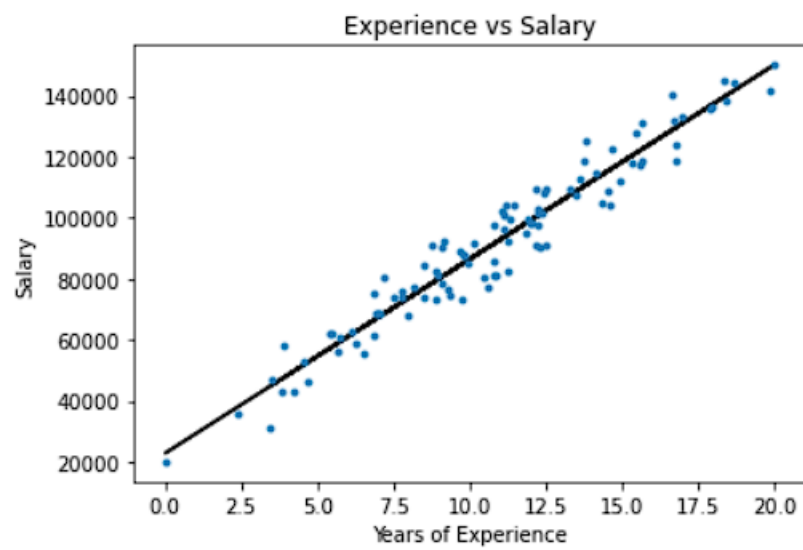


Code:

```
1 from sklearn.linear_model import LinearRegression
2 reg_model = LinearRegression()
3 reg_model.fit(x,y)
4 y_pred=reg_model.predict(x)
5 plt.plot(x,y_pred,color="black")
6 plt.plot(x,y,'.',label="training data")
7 plt.xlabel("Years of Experience")
8 plt.ylabel("Salary")
9 plt.title("Experience vs Salary")
```

Output:

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



Code:

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

Output:

	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

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:


```

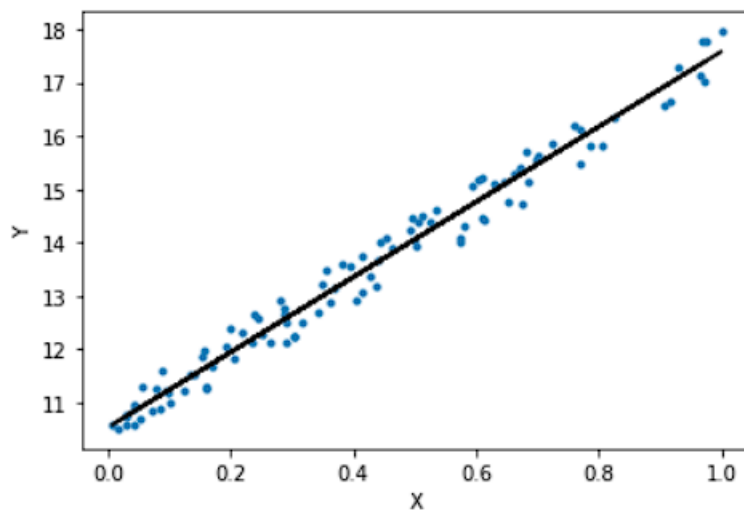
1 x1=[[13.0]]
2 y1=reg_model.predict(x1)
3 print(np.round(y1))
4 reg_model1=LinearRegression()
5 x=np.random.rand(100,1)
6 yintercept=10
7 slope=7
8 error=np.random.rand(100,1)
9 y=yintercept+slope*x+error
10 reg_model1.fit(x,y)
11 y_pred=reg_model1.predict(x)
12 plt.scatter(x,y,s=10)
13 plt.xlabel("X")
14 plt.ylabel("Y")
15 plt.plot(x,y_pred,color="black")

```

Output:

[105534.]

[<matplotlib.lines.Line2D at 0x169f579a940>]



Practical no 6

Aim: Write a python program to implement multiple linear regression on the dataset

Boston.csv

The dataset provides Housing Values in Suburbs of Boston

The medv(Price) variable is the target /dependent variable.

Data description

The Boston data frame has 506 rows and 14 columns.

This data frame contains the following columns:

crim

per capita crime rate by town.

zn

proportion of residential land zoned for lots over 25,000 sq.ft.

indus

proportion of non-retail business acres per town.

chas

Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).

nox

nitrogen oxides concentration (parts per 10 million).

rm

average number of rooms per dwelling.

age

proportion of owner-occupied units built prior to 1940.

dis

weighted mean of distances to five Boston employment centres.

rad

index of accessibility to radial highways.

tax

full-value property-tax rate per \ \$10,000.

ptratio

pupil-teacher ratio by town.

black

$1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town.

lstat

lower status of the population (percent).

Medv(Price)

median value of owner-occupied homes in \ \$1000s.

Code:

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import sklearn
4 boston = pd.read_csv("Boston.csv")
5 boston.head()
6
7 boston.info()
8
9 boston = boston.drop(columns="Unnamed: 0")
10 boston.info()
11
12 boston_x = pd.DataFrame(boston.iloc[:, :13])
13 boston_x.head()
14
15 boston_y = pd.DataFrame(boston.iloc[:, -1])
16 boston_y.head()
17
18 from sklearn.model_selection import train_test_split
19 X_train, X_test, Y_train, Y_test = train_test_split(boston_x, boston_y, test_size=0.3)
20 print("xtrain shape", X_train.shape)
21 print("ytrain shape", Y_train.shape)
22
23 print("xtest shape", X_test.shape)
24 print("ytest shape", Y_test.shape)
25
26 plt.scatter(Y_test, Y_pred_linear, c="green")
27 plt.xlabel("Actual Price(medv)")
28 plt.ylabel("Predicted Price(medv)")
29 plt.title("Actual vs Prediction")
30 plt.show()

```

Output:

	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

```

<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

```

```

<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

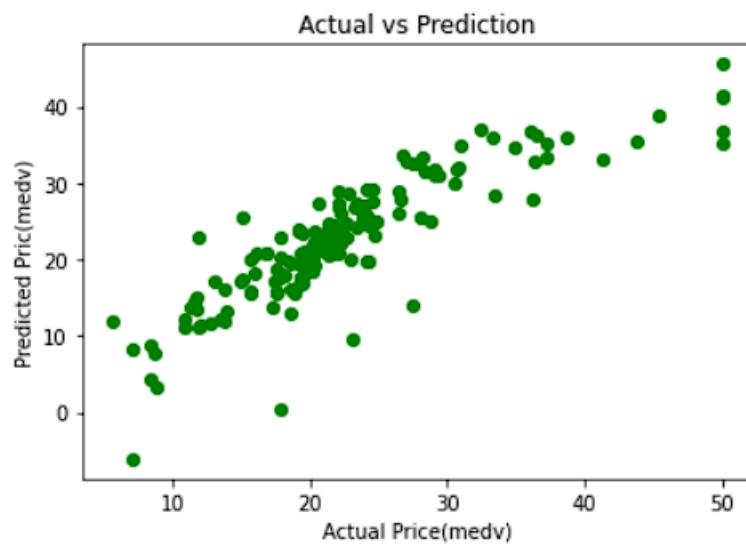
```

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

medv	
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2

xtrain shape (354, 13)
ytrain shape (354, 1)

xtest shape (152, 13)
ytest shape (152, 1)



Practical no 7

Aim: Write a python program to implement KNN algorithm to predict breast cancer using breast cancer Wisconsin dataset

K Nearest Neighbor classification Algorithm

Data Set Information:

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

Attribute Information:

1) ID number
2) Diagnosis (M = malignant, B = benign)
(3-32)

Ten real-valued features are computed for each cell nucleus:

- a) radius (mean of distances from center to points on the perimeter)
- b) texture (standard deviation of gray-scale values)
- c) perimeter
- d) area
- e) smoothness (local variation in radius lengths)
- f) compactness ($\text{perimeter}^2 / \text{area} - 1.0$)
- g) concavity (severity of concave portions of the contour)
- h) concave points (number of concave portions of the contour)
- i) symmetry
- j) fractal dimension ("coastline approximation" - 1)

Code:

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from sklearn.datasets import load_breast_cancer
5 from sklearn.metrics import confusion_matrix
6 from sklearn.neighbors import KNeighborsClassifier
7 from sklearn.model_selection import train_test_split
8 import seaborn as sns
9
10
11 breast_cancer_df = load_breast_cancer()
12 x=pd.DataFrame(breast_cancer_df.data,columns=breast_cancer_df.feature_names)
13 x.head()
14
15
16 columns_to_select = ["mean area", "mean compactness"]
17 x=x[columns_to_select]
18 x.head()
19
20
21 y=pd.Categorical.from_codes(breast_cancer_df.target,breast_cancer_df.target_names)
22 print(y)
23
24
25 y=pd.get_dummies(y,drop_first=True)
26 print(y)
27
28
29 X_train, X_test, Y_train, Y_test = train_test_split(x, y, random_state=1)
30 knn = KNeighborsClassifier(n_neighbors=5, metric="euclidean")
31 knn.fit(X_train, Y_train)
32 sns.set()
33 sns.scatterplot(x="mean area", y="mean compactness", hue="benign", data=X_test.join(Y_test, how="outer"))
34
35
36 cf=confusion_matrix(Y_test,y_pred)
37 print(cf)
38
39
40 y_pred=knn.predict(X_test)
41 plt.scatter(X_test["mean area"],X_test["mean compactness"],c=y_pred,cmap="coolwarm",alpha=0.7)
42
43
44 labels=["True Negative","False Positive","False Negative","True Positive"]
45 labels=np.asarray(labels).reshape(2,2)
46 categories=["Zero","One"]
47 ax=plt.subplot()
48 sns.heatmap(cf,annot=True,ax=ax)
49 ax.set_xlabel("Predicted Values")
50 ax.set_ylabel("Actual Values")
51 ax.set_title("Confusion Matrix")
52 ax.xaxis.set_ticklabels(["Malignant","Benign"])
53 ax.yaxis.set_ticklabels(["Malignant","Benign"])

```

Output:

	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



	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

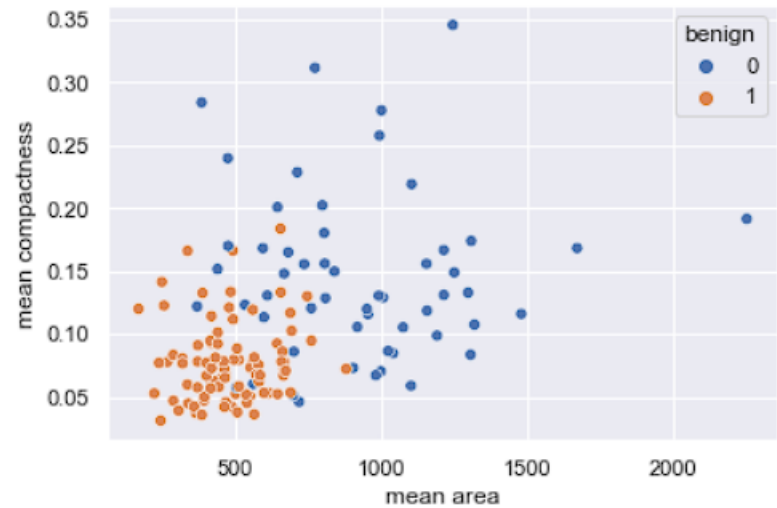
```
['malignant', 'malignant', 'malignant', 'malignant', 'malignant', ..., 'malignant', 'malignant', 'malignant', 'malignant', 'ben
ign']
Length: 569
Categories (2, object): ['malignant', 'benign']
```

```

      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]
```

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

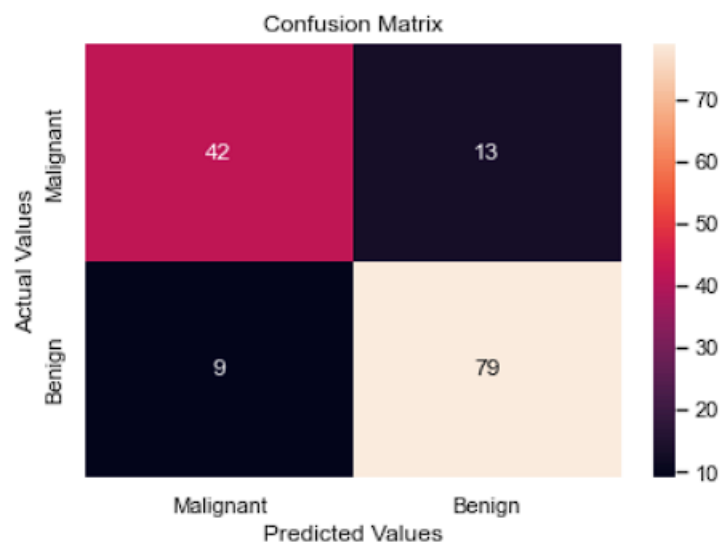


```
[[42 13]
 [ 9 79]]
```

```
<matplotlib.collections.PathCollection at 0x169f7e3e760>
```



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



Practical no 8

Aim: Introduction to NOSQL using MongoDB

1. Create a database Company, 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
- Sort the documents in descending order of Salary
- Display employee with designation with "Manager" or salary greater than Rs. 50,000/-.
- Update the salary of all employees with designation as "Accountant" to Rs.45000.
- Remove the documents of employees whose salary is greater than Rs100000.

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.
- Sort the documents in descending order of TotalMarks.
- Display students of class "MSc" or marks greater than 400.
- Remove all the documents with TotalMarks<200

Code & Output:

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

Use Company

Db.createCollection("Staff")

```
test> use Company
switched to db Company
Company> db.createCollection("Staff")
{ ok: 1 }
```

Db

Company

db.Staff.insertMany([

... { empid: 1, empname: "John Doe", salary: 60000, designation: "Manager" },

... { empid: 2, empname: "Jane Smith", salary: 55000, designation: "Accountant" },

```

... { empid: 3, empname: "Bob Johnson", salary: 70000, designation: "Engineer" },
... { empid: 4, empname: "Alice Williams", salary: 80000, designation: "Manager" },
... { empid: 5, empname: "Charlie Brown", salary: 45000, designation: "Salesperson" },
... { empid: 6, empname: "Emma Davis", salary: 90000, designation: "Accountant" },
... { empid: 7, empname: "David Wilson", salary: 75000, designation: "Engineer" },
... { empid: 8, empname: "Sophia Miller", salary: 60000, designation: "Manager" },
... { empid: 9, empname: "Daniel Smith", salary: 120000, designation: "CEO" },
... { empid: 10, empname: "Olivia Brown", salary: 48000, designation: "Salesperson" }
... ]);

```

```

Company> db.Staff.insertMany([
...   { empid: 1, empname: "John Doe", salary: 60000, designation: "Manager" },
...   { empid: 2, empname: "Jane Smith", salary: 55000, designation: "Accountant" },
...   { empid: 3, empname: "Bob Johnson", salary: 70000, designation: "Engineer" },
...   { empid: 4, empname: "Alice Williams", salary: 80000, designation: "Manager" },
...   { empid: 5, empname: "Charlie Brown", salary: 45000, designation: "Salesperson" },
...   { empid: 6, empname: "Emma Davis", salary: 90000, designation: "Accountant" },
...   { empid: 7, empname: "David Wilson", salary: 75000, designation: "Engineer" },
...   { empid: 8, empname: "Sophia Miller", salary: 60000, designation: "Manager" },
...   { empid: 9, empname: "Daniel Smith", salary: 120000, designation: "CEO" },
...   { empid: 10, empname: "Olivia Brown", salary: 48000, designation: "Salesperson" }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65f19bd48679f2082505e134'),
    '1': ObjectId('65f19bd48679f2082505e135'),
    '2': ObjectId('65f19bd48679f2082505e136'),
    '3': ObjectId('65f19bd48679f2082505e137'),
    '4': ObjectId('65f19bd48679f2082505e138'),
    '5': ObjectId('65f19bd48679f2082505e139'),
    '6': ObjectId('65f19bd48679f2082505e13a'),
    '7': ObjectId('65f19bd48679f2082505e13b'),
    '8': ObjectId('65f19bd48679f2082505e13c'),
    '9': ObjectId('65f19bd48679f2082505e13d')
  }
}
Company>

```

Db.Staff.find().pretty()

```
Company> db.Staff.find().pretty()
[
  {
    _id: ObjectId('65f19bd48679f2082505e134'),
    empid: 1,
    empname: 'John Doe',
    salary: 60000,
    designation: 'Manager'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e135'),
    empid: 2,
    empname: 'Jane Smith',
    salary: 55000,
    designation: 'Accountant'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e136'),
    empid: 3,
    empname: 'Bob Johnson',
    salary: 70000,
    designation: 'Engineer'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e137'),
    empid: 4,
    empname: 'Alice Williams',
    salary: 80000,
    designation: 'Manager'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e138'),
    empid: 5,
    empname: 'Charlie Brown',
    salary: 45000,
    designation: 'Salesperson'
  },
]
```

```
{
  _id: ObjectId('65f19bd48679f2082505e139'),
  empid: 6,
  empname: 'Emma Davis',
  salary: 90000,
  designation: 'Accountant'
},
{
  _id: ObjectId('65f19bd48679f2082505e13a'),
  empid: 7,
  empname: 'David Wilson',
  salary: 75000,
  designation: 'Engineer'
},
{
  _id: ObjectId('65f19bd48679f2082505e13b'),
  empid: 8,
  empname: 'Sophia Miller',
  salary: 60000,
  designation: 'Manager'
},
{
  _id: ObjectId('65f19bd48679f2082505e13c'),
  empid: 9,
  empname: 'Daniel Smith',
  salary: 120000,
  designation: 'CEO'
},
{
  _id: ObjectId('65f19bd48679f2082505e13d'),
  empid: 10,
  empname: 'Olivia Brown',
  salary: 48000,
  designation: 'Salesperson'
}
]
Company>
```

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

Db.Staff.find({});

```
Company> db.Staff.find({});
[
  {
    _id: ObjectId('65f19bd48679f2082505e134'),
    empid: 1,
    empname: 'John Doe',
    salary: 60000,
    designation: 'Manager'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e135'),
    empid: 2,
    empname: 'Jane Smith',
    salary: 55000,
    designation: 'Accountant'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e136'),
    empid: 3,
    empname: 'Bob Johnson',
    salary: 70000,
    designation: 'Engineer'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e137'),
    empid: 4,
    empname: 'Alice Williams',
    salary: 80000,
    designation: 'Manager'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e138'),
    empid: 5,
    empname: 'Charlie Brown',
    salary: 45000,
    designation: 'Salesperson'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e139'),
    empid: 6,
    empname: 'Emma Davis',
    salary: 90000,
    designation: 'Accountant'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e13a'),
    empid: 7,
    empname: 'David Wilson',
    salary: 75000,
    designation: 'Engineer'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e13b'),
    empid: 8,
    empname: 'Sophia Miller',
    salary: 60000,
    designation: 'Manager'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e13c'),
    empid: 9,
    empname: 'Daniel Smith',
    salary: 120000,
    designation: 'CEO'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e13d'),
    empid: 10,
    empname: 'Olivia Brown',
    salary: 48000,
    designation: 'Salesperson'
  }
]
Company>
```

Db.Staff.find({}, { empid: 1, designation: 1, _id: 0});

```
Company> db.Staff.find({}, { empid: 1, designation: 1, _id: 0 });
[
  { empid: 1, designation: 'Manager' },
  { empid: 2, designation: 'Accountant' },
  { empid: 3, designation: 'Engineer' },
  { empid: 4, designation: 'Manager' },
  { empid: 5, designation: 'Salesperson' },
  { empid: 6, designation: 'Accountant' },
  { empid: 7, designation: 'Engineer' },
  { empid: 8, designation: 'Manager' },
  { empid: 9, designation: 'CEO' },
  { empid: 10, designation: 'Salesperson' }
]
```

- Sort the documents in descending order of Salary

```
Db.Staff.find({}).sort({ salary: -1 });
```

```
Company> db.Staff.find({}).sort({ salary: -1 });
[
  {
    _id: ObjectId('65f19bd48679f2082505e13c'),
    empid: 9,
    empname: 'Daniel Smith',
    salary: 120000,
    designation: 'CEO'
  },
  {
    _id: ObjectId('65f19fe2c64b92a8fc7a0b80'),
    empid: 9,
    empname: 'Daniel Smith',
    salary: 120000,
    designation: 'CEO'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e139'),
    empid: 6,
    empname: 'Emma Davis',
    salary: 90000,
    designation: 'Accountant'
  },
  {
    _id: ObjectId('65f19fe2c64b92a8fc7a0b7d'),
    empid: 6,
    empname: 'Emma Davis',
    salary: 90000,
    designation: 'Accountant'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e137'),
    empid: 4,
    empname: 'Alice Williams',
    salary: 80000,
    designation: 'Manager'
  },
]
```



```
{
  _id: ObjectId('65f19fe2c64b92a8fc7a0b7b'),
  empid: 4,
  empname: 'Alice Williams',
  salary: 80000,
  designation: 'Manager'
},
{
  _id: ObjectId('65f19bd48679f2082505e13a'),
  empid: 7,
  empname: 'David Wilson',
  salary: 75000,
  designation: 'Engineer'
},
{
  _id: ObjectId('65f19fe2c64b92a8fc7a0b7e'),
  empid: 7,
  empname: 'David Wilson',
  salary: 75000,
  designation: 'Engineer'
},
{
  _id: ObjectId('65f19bd48679f2082505e136'),
  empid: 3,
  empname: 'Bob Johnson',
  salary: 70000,
  designation: 'Engineer'
},
{
  _id: ObjectId('65f19fe2c64b92a8fc7a0b7a'),
  empid: 3,
  empname: 'Bob Johnson',
  salary: 70000,
  designation: 'Engineer'
},
}
```

```

{
  _id: ObjectId('65f19fe2c64b92a8fc7a0b79'),
  empid: 2,
  empname: 'Jane Smith',
  salary: 55000,
  designation: 'Accountant'
},
{
  _id: ObjectId('65f19bd48679f2082505e13d'),
  empid: 10,
  empname: 'Olivia Brown',
  salary: 48000,
  designation: 'Salesperson'
},
{
  _id: ObjectId('65f19fe2c64b92a8fc7a0b81'),
  empid: 10,
  empname: 'Olivia Brown',
  salary: 48000,
  designation: 'Salesperson'
},
{
  _id: ObjectId('65f19bd48679f2082505e138'),
  empid: 5,
  empname: 'Charlie Brown',
  salary: 45000,
  designation: 'Salesperson'
},
{
  _id: ObjectId('65f19fe2c64b92a8fc7a0b7c'),
  empid: 5,
  empname: 'Charlie Brown',
  salary: 45000,
  designation: 'Salesperson'
}
]

```

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

```
Db.Staff.find ({ $or: [{ designation: "Manager" }, { salary: { $gt: 50000 } }] });
```

```
Company> db.Staff.find({ $or: [{ designation: "Manager" }, { salary: { $gt: 50000 } }] });
[
  {
    _id: ObjectId('65f19bd48679f2082505e134'),
    empid: 1,
    empname: 'John Doe',
    salary: 60000,
    designation: 'Manager'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e135'),
    empid: 2,
    empname: 'Jane Smith',
    salary: 55000,
    designation: 'Accountant'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e136'),
    empid: 3,
    empname: 'Bob Johnson',
    salary: 70000,
    designation: 'Engineer'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e137'),
    empid: 4,
    empname: 'Alice Williams',
    salary: 80000,
    designation: 'Manager'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e139'),
    empid: 6,
    empname: 'Emma Davis',
    salary: 90000,
    designation: 'Accountant'
  },
]
```

```

    {
      _id: ObjectId('65f19fe2c64b92a8fc7a0b7e'),
      empid: 7,
      empname: 'David Wilson',
      salary: 75000,
      designation: 'Engineer'
    },
    {
      _id: ObjectId('65f19fe2c64b92a8fc7a0b7f'),
      empid: 8,
      empname: 'Sophia Miller',
      salary: 60000,
      designation: 'Manager'
    },
    {
      _id: ObjectId('65f19fe2c64b92a8fc7a0b80'),
      empid: 9,
      empname: 'Daniel Smith',
      salary: 120000,
      designation: 'CEO'
    }
  ]

```

- Update the salary of all employees with designation as “Accountant” to Rs.45000.

```
Db.Staff.updateMany({ designation: "Accountant" }, { $set: { salary: 45000 } });
```

```

Company> db.Staff.updateMany({ designation: "Accountant" }, { $set: { salary: 45000 } });
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 0,
  upsertedCount: 0
}

```

```
Db.Staff.find({ designation: "Accountant" });
```

```

Company> db.Staff.find({ designation: "Accountant" });
[
  {
    _id: ObjectId('65f19bd48679f2082505e135'),
    empid: 2,
    empname: 'Jane Smith',
    salary: 45000,
    designation: 'Accountant'
  },
  {
    _id: ObjectId('65f19bd48679f2082505e139'),
    empid: 6,
    empname: 'Emma Davis',
    salary: 45000,
    designation: 'Accountant'
  },
  {
    _id: ObjectId('65f19fe2c64b92a8fc7a0b79'),
    empid: 2,
    empname: 'Jane Smith',
    salary: 45000,
    designation: 'Accountant'
  },
  {
    _id: ObjectId('65f19fe2c64b92a8fc7a0b7d'),
    empid: 6,
    empname: 'Emma Davis',
    salary: 45000,
    designation: 'Accountant'
  }
]

```

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

```
Db.Staff.deleteMany({ salary: { $gt: 100000 } });
```

```
Company> db.Staff.deleteMany({ salary: { $gt: 100000 } });  
{ acknowledged: true, deletedCount: 2 }
```

```
Db.Staff.find();
```

```
Company> db.Staff.find()  
[  
  {  
    _id: ObjectId('65f19bd48679f2082505e134'),  
    empid: 1,  
    empname: 'John Doe',  
    salary: 60000,  
    designation: 'Manager'  
  },  
  {  
    _id: ObjectId('65f19bd48679f2082505e135'),  
    empid: 2,  
    empname: 'Jane Smith',  
    salary: 45000,  
    designation: 'Accountant'  
  },  
  {  
    _id: ObjectId('65f19bd48679f2082505e136'),  
    empid: 3,  
    empname: 'Bob Johnson',  
    salary: 70000,  
    designation: 'Engineer'  
  },  
  {  
    _id: ObjectId('65f19bd48679f2082505e137'),  
    empid: 4,  
    empname: 'Alice Williams',  
    salary: 80000,  
    designation: 'Manager'  
  },  
  {  
    _id: ObjectId('65f19bd48679f2082505e138'),  
    empid: 5,  
    empname: 'Charlie Brown',  
    salary: 45000,  
    designation: 'Salesperson'  
  },  
]
```

```
{
  _id: ObjectId('65f19bd48679f2082505e139'),
  empid: 6,
  empname: 'Emma Davis',
  salary: 45000,
  designation: 'Accountant'
},
{
  _id: ObjectId('65f19bd48679f2082505e13a'),
  empid: 7,
  empname: 'David Wilson',
  salary: 75000,
  designation: 'Engineer'
},
{
  _id: ObjectId('65f19bd48679f2082505e13b'),
  empid: 8,
  empname: 'Sophia Miller',
  salary: 60000,
  designation: 'Manager'
},
{
  _id: ObjectId('65f19bd48679f2082505e13d'),
  empid: 10,
  empname: 'Olivia Brown',
  salary: 48000,
  designation: 'Salesperson'
},
```

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).

Use Institution;

Db.createCollection("Student")

```
> use Institution;
switched to db Institution
Institution> db.createCollection("Student")
{ ok: 1 }
```

db.Student.insertMany([

```
... { RollNo: 1, Name: "Alice", Class: "BSc", TotalMarks: 480 },
... { RollNo: 2, Name: "Bob", Class: "MSc", TotalMarks: 420 },
... { RollNo: 3, Name: "Charlie", Class: "BSc", TotalMarks: 350 },
... { RollNo: 4, Name: "David", Class: "MSc", TotalMarks: 480 },
... { RollNo: 5, Name: "Emma", Class: "BSc", TotalMarks: 300 },
... { RollNo: 6, Name: "Frank", Class: "MSc", TotalMarks: 450 },
... { RollNo: 7, Name: "Grace", Class: "BSc", TotalMarks: 420 },
... { RollNo: 8, Name: "Henry", Class: "MSc", TotalMarks: 400 },
... { RollNo: 9, Name: "Ivy", Class: "BSc", TotalMarks: 250 },
... { RollNo: 10, Name: "Jack", Class: "MSc", TotalMarks: 490 }
... ]);
```



```

Institution> db.Student.insertMany([
...   { RollNo: 1, Name: "Alice", Class: "BSc", TotalMarks: 480 },
...   { RollNo: 2, Name: "Bob", Class: "MSc", TotalMarks: 420 },
...   { RollNo: 3, Name: "Charlie", Class: "BSc", TotalMarks: 350 },
...   { RollNo: 4, Name: "David", Class: "MSc", TotalMarks: 480 },
...   { RollNo: 5, Name: "Emma", Class: "BSc", TotalMarks: 300 },
...   { RollNo: 6, Name: "Frank", Class: "MSc", TotalMarks: 450 },
...   { RollNo: 7, Name: "Grace", Class: "BSc", TotalMarks: 420 },
...   { RollNo: 8, Name: "Henry", Class: "MSc", TotalMarks: 400 },
...   { RollNo: 9, Name: "Ivy", Class: "BSc", TotalMarks: 250 },
...   { RollNo: 10, Name: "Jack", Class: "MSc", TotalMarks: 490 }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65f1a98a05a367268b7f06da'),
    '1': ObjectId('65f1a98a05a367268b7f06db'),
    '2': ObjectId('65f1a98a05a367268b7f06dc'),
    '3': ObjectId('65f1a98a05a367268b7f06dd'),
    '4': ObjectId('65f1a98a05a367268b7f06de'),
    '5': ObjectId('65f1a98a05a367268b7f06df'),
    '6': ObjectId('65f1a98a05a367268b7f06e0'),
    '7': ObjectId('65f1a98a05a367268b7f06e1'),
    '8': ObjectId('65f1a98a05a367268b7f06e2'),
    '9': ObjectId('65f1a98a05a367268b7f06e3')
  }
}

```

- Display all documents in Student.

Db.Student.find();

```
Institution> db.Student.find();
[
  {
    _id: ObjectId('65f1a98a05a367268b7f06da'),
    RollNo: 1,
    Name: 'Alice',
    Class: 'BSc',
    TotalMarks: 480
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06db'),
    RollNo: 2,
    Name: 'Bob',
    Class: 'MSc',
    TotalMarks: 420
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06dc'),
    RollNo: 3,
    Name: 'Charlie',
    Class: 'BSc',
    TotalMarks: 350
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06dd'),
    RollNo: 4,
    Name: 'David',
    Class: 'MSc',
    TotalMarks: 480
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06de'),
    RollNo: 5,
    Name: 'Emma',
    Class: 'BSc',
    TotalMarks: 300
  },
]
```

```
{
  {
    _id: ObjectId('65f1a98a05a367268b7f06df'),
    RollNo: 6,
    Name: 'Frank',
    Class: 'MSc',
    TotalMarks: 450
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06e0'),
    RollNo: 7,
    Name: 'Grace',
    Class: 'BSc',
    TotalMarks: 420
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06e1'),
    RollNo: 8,
    Name: 'Henry',
    Class: 'MSc',
    TotalMarks: 400
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06e2'),
    RollNo: 9,
    Name: 'Ivy',
    Class: 'BSc',
    TotalMarks: 250
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06e3'),
    RollNo: 10,
    Name: 'Jack',
    Class: 'MSc',
    TotalMarks: 490
  }
]
```

- Sort the documents in descending order of TotalMarks.

```
Db.Student.find({}).sort({ TotalMarks: -1 });
```

```
Institution> db.Student.find({}).sort({ TotalMarks: -1 });
[
  {
    _id: ObjectId('65f1a98a05a367268b7f06e3'),
    RollNo: 10,
    Name: 'Jack',
    Class: 'MSc',
    TotalMarks: 490
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06da'),
    RollNo: 1,
    Name: 'Alice',
    Class: 'BSc',
    TotalMarks: 480
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06dd'),
    RollNo: 4,
    Name: 'David',
    Class: 'MSc',
    TotalMarks: 480
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06df'),
    RollNo: 6,
    Name: 'Frank',
    Class: 'MSc',
    TotalMarks: 450
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06db'),
    RollNo: 2,
    Name: 'Bob',
    Class: 'MSc',
    TotalMarks: 420
  },
]
```

```
{
  _id: ObjectId('65f1a98a05a367268b7f06e0'),
  RollNo: 7,
  Name: 'Grace',
  Class: 'BSc',
  TotalMarks: 420
},
{
  _id: ObjectId('65f1a98a05a367268b7f06e1'),
  RollNo: 8,
  Name: 'Henry',
  Class: 'MSc',
  TotalMarks: 400
},
{
  _id: ObjectId('65f1a98a05a367268b7f06dc'),
  RollNo: 3,
  Name: 'Charlie',
  Class: 'BSc',
  TotalMarks: 350
},
{
  _id: ObjectId('65f1a98a05a367268b7f06de'),
  RollNo: 5,
  Name: 'Emma',
  Class: 'BSc',
  TotalMarks: 300
},
{
  _id: ObjectId('65f1a98a05a367268b7f06e2'),
  RollNo: 9,
  Name: 'Ivy',
  Class: 'BSc',
  TotalMarks: 250
}
]
```

```
Institution> db.Student.find({}).sort({ TotalMarks: -1 });
[
  {
    _id: ObjectId('65f1a98a05a367268b7f06e3'),
    RollNo: 10,
    Name: 'Jack',
    Class: 'MSc',
    TotalMarks: 490
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06da'),
    RollNo: 1,
    Name: 'Alice',
    Class: 'BSc',
    TotalMarks: 480
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06dd'),
    RollNo: 4,
    Name: 'David',
    Class: 'MSc',
    TotalMarks: 480
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06df'),
    RollNo: 6,
    Name: 'Frank',
    Class: 'MSc',
    TotalMarks: 450
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06db'),
    RollNo: 2,
    Name: 'Bob',
    Class: 'MSc',
    TotalMarks: 420
  },
]
```

```

{
  _id: ObjectId('65f1a98a05a367268b7f06e0'),
  RollNo: 7,
  Name: 'Grace',
  Class: 'BSc',
  TotalMarks: 420
},
{
  _id: ObjectId('65f1a98a05a367268b7f06e1'),
  RollNo: 8,
  Name: 'Henry',
  Class: 'MSc',
  TotalMarks: 400
},
{
  _id: ObjectId('65f1a98a05a367268b7f06dc'),
  RollNo: 3,
  Name: 'Charlie',
  Class: 'BSc',
  TotalMarks: 350
},
{
  _id: ObjectId('65f1a98a05a367268b7f06de'),
  RollNo: 5,
  Name: 'Emma',
  Class: 'BSc',
  TotalMarks: 300
},
{
  _id: ObjectId('65f1a98a05a367268b7f06e2'),
  RollNo: 9,
  Name: 'Ivy',
  Class: 'BSc',
  TotalMarks: 250
}
]

```

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

```
db.Student.find({ $or: [{ Class: "MSc" }, { TotalMarks: { $gt: 400 } }] });
```

```
Institution> db.Student.find({ $or: [{ Class: "MSc" }, { TotalMarks: { $gt: 400 } }] });
[
  {
    _id: ObjectId('65f1a98a05a367268b7f06da'),
    RollNo: 1,
    Name: 'Alice',
    Class: 'BSc',
    TotalMarks: 480
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06db'),
    RollNo: 2,
    Name: 'Bob',
    Class: 'MSc',
    TotalMarks: 420
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06dd'),
    RollNo: 4,
    Name: 'David',
    Class: 'MSc',
    TotalMarks: 480
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06df'),
    RollNo: 6,
    Name: 'Frank',
    Class: 'MSc',
    TotalMarks: 450
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06e0'),
    RollNo: 7,
    Name: 'Grace',
    Class: 'BSc',
    TotalMarks: 420
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06e1'),
    RollNo: 8,
    Name: 'Henry',
    Class: 'MSc',
    TotalMarks: 400
  },
  {
    _id: ObjectId('65f1a98a05a367268b7f06e3'),
    RollNo: 10,
    Name: 'Jack',
    Class: 'MSc',
    TotalMarks: 490
  }
]
```

- Remove all the documents with TotalMarks<200

```
Db.Student.deleteMany({ TotalMarks: { $lt: 200 } });
```



```
Institution> db.Student.deleteMany({ TotalMarks: { $lt: 200 } });  
{ acknowledged: true, deletedCount: 0 }
```