Sri Jayachamarajendra (Govt.) Polytechnic

Seshadri Road, Bengaluru-560001

Department of Computer Science

LAB MANUAL

Artificial Intelligence & Machine Learning

Course Code: 20CS51I

INDEX

Week-01

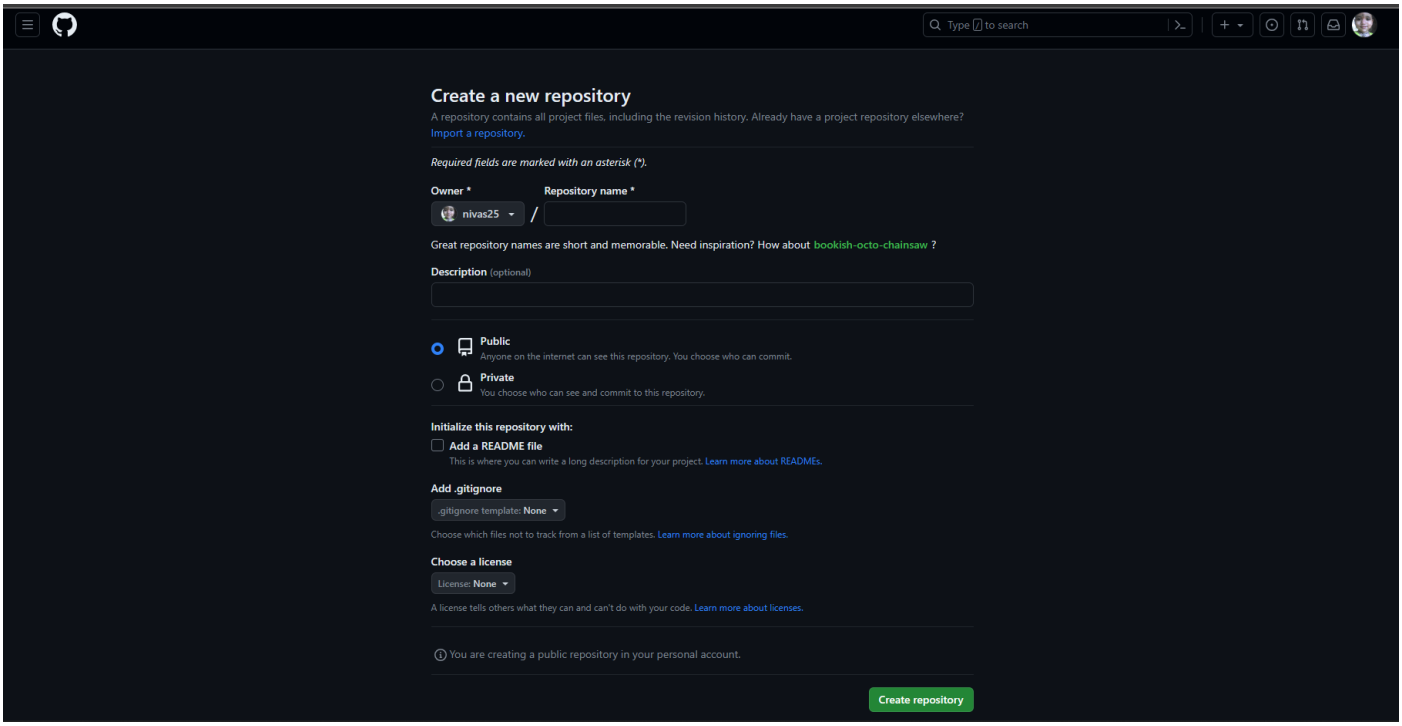
1.GitHub

Account creation and configuration:

- Go to the GitHub website: GitHub: Let's build from here · GitHub
- Click "Sign Up" in the upper-right corner.
- Choose a plan (usually "Free") and provide a unique username, email, and password.
- Verify your email address by clicking the link sent to your inbox.
- Complete your account setup by adding a profile picture and configuring account settings.

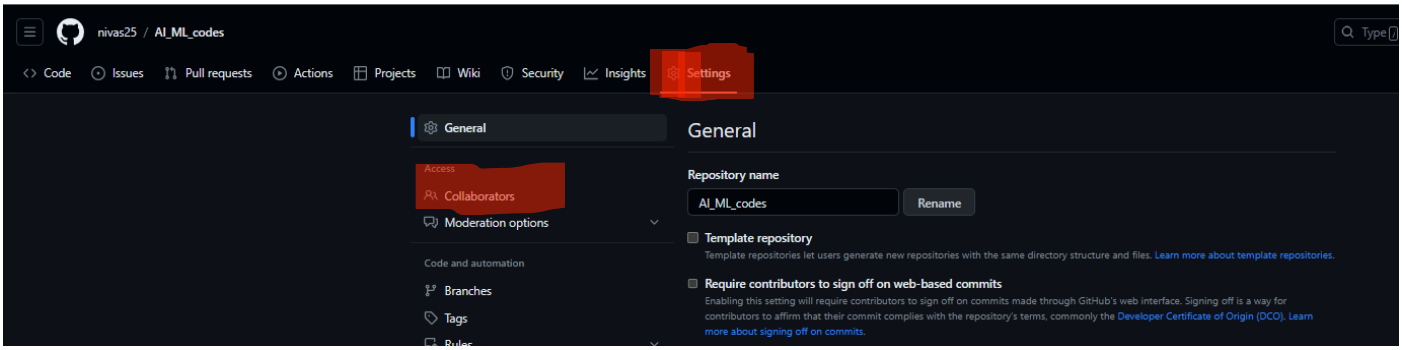
Create and push to repositories:

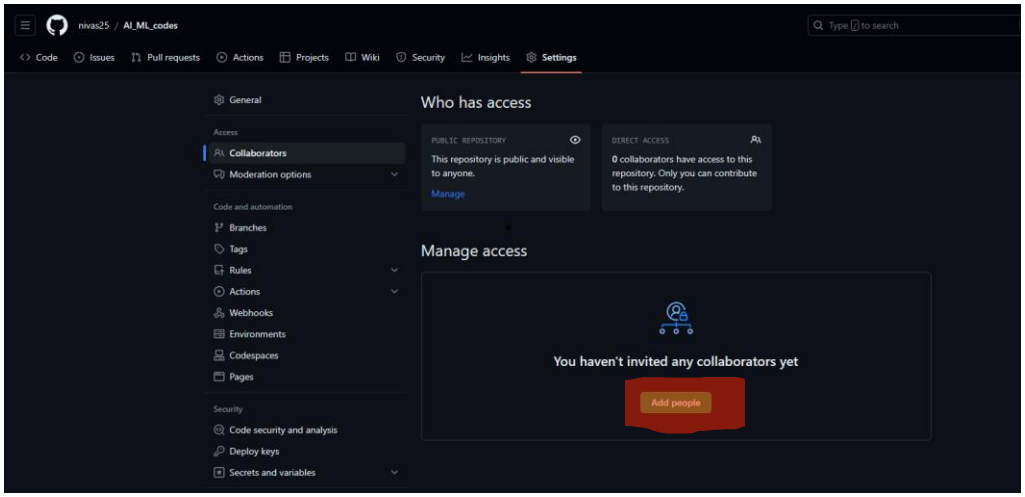
- Sign in to your GitHub account.
- Click on the "+" icon in the top-right corner and select "New repository."
- Enter a name for your repository, add an optional description, and choose the repository's visibility (public or private).



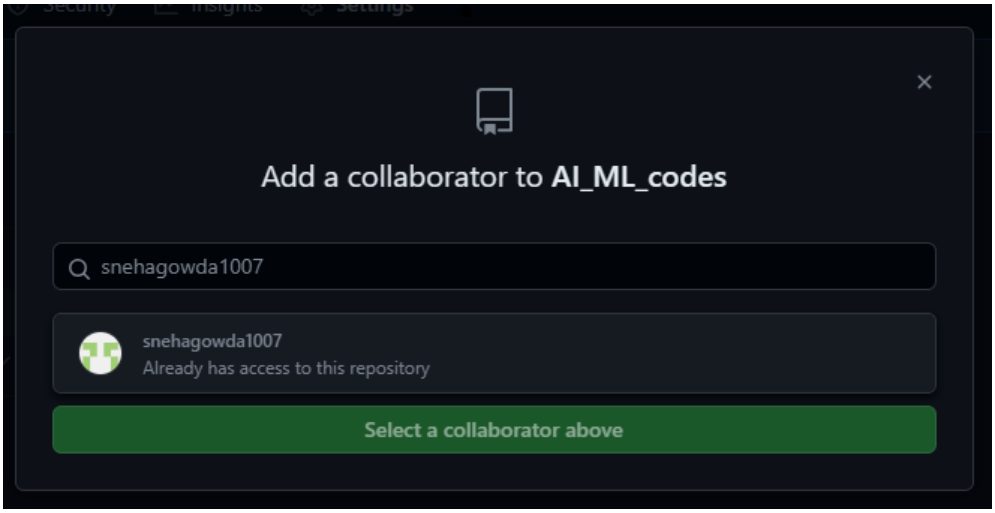
Collaboration:

- Open your GitHub repository.
- Go to "Settings."
- Select "Collaborators" from the left sidebar.

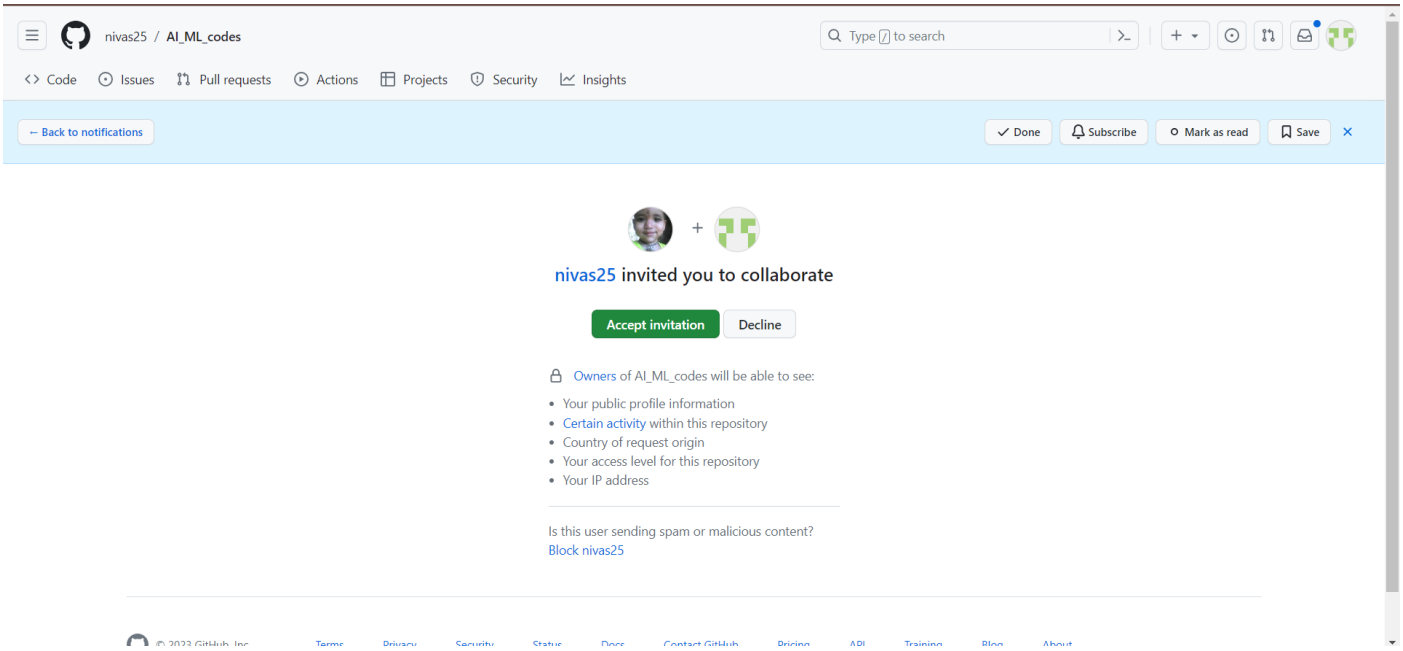




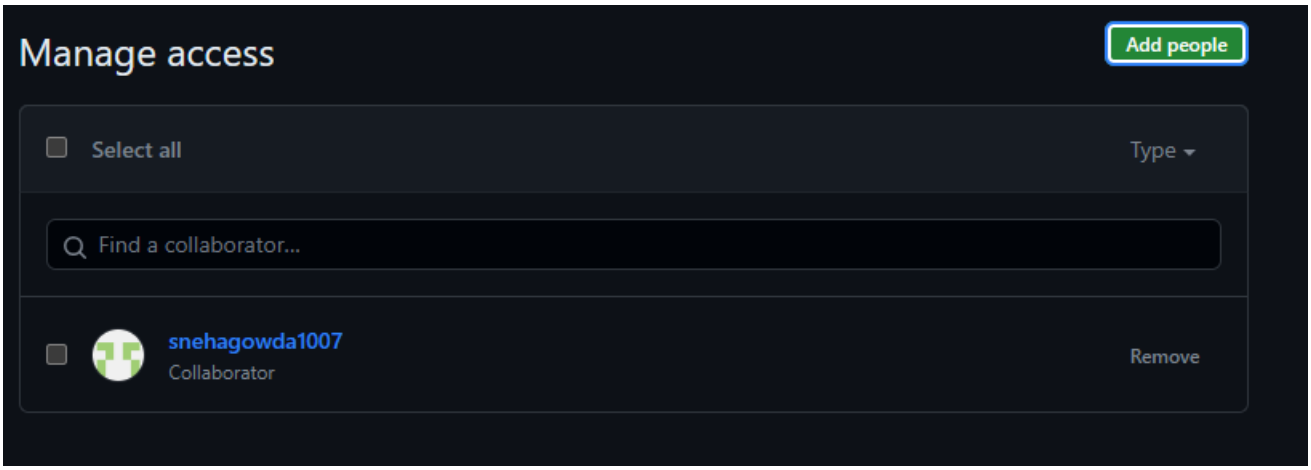
- Add the collaborator's username or email.
- Click "Add" to send the invite.



- Collaborator accepts the invite.



- That's all collaborator is added.



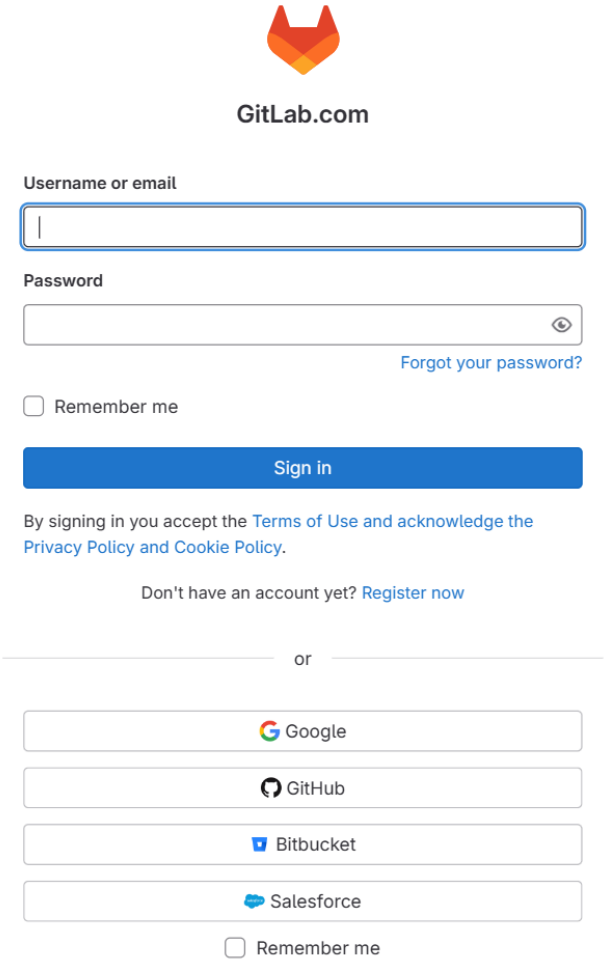
Migration

Migration, in a software development context, refers to the process of moving data, applications, or other resources from one environment, platform, system, or version to another.

- In this experiment we are moving an existing GitHub repository to new platform known as Gitlab

Creation of Gitlab account:

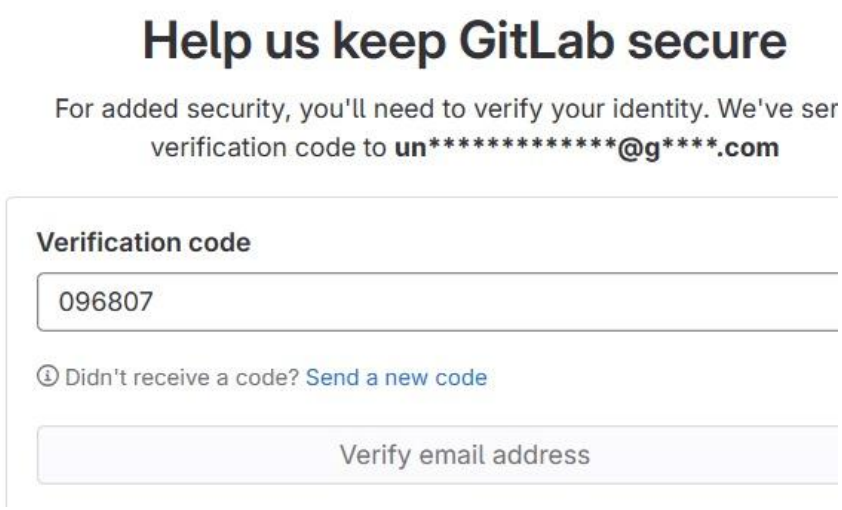
- Go to this website: Sign in · GitLab



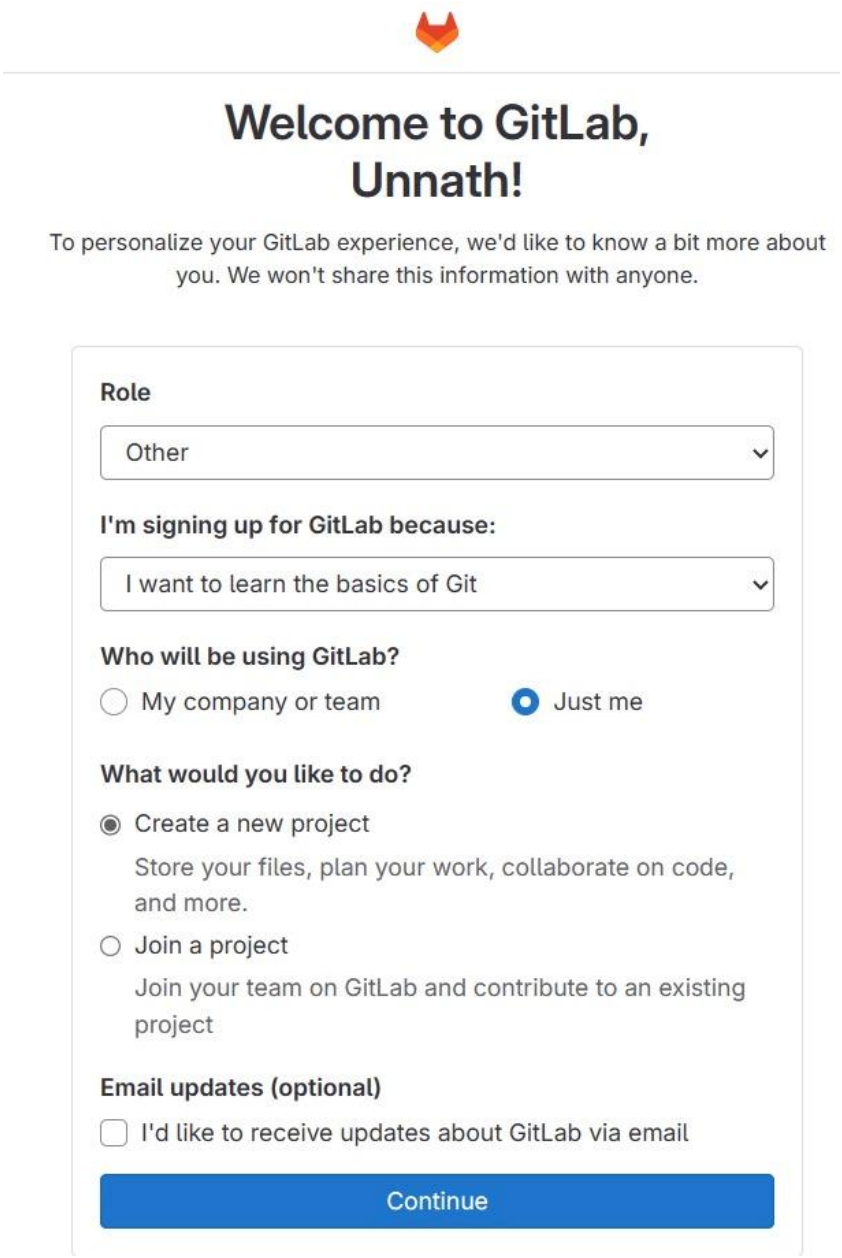
- Login using google account [Use the same account which is used in GitHub]

- **Agree to the Terms:**
  Read through GitLab's Terms of Service and Privacy Policy, and if you agree, check the box indicating your acceptance.

- **Verify Your Email:**
  GitLab will send a verification email to the address you provided during registration. Open your email inbox and look for the verification email. Click the link in the email to verify your email address. This step is important to ensure that you receive notifications and can reset your password if needed.

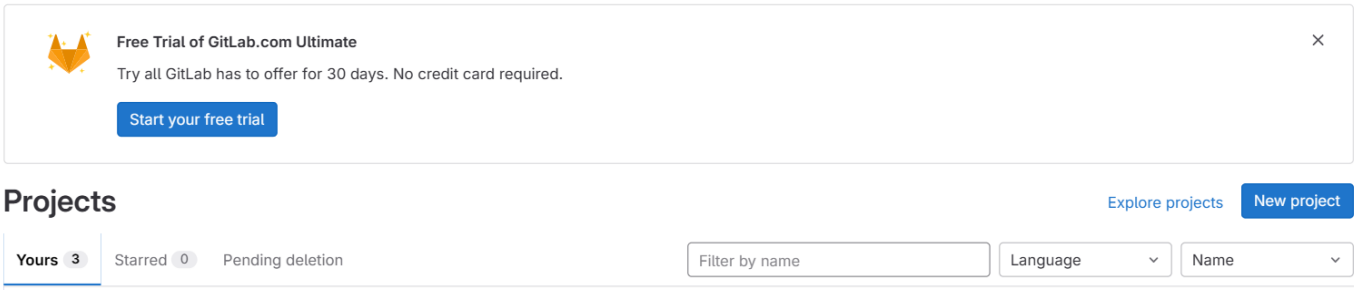


- **Set Up Your Profile:**

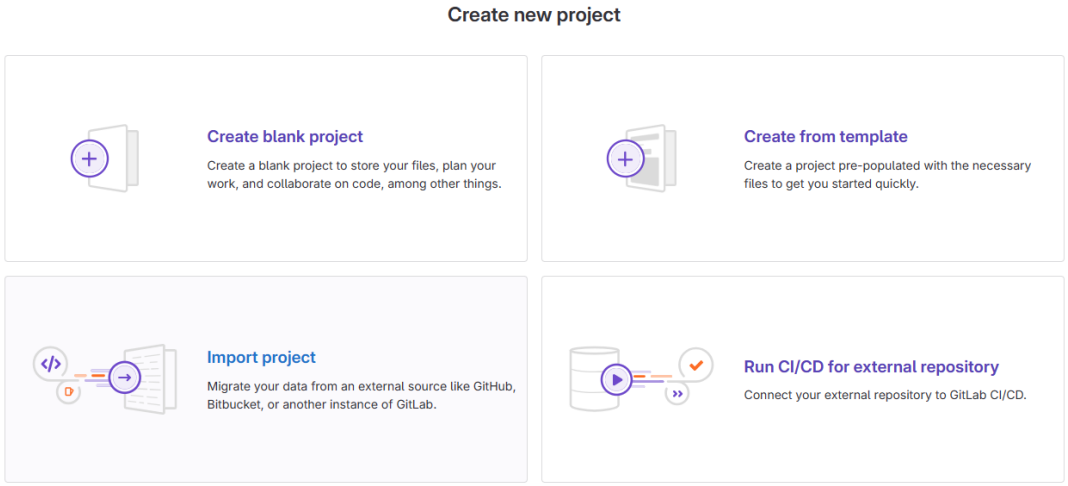


Importing Repository from GitHub to Gitlab

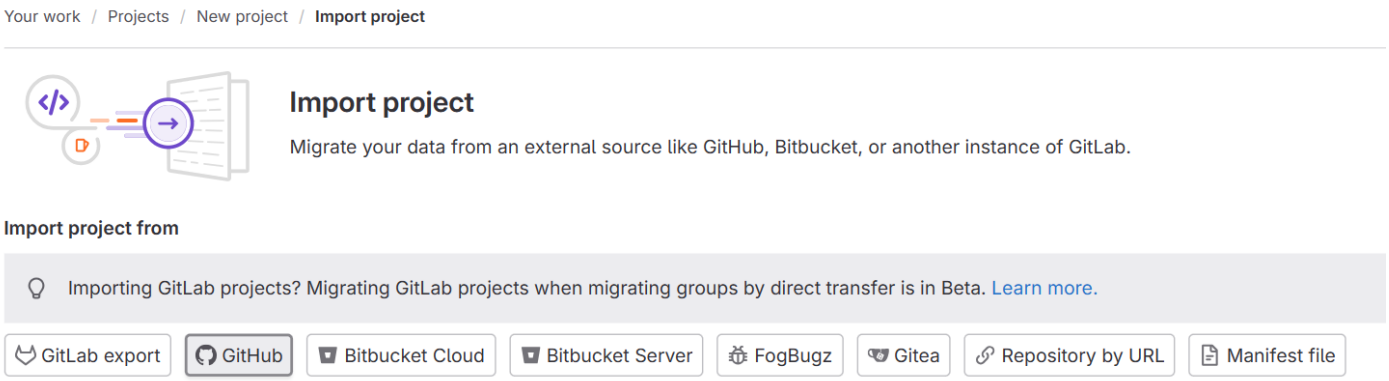
- Restart the browser and go to Projects · GitLab
- Click on new project



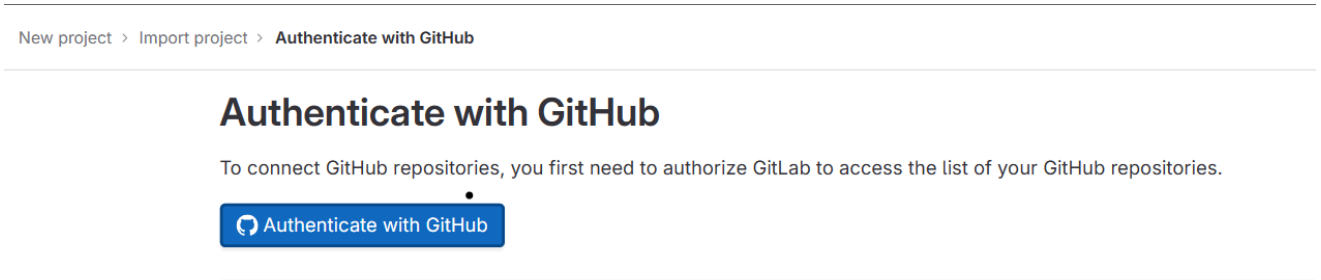
- Click on Import projects to import a repository from GitHub to Gitlab



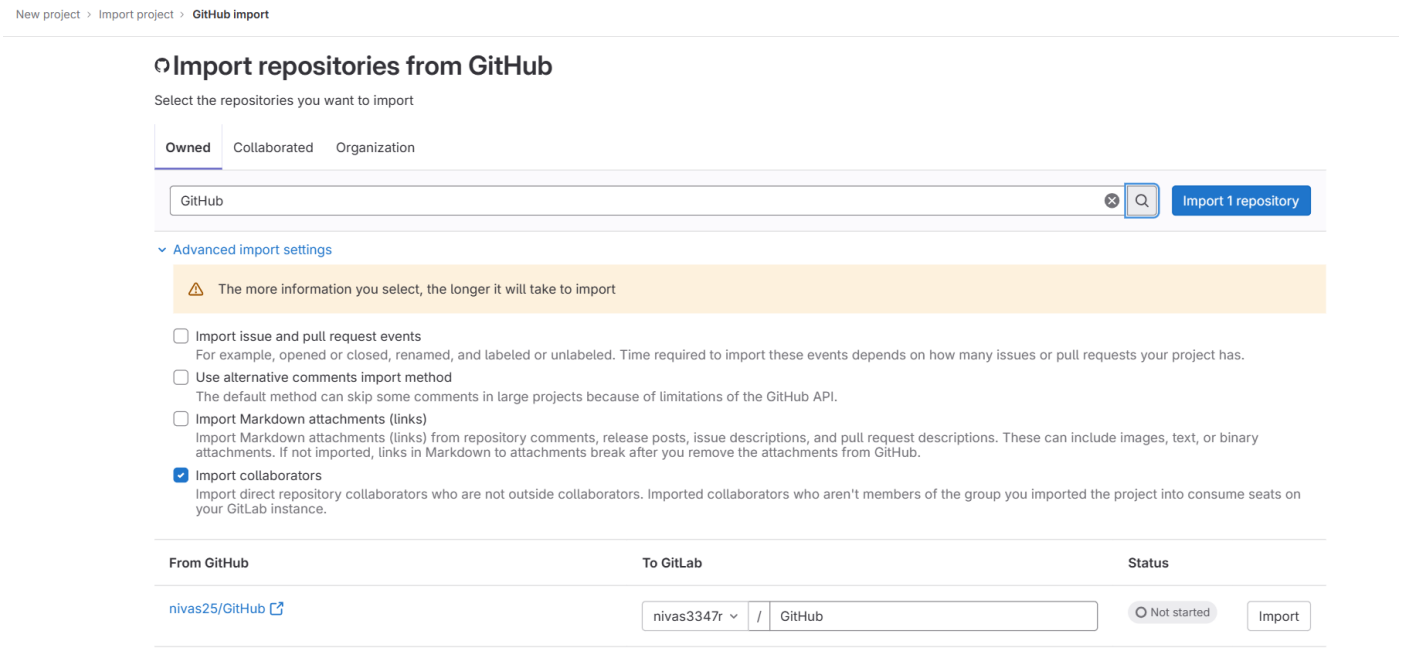
- Click on GitHub to connect your GitHub account to Gitlab



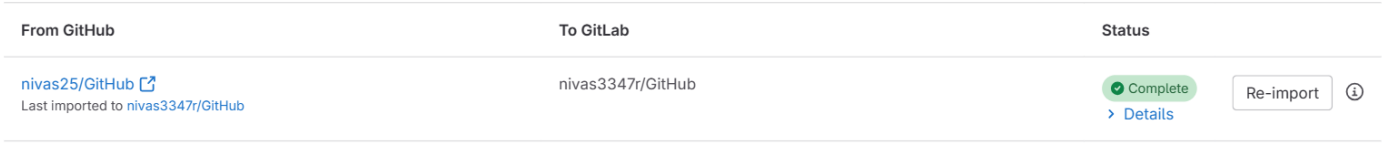
- Authenticate with GitHub



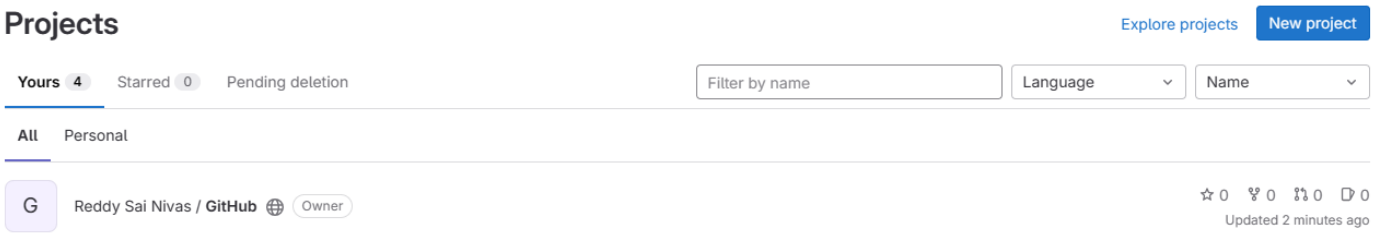
- If you're GitHub account email and Gitlab email is same it connects without any problem
- Type the repository name which u need to import and click on search icon



- Click on import repository.
- Wait for few seconds to complete the import process after that it shows completed if any errors don't occur.



- Check in projects to view the imported repository.



Video Sources:

nivas25/Git_GitHub_Videos

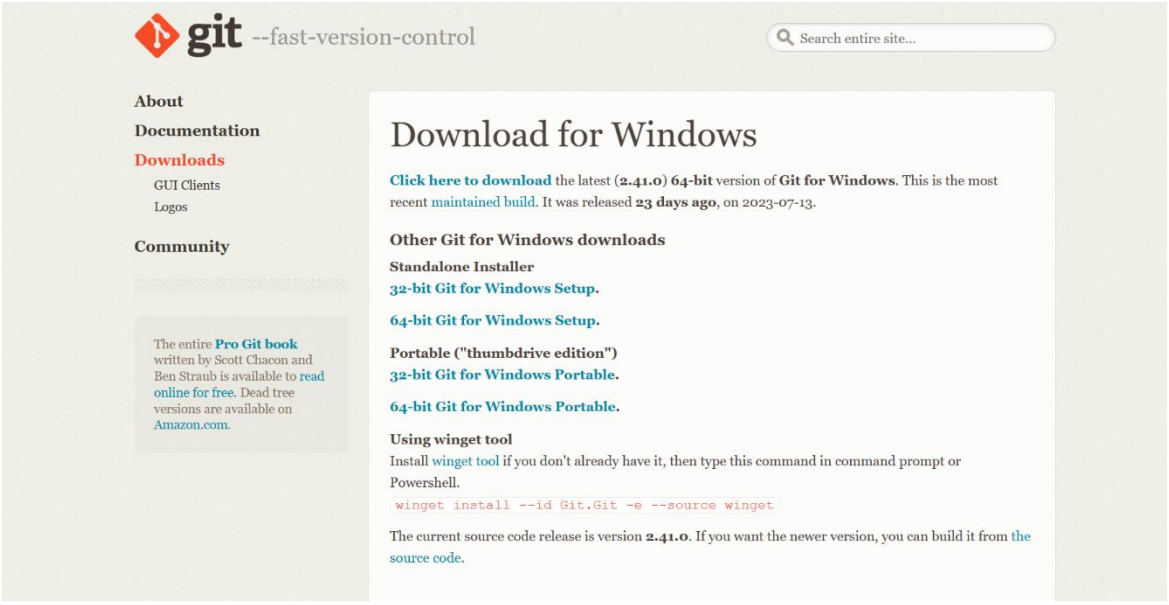
Code Links:

github.com

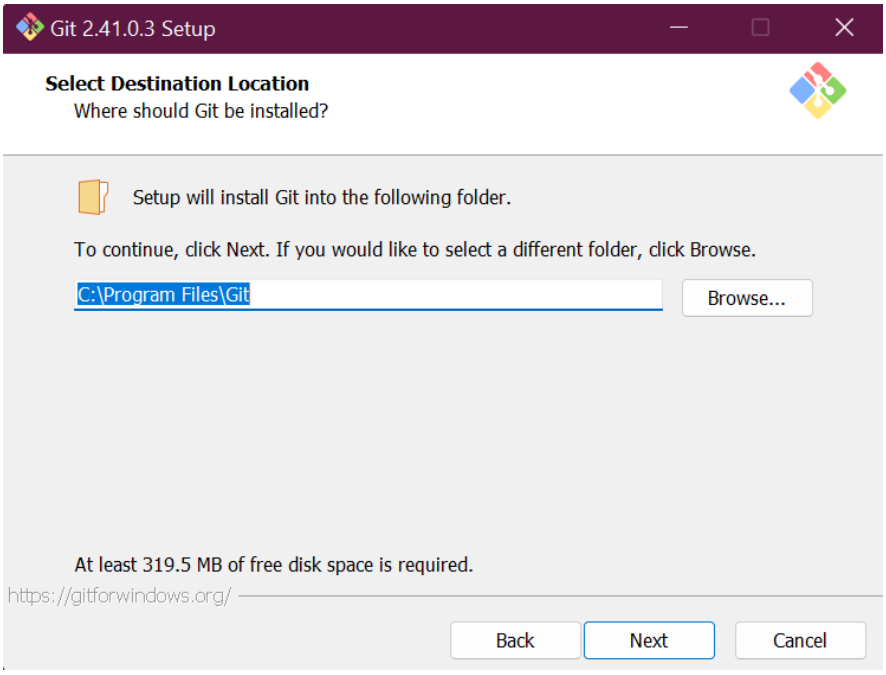
2.Git installation and setup

Installation:

- Download the Git installer for Windows from Git - Downloads (git-scm.com) .
- Download the 64bit Windows Setup.



- Open the git installer .exe file and start installation
- A git installer window opens and click on next and select installation destination if it doesn't show default installation location.



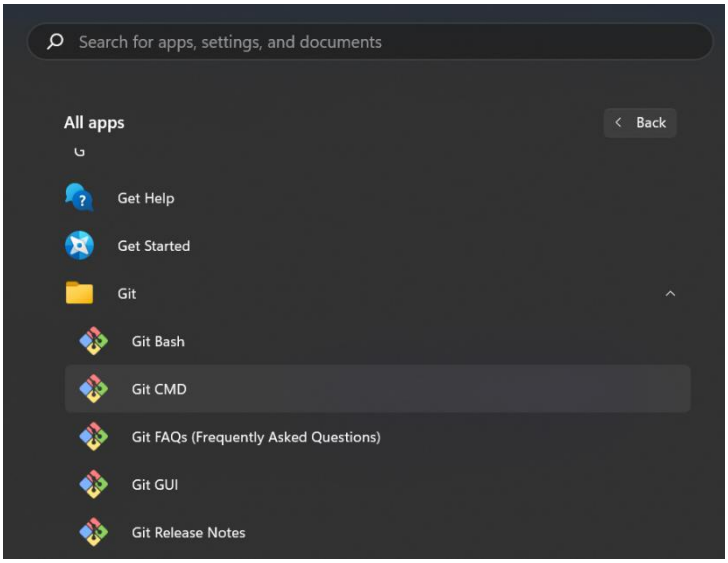
- Be clicking on next button until install button appears and click on "Install" to complete the installation.
- That's all GIT is ready for working!!

Visual Studio Code Installation:

- Download the VS code installer from Visual Studio Code - Code Editing. Redefined .
- Open the vs code installer .exe file and start installation
- Run the installer and follow the on-screen instructions with default settings.

Git Configuration:

- Open the Git CMD which will be there in git folder and it appears in Start



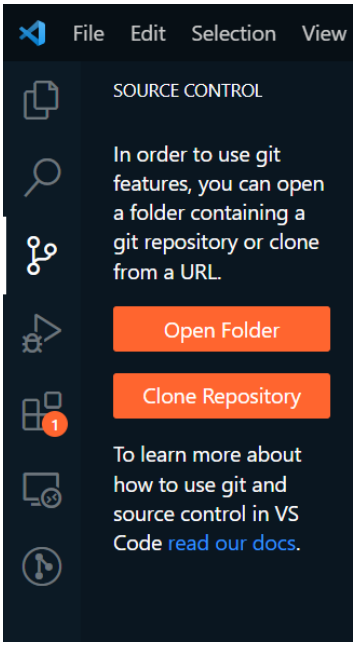
- Run the commands to config the git, enter a username and email id

git config --global user.name “yourname”

git config --global email “your_email_id”

```
C:\Users\reddy>git config --global user.name "Nivas"
C:\Users\reddy>git config --global user.email nivas3347@gmail.com
```

- If successfully configed the git , u can check it by opening VS code and by clicking on source control which is the left side of panel



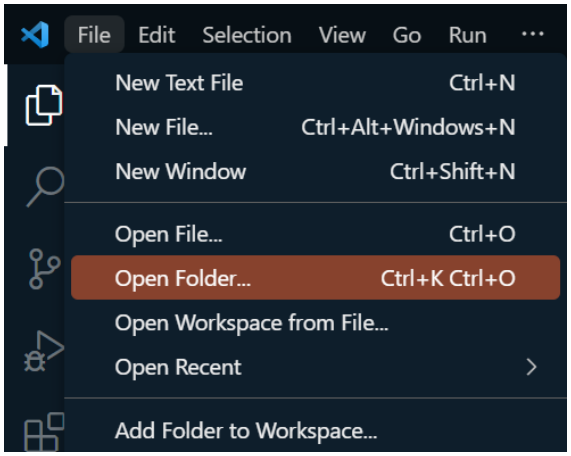
Basic local Git operations

- Local Git refers to the use of Git on your local machine, where you work directly with the Git version control system without the need for a remote repository or internet connection.

Creating a repository:

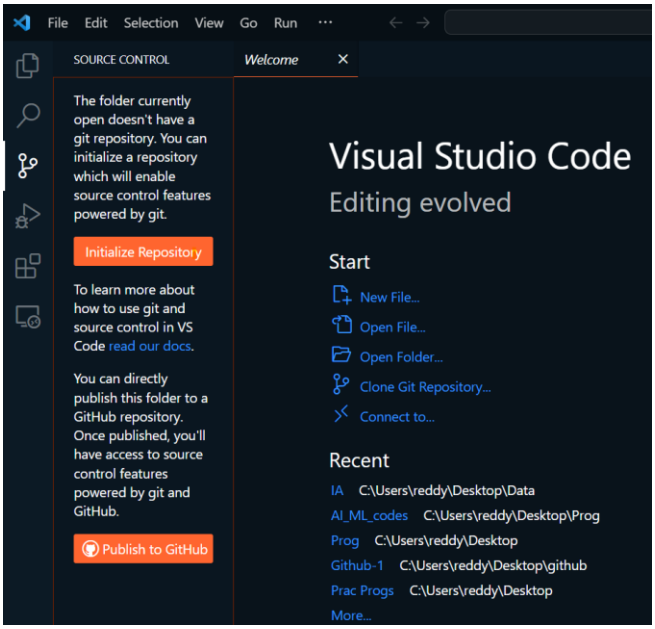
- A repository, often abbreviated as "repo," is a central place where version control systems like Git store all the files, data, and information related to a project.

- Create a Folder in desktop or any other location and name it. Open the VS code.
- Open the Folder in VS code

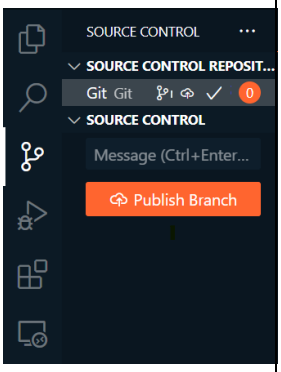


- Go to Source  Control which is on left side of the panel.

- Click on Initialize Repository for Initialize that folder to Git.



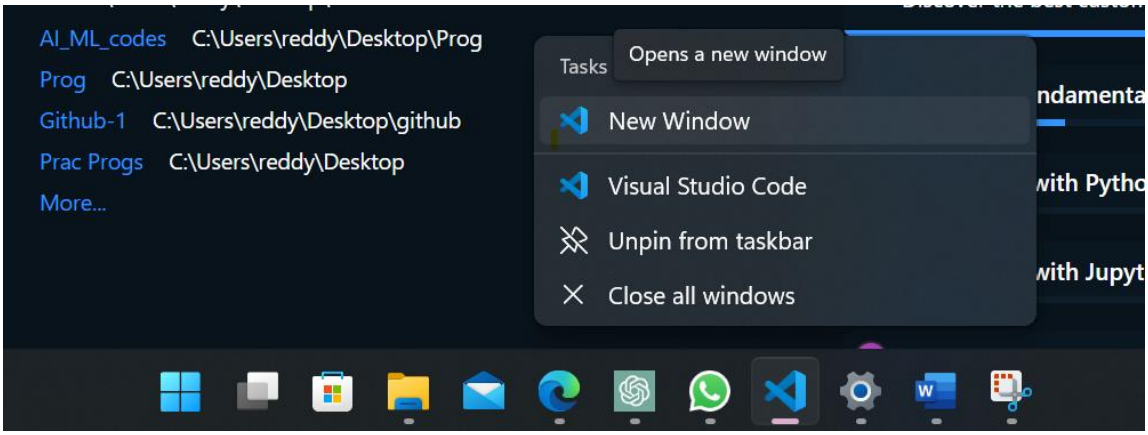
After successfully Initial of folder Source Control will appear like this--->



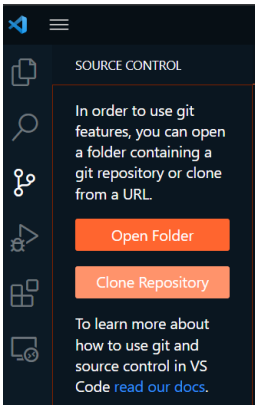
Cloning a repository.

- Cloning is the process of creating an exact copy of a Git repository from a remote source (such as a Git hosting service like GitHub, GitLab, or Bitbucket) to your local machine. When you clone a repository, you download all the files, commit history, branches, and metadata associated with the original repository.

- Every time when u do a new thing in VS code u need to open a new Window.

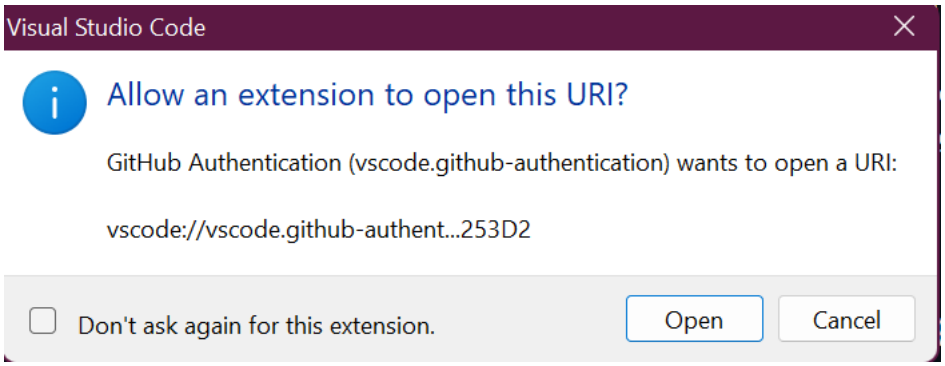


- Click on Clone Repository in Source Control.



- Click on remote sources.

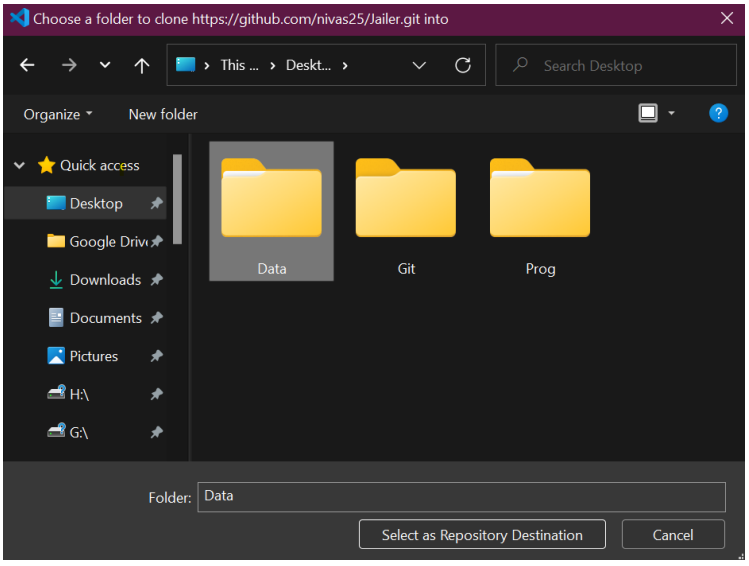
- An Authentication Allow message appears in browser click allow again click on open in VS code Authentication.



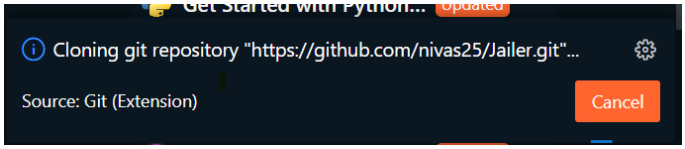
NOTE: u need to have GitHub account created using email id which we u used in git cmd configuration…

- After successful authentication in browser by entering password and username of

  GitHub account, in VS code this appears.
- Click on the required repository need to cloned.
- Select the Cloning Destination [USE a FOLDER for this process].

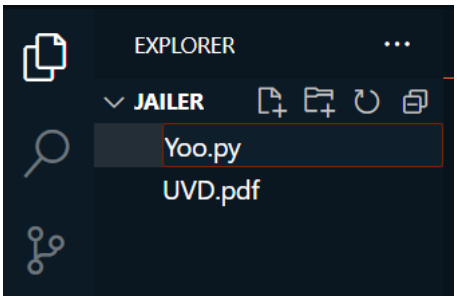


Cloning in Process ---→

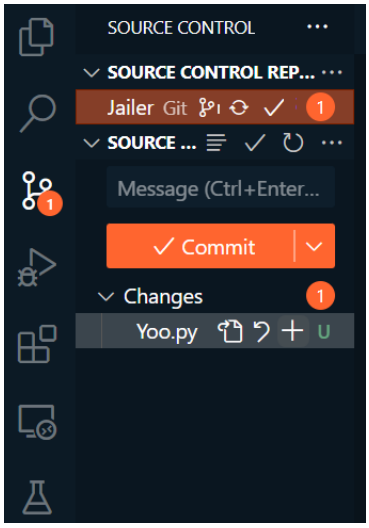


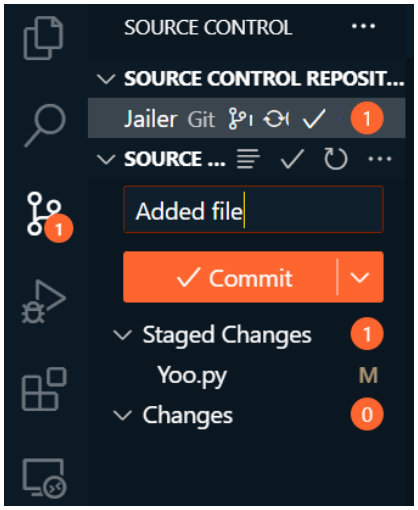
Making and Recording changes:

- Here I am creating a new file named Yoo.py and saving it.
- U can continue this experiment in previous window or new Window, it's you're wish.
- For creating a new file click on create a new file in Explorer [left side of window]



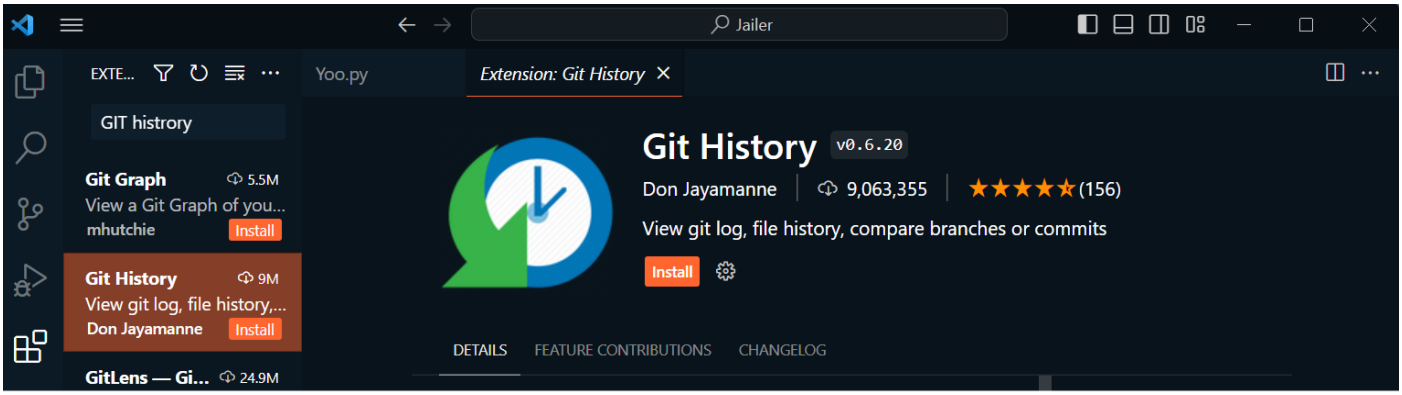
- In Source Control we can see in Changes tab a change is notified click on + to add it to Stagging place and give a commit message and click on "COMMIT".



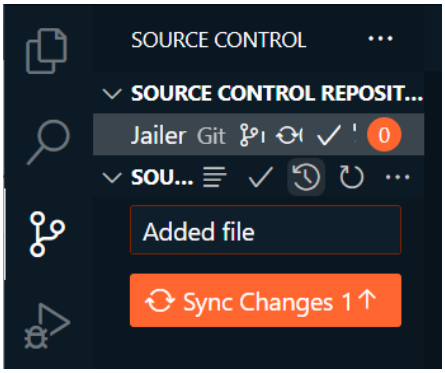


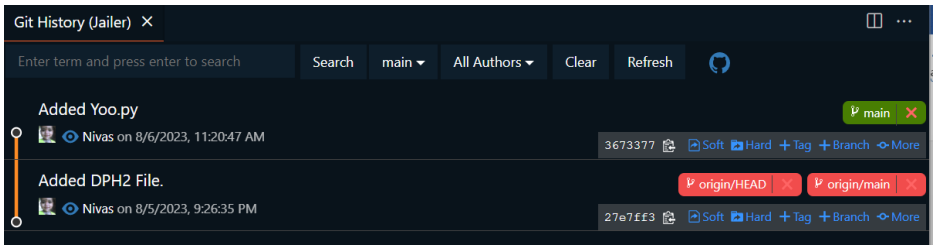
Viewing the history of all the Changes:

- Go to Extensions and search for "GIT HISTORY" and click on install



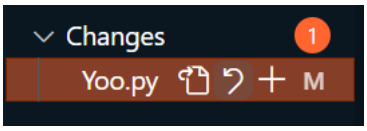
- In source control click on clock like symbol to view the commit history.



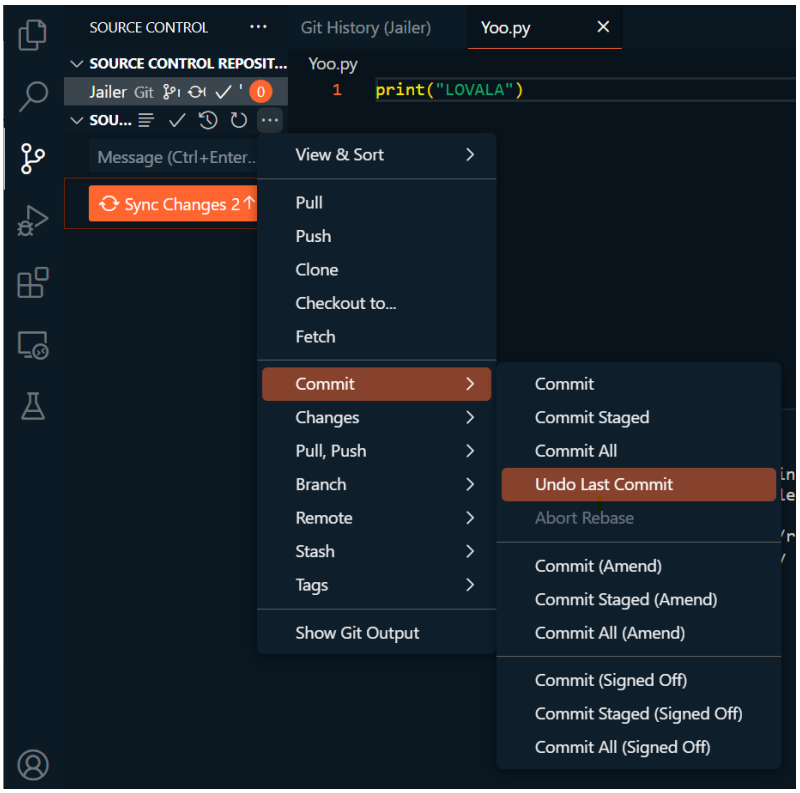


Undoing Changes:

- **Undoing a change when it's in changed stage** → In source control--- In Changed ---- click on discard Changes…



- **Undoing the Commits** → In source control --- click on 3dots --- commit --- undo last commit.



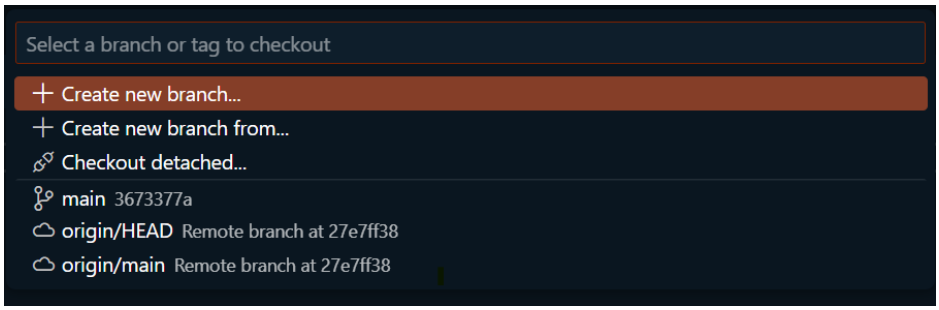
- By clicking remove from Stage and discard changes it completely goes to previous commit stage.

Git Branching and merging.

Creating and switching to new branches

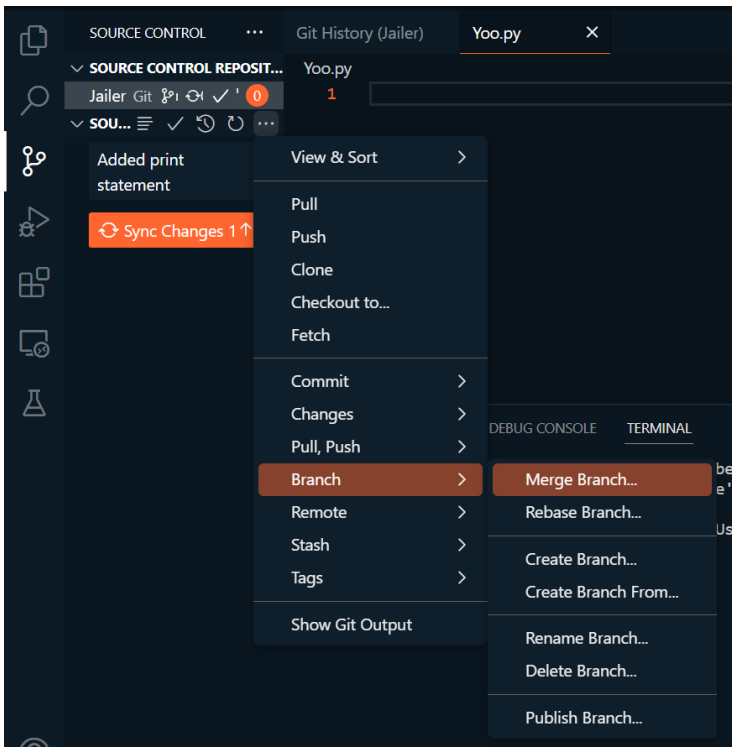
- Click on branch name [main] which is visible on left-bottom of window.

- To create a new branch, click on create new branch and to switch between them click on their names.



Merging local branches together:

- Select the branch you want to merge into the current branch
- In source control --- 3 dots --- Branch --- Merge Branch
- Resolve conflicts any and stage and commit the changes to complete the merge.



Week 3

3. Array Aggregation Functions [NUMPY]

Code:

```
import numpy as n

a=n.array([54,78,32,46,89,76])

print("\nAn Array:",a)

print("\nSum:",n.sum(a))

print("Product",n.prod(a))

print("Mean:",n.mean(a))

print("Standard Deviation:",n.std(a))

print("Variance",n.var(a))

print("Minimum Value:",n.min(a))

print("Max:",n.max(a))

print("Min Index:",n.argmin(a))

print("Max Index:",n.argmax(a))

print("Median:",n.median(a))

print("Product:",n.prod(a))
```

Output:

```
An Array: [54 78 32 46 89 76]

Sum: 375
Product -1012440064
Mean: 62.5
Standard Deviation: 19.997916558148418
Variance 399.9166666666667
Minimum Value: 32
Max: 89
Min Index: 2
Max Index: 4
Median: 65.0
Product: -1012440064
```

4. Vectorized Operations using NUMPY
Vectorized Sum and Multiplication

Code:
[Product]

```
import numpy as np
import timeit

np.a=[4,5,1]
print(np.prod(np.a))
print("Time taken by vectorized product : ",end= "")
%timeit np.prod(np.a)

total = 1
for item in np.a:
 total =total*item
t = total
print(t)
print("Time taken by iterative multiplication : ",end= "")
%timeit t
```

Output:

```
20
Time taken by vectorized product : 6.37 µs ± 993 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
20
Time taken by iterative multiplication : 22.5 ns ± 5.09 ns per loop (mean ± std. dev. of 7 runs, 100,000,000 loops each)
```

Code:
[Sum]

```
import numpy as n
import timeit

print(n.sum(n.arange(4)))
print("Time taken  to vectorized sum:")
%timeit n.sum(n.arange(4))

t=0
for i in range(0,4):
    t+=i
a=t
print("\n"+str(a))
print("Time Taken by iterative sum:",end="")
%timeit a
```

Output:

```
6
Time taken  to vectorized sum:
13.9 µs ± 3.23 µs per loop (mean ± std. dev. of 7 runs, 100,000 loops each)

6
Time Taken by iterative sum:24.5 ns ± 4.3 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
```

5. **Use Map, Filter, Reduce and Lambda Functions on List using Numpy**

Code:

```
import numpy as n
da=[60,8,7,5,34,78]
d=n.array(da)
from functools import reduce as r
print(list(map(lambda num:num**2,d)))
print(list(filter(lambda num:num>2,d)))
print(r(lambda x,y:x+y,d))
```

Output:

```
[3600, 64, 49, 25, 1156, 6084]
[60, 8, 7, 5, 34, 78]
192
```

6. **Using aggregation functions on a Data Frame**

Code:

```
import pandas as p
d=p.DataFrame([[2,5,6],
               [4,6,3],
               [5,7,8]],
               columns=["Maths","Java","Py"])
print(d)
c=d.agg(['sum','min','max','count','mean','median','std','size',])
print()
print(c)
```

Output:

```
   Maths  Java  Py
0      2     5   6
1      4     6   3
2      5     7   8

            Maths  Java         Py
sum     11.000000  18.0  17.000000
min      2.000000   5.0   3.000000
max      5.000000   7.0   8.000000
count    3.000000   3.0   3.000000
mean     3.666667   6.0   5.666667
median   4.000000   6.0   6.000000
std      1.527525   1.0   2.516611
size     3.000000   3.0   3.000000
```

7. **Grouping using Pandas on a Dataframe**

Code:

```
import pandas as p

t={
    'Course':["PY","JV","DBMS","MMA","MMA"],
    'Fee':[300,600,21,350,67],
    'Complexity':[100,56,32,10,67]
}

d=p.DataFrame(t)
print(d)

c=d.groupby('Course').agg({'Fee':'min'})
print("\n",c)
```

Output:

```
  Course  Fee  Complexity
0     PY  300         100
1     JV  600          56
2   DBMS   21          32
3    MMA  350          10
4    MMA   67          67

          Fee
Course
DBMS       21
JV        600
MMA        67
PY        300
```

8. **Pivot and melt functions using Pandas.**

Code:

```
import pandas as p

t={
    'Course':["PY","JV","DBMS","MMA","MMA"],
    'Fee':[300,600,21,350,67],
    'Complexity':[100,56,32,10,67]
}

d=p.DataFrame(t)

print(d)
print("\n",d.pivot(columns='Course',values='Complexity'))
print("\n",d.melt())
```

Output:

```
  Course  Fee  Complexity
0     PY  300         100
1     JV  600          56
2   DBMS   21          32
3    MMA  350          10
4    MMA   67          67

 Course  DBMS    JV   MMA     PY
0        NaN   NaN   NaN  100.0
1        NaN  56.0   NaN    NaN
2       32.0   NaN   NaN    NaN
3        NaN   NaN  10.0    NaN
4        NaN   NaN  67.0    NaN

          variable value
0       Course    PY
1       Course    JV
2       Course  DBMS
3       Course   MMA
4       Course   MMA
5          Fee   300
6          Fee   600
7          Fee    21
8          Fee   350
9          Fee    67
10  Complexity   100
11  Complexity    56
12  Complexity    32
13  Complexity    10
14  Complexity    67
```

9. **Use Map, Filter and Reduce, Lambda functions using Pandas [Data Frame]**

Code:

```
import pandas as pd
from functools import reduce

data = {
    'Numbers': [1, 2, 3, 4, 5],
    'Letters': ['A', 'B', 'C', 'D', 'E']
}

df = pd.DataFrame(data)

sq=df['Numbers'].map(lambda x: x**2)

ev=list(filter(lambda x: x % 2 == 0, df['Numbers']))

po = reduce(lambda x, y: x * y, df['Numbers'])

print("Dataframe:\n",df)
print("\nMap for Squaring:\n",sq)
print("\nReduce for product:\n", po)
```

Output:

```
Dataframe:
    Numbers Letters
0        1       A
1        2       B
2        3       C
3        4       D
4        5       E

Map for Squaring:
 0     1
1     4
2     9
3    16
4    25
Name: Numbers, dtype: int64

Reduce for product:
 120
```

10. **Time series using Pandas (resample, shift operations)**

Code:

```
import numpy as n
import pandas as p

d=p.DataFrame(
    {"date":p.date_range(start="2023-09-
07",periods=5,freq="D"),"temp":n.random.randint(18,30,size=5)}
)

d["f"]=d["temp"].shift(1)
print("Shift:\n",d)

dfw=d.resample("M",on="date").mean()
print("\nResampling:\n",dfw)
```

Output:

```
Shift:
         date  temp     f
0 2023-09-07    27   NaN
1 2023-09-08    25  27.0
2 2023-09-09    29  25.0
3 2023-09-10    28  29.0
4 2023-09-11    27  28.0

Resampling:
             temp      f
date
2023-09-30  27.2  27.25
```

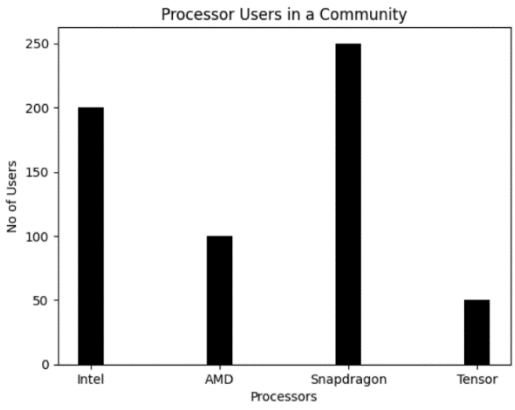
11. Data visualization using Matplotlib (Bar chart,pie,line ,histogram,scatter)

Code:

Output:

Bar Chart:

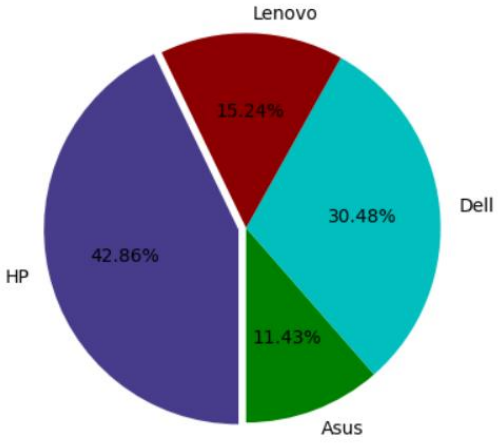
```
from matplotlib import pyplot as p

pro_na=["Intel","AMD","Snapdragon","Tensor"]
use=[200,100,250,50]

p.bar(pro_na,use,color='black',width=0.2)
p.xlabel("Processors"),p.ylabel("No of Users")
p.title("Processor Users in a Community")
p.show()
```



Pie chart:

Output:

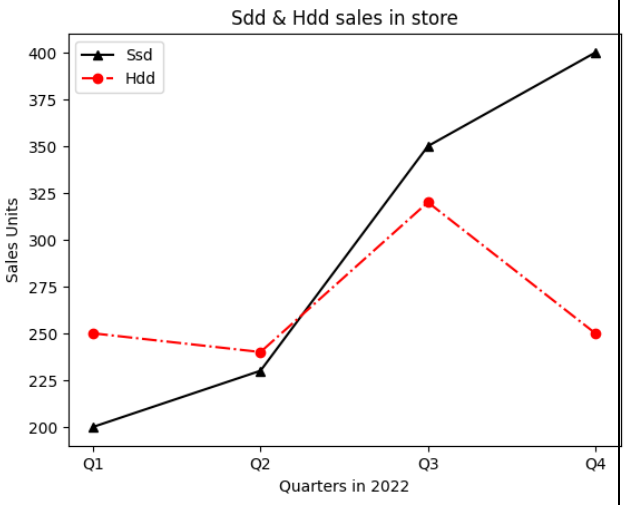
```
from matplotlib import pyplot as pi
us=[12,32,16,45]
la=["Asus","Dell","Lenovo","HP"]
e=[0,0,0,0.04]
c=["g","c","#8B0000","#473C8B"]
pi.pie(us,labels=la,startangle=270,
      explode=e,colors=c,autopct='%1.2f%%')

pi.show()
```



Line graph:

Output:

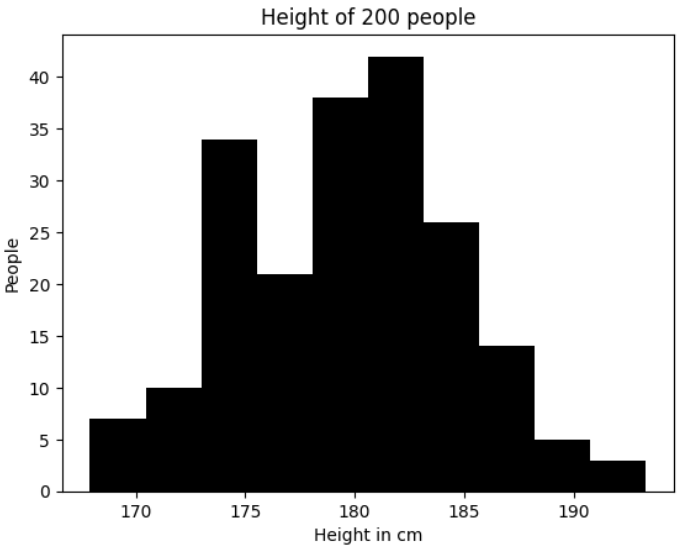
```
from matplotlib import pyplot as p
Q=["Q1","Q2","Q3","Q4"]
ssd=[200,230,350,400]
hdd=[250,240,320,250]
p.plot(Q,ssd,'^-',color='black')
p.plot(Q,hdd,'o-.r')
p.xlabel("Quarters in 2022"),p.ylabel("Sales
Units")
p.title("Sdd & Hdd sales in store")
p.legend(['Ssd','Hdd'])
p.show()
```



Histogram:

```
from matplotlib import pyplot as p
import numpy as n
x=n.random.normal(180,5,200)
p.hist(x,color='k')
p.xlabel("Height in
cm"),p.ylabel("People")
p.title("Height of 200 people")
p.show()
```

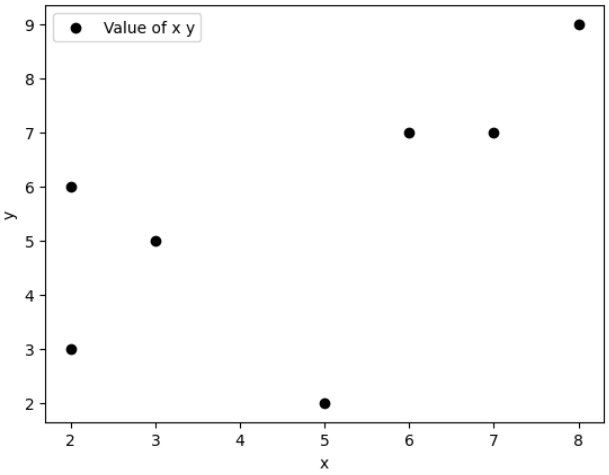
Output:



Scatter Plot:

```
from matplotlib import pyplot as p
x=[2,6,8,7,3,2,5]
y=[6,7,9,7,5,3,2]
c=['k','b']
p.scatter(x,y,label='Value of x
y',color='k')
p.xlabel('x')
p.ylabel('y')
p.legend()
p.show()
```

Output:

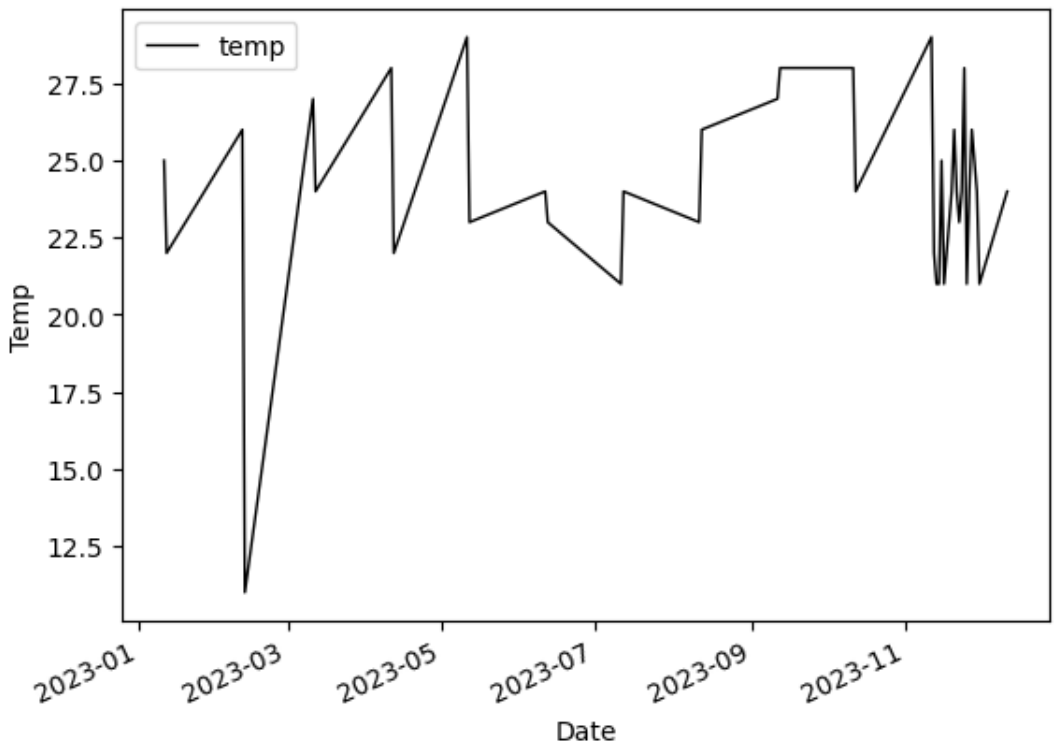


12. **Visualization of time series data using temperature on different days**

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

t = pd.read_csv('C:\\Users\\Desktop\\Data\\te.csv', parse_dates=['day'],
index_col='day')

a = t.plot(color='k', linewidth=1)

plt.xticks(rotation=25)

a.set_ylabel('Temp')

plt.xlabel('Date')

plt.show()
```

Output:

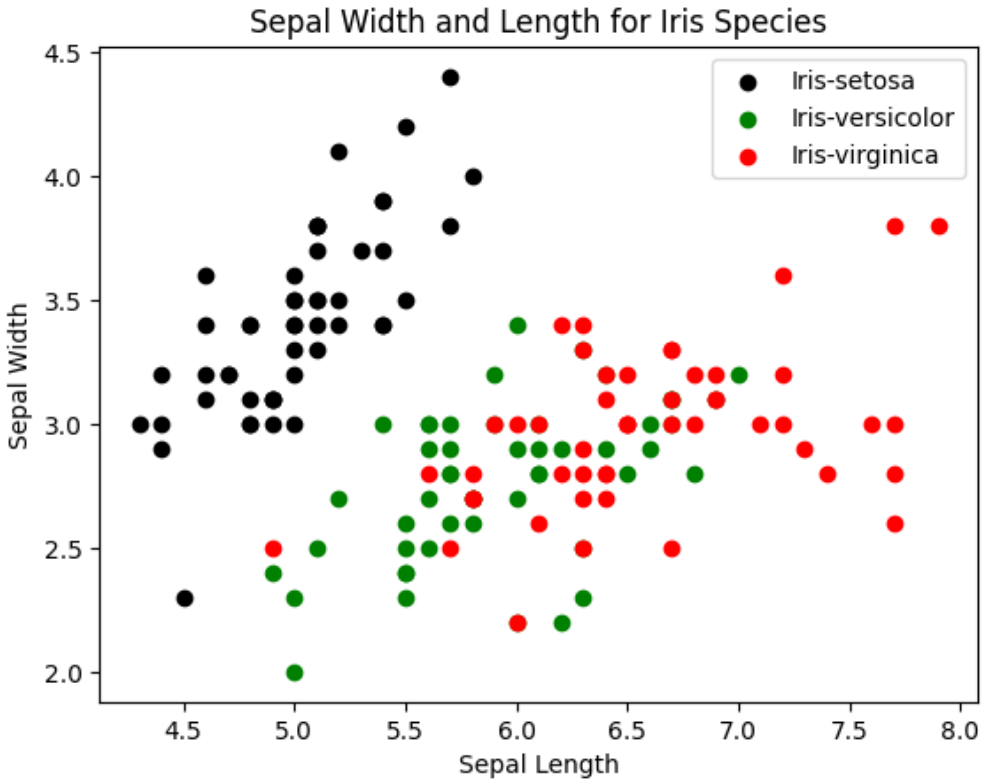


13. **Visualization of Iris-dataset using Scatter Plot**

Code:

```
import pandas as pd
from matplotlib import pyplot as plt
t = pd.read_csv("C:\\Users\\Desktop\\Data\\i.csv")

species_colors = {
    'Iris-setosa': 'k','Iris-versicolor': 'g','Iris-virginica': 'r'
}

for species, color in species_colors.items():
    sl = t[t['species'] == species]['sepal_length']
    sw = t[t['species'] == species]['sepal_width']
    plt.scatter(sl, sw, color=color, label=species)

plt.legend()
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Sepal Width and Length for Iris Species')
plt.show()
```

Output:

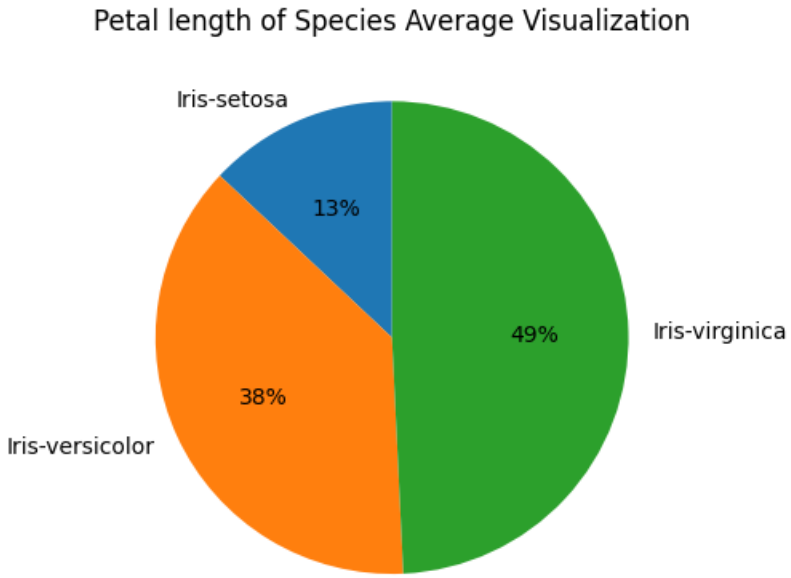


14. Visualization of Iris-dataset using Pie Chart

Code:

```
import pandas as pd
from matplotlib import pyplot as plt

t = pd.read_csv("C:\\Users\\Desktop\\Data\\i.csv")
sv=t.groupby("species")["petal_length"].mean()

plt.pie(sv,labels=sv.index,startangle=90,autopct="%1.0f%%")
plt.title("Petal length of Species Average Visualization")
plt.show
```

Output:

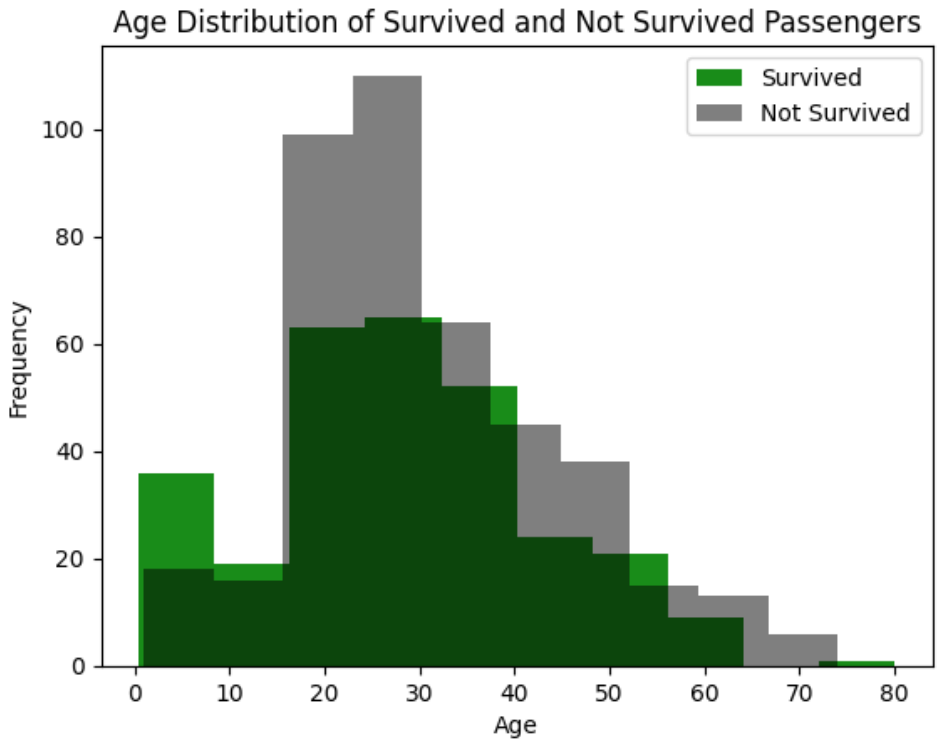


15. **Visualization of Titanic-dataset using Histogram**

Code:

```
import matplotlib.pyplot as plt
import pandas as pd

data = pd.read_csv('C:\\Users\\Desktop\\Progs\\Titanic-Dataset.csv')
age_survived = data[data['Survived'] == 1]['Age']
age_not_survived = data[data['Survived'] == 0]['Age']

plt.hist(age_survived,  color='g', alpha=0.9, label='Survived')
plt.hist(age_not_survived, color='k', alpha=0.5,label='Not Survived')

plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Age Distribution of Survived and Not Survived Passengers')
plt.legend()
plt.show()
```

Output:

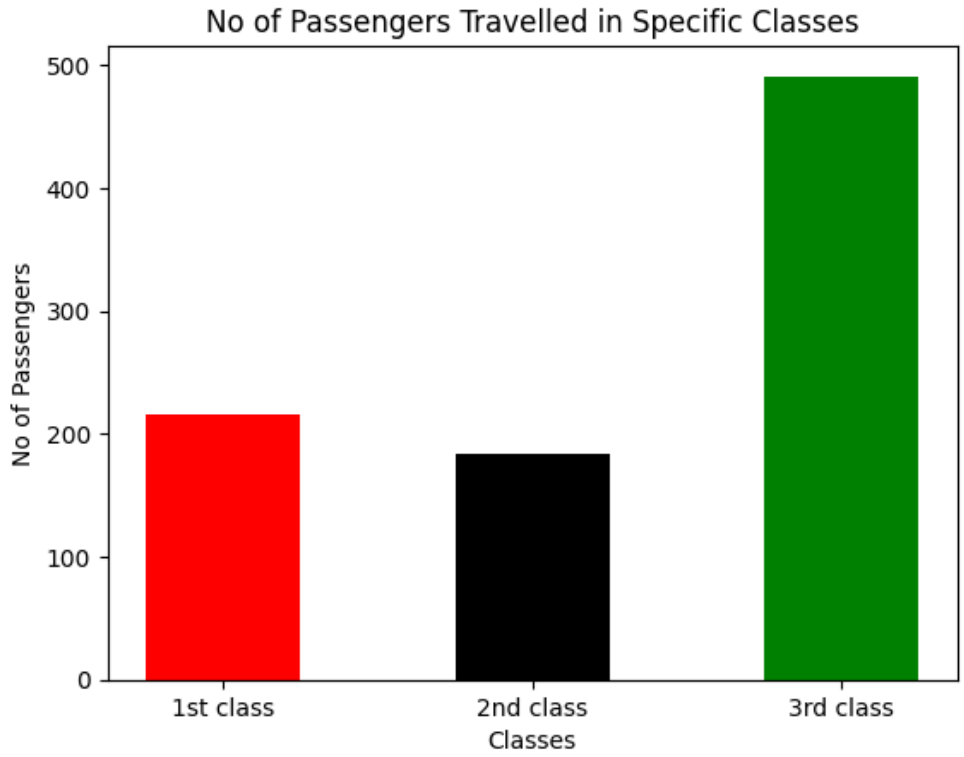


16. Visualization of Titanic-dataset using bar chart

Code:

```
import pandas as p
import matplotlib.pyplot as m
d=p.read_csv("C:\\Users\\Desktop\\Progs\\Titanic-Dataset.csv")

c=d["Pclass"].value_counts()
co=['g','r','k']
m.bar(c.index,c.values,color=co,width=0.5)

m.xticks([1,2,3],["1st class","2nd class","3rd class"])
m.xlabel("Classes");m.ylabel("No of Passengers");m.title("No of
Passengers Travelled in Specific Classes")
m.show()
```

Output:



17. Visualize Employee dataset using Line graph [Represent Salary and Experience]

Code:

```
import pandas as p
import matplotlib.pyplot as m
d={"Ex":[1,1.3,1.5,2,2.2,2.9,3,3.2,3.2],
   "Salary":[1000,3000,6000,8000,10000,12000,18000,20000,30000]}

df=p.DataFrame(d)
m.plot(df["Ex"],df["Salary"],'^--',color='k')
m.xlabel("Experience in years");m.ylabel("Salary");m.title("Salary with
Experience")
m.show()
```

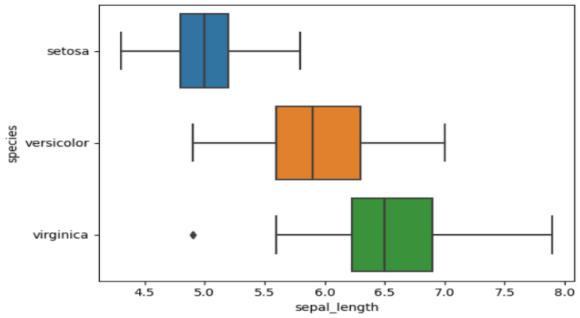
Output:

18. Visualize Iris dataset using Box-Plot

Code:

```
import seaborn as s
import matplotlib.pyplot as p
d=s.load_dataset('iris')
s.boxplot(x=d['sepal_length'],y=d['species'])
p.show()
```

Output:



19.

2. Create a dataframe with following data.

	First Name	Last Name	Type	Department	YoE	Salary
0	Aryan	Singh	Full-time Employee	Administration	2	20000
1	Rohan	Agarwal	Intern	Technical	3	5000
2	Riya	Shah	Full-time Employee	Administration	5	10000
3	Yash	Bhatia	Part-time Employee	Technical	7	10000
4	Siddhant	Khanna	Full-time Employee	Management	6	20000

1. Make a pivot table which shows average salary of each type of employee for each department.

2. Make a pivot table which shows the sum and mean of the salaries of each type of employee and the number of employees of each type.

3. Make a pivot table which shows standard deviation for salary column.

Code:

```
import pandas as p
import matplotlib.pyplot as m

d={
    "First_name":["Aryan","Rohan","Riya","Yash","Siddhant"],
    "Last_name":["Singh","Agarwal","Shah","Bhatia","Khanna"],
    "Type":["Full-Time","Itern","Full-Time","Part-Time","Full-Time"],
    "Dept":["Administration","Technical","Administration","Technical","Manageme
nt"],
    'YoE':[2,3,5,7,6],"Salary":[20000,5000,10000,10000,20000]
}

df=p.DataFrame(d)
av=df.pivot_table(index=['Dept', 'Type'], values='Salary', aggfunc='mean')
print("Average Salary from ecah dept:\n",av)

sm=df.pivot_table(index=['Type'], values='Salary', aggfunc=['sum', 'mean',
'count'])
sm.columns=['Total Salary', 'Mean Salary', 'Number of Employees']
print("\nSum and Mean of:\n",sm)

st=df.pivot_table(values='Salary', index='Type',aggfunc='std')
print("\nStandard Deviation:\n",st)
```

Output:

```
Average Salary from ecah dept:
                            Salary
Dept           Type
Administration Full-Time    15000
Management     Full-Time    20000
Technical      Itern         5000
               Part-Time    10000

Sum and Mean of:
            Total Salary   Mean Salary  Number of Employees
Type
Full-Time          50000  16666.666667                    3
Itern               5000   5000.000000                    1
Part-Time          10000  10000.000000                    1

Standard Deviation:
                 Salary
Type
Full-Time  5773.502692
```

20.

20.

3. Create two series as shown using pd.series() function.Series_A = [10,20,30,40,50] Series_B = [40,50,60,70,80].

i. Get the items not common to both.

ii. Identify the smallest and largest element in the series A

iii. Find the sum of series B

iv. Calculate average in the series A

v. Find median in the given series B

Code:

```
import pandas as pd
a = pd.Series([10, 20, 30, 40, 50])
b = pd.Series([40, 50, 60, 70, 80])
print("Series A:")
print(a)
print("\nSeries B:")
print(b)

non_com = a[~a.isin(b)].tolist() + b[~b.isin(a)].tolist()
print("Items not common to both Series:")
print(non_com)

print("\nSmallest element in Series A:\n", a.min())
print("\nLargest element in Series A:\n",a.max())

print("\nSum of Series B:\n", b.sum())
print("\nAverage of Series A:\n",a.mean())
print("\nMedian of Series B:\n", a.median())
```

Output:

```
Series A:
0    10
1    20
2    30
3    40
4    50
dtype: int64

Series B:
0    40
1    50
2    60
3    70
4    80
dtype: int64
Items not common to both Series:
[10, 20, 30, 60, 70, 80]
```

```
Smallest element in Series A:
 10

Largest element in Series A:
 50

Sum of Series B:
 300

Average of Series A:
 30.0

Median of Series B:
 30.0
```

21. Perform the following operations on Car manufacturing company dataset auto-mpg.csv given below using pandas

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18	8	307	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15	8	350	165	3693	11.5	71	1	buick skylark 320
2	18	6	318	150	3436	11.0	70	1	plymouth satellite
3	16	4	304	150	3433	12.0	80	1	amc rebel sst
4	17	8	302	140	3449	10.5	70	1	ford torino

i. statistical details of dataset

ii. Get all cars with 8 cylinders

iii. Get the number of cars manufactured in each year.

Code:

```
import pandas as pd
da={
    "mpg":[18,15,18,16,17],"cylinders":[8,8,6,4,8],"displacement":[307,350,318,
304,302],
    "horsepower":[130,165,150,150,140],"weigth":[3504,3693,3436,3433,3449],
    "acceleration":[12.0,11.5,11.0,12.0,10.5],"model year":[70,71,70,80,70],
    "origin":[1,1,1,1,1],"car
name":["cheverlot","buick","plymoth","amc","ford"]

}
df=pd.DataFrame(da)

sa=df.describe()
ei=df[df["cylinders"]==8]
ye = df.groupby('model year')["model year"].count()

print("Satistical:\n",sa)
print("\n8 cylinders:\n",ei)
print("\nBy year:\n",ye)
```

Output:

```
Satistical:
             mpg  cylinders  displacement  horsepower       weigth  \
count   5.00000   5.000000      5.000000    5.000000     5.000000
mean   16.80000   6.800000    316.200000  147.000000  3503.000000
std     1.30384   1.788854     19.879638   13.038405   110.006818
min    15.00000   4.000000    302.000000  130.000000  3433.000000
25%    16.00000   6.000000    304.000000  140.000000  3436.000000
50%    17.00000   8.000000    307.000000  150.000000  3449.000000
75%    18.00000   8.000000    318.000000  150.000000  3504.000000
max    18.00000   8.000000    350.000000  165.000000  3693.000000

       acceleration  model year  origin
count       5.00000     5.00000     5.0
mean       11.40000    72.20000     1.0
std         0.65192     4.38178     0.0
min        10.50000    70.00000     1.0
25%        11.00000    70.00000     1.0
50%        11.50000    70.00000     1.0
75%        12.00000    71.00000     1.0
max        12.00000    80.00000     1.0
```

```
8 cylinders:
    mpg  cylinders  displacement  horsepower  weigth  acceleration  model year  \
0   18          8           307         130    3504          12.0          70
1   15          8           350         165    3693          11.5          71
4   17          8           302         140    3449          10.5          70

   origin   car name
0       1  cheverlot
1       1      buick
4       1       ford

By year:
 model year
70    3
71    1
80    1
Name: model year, dtype: int64
```

22. **Data from an online platform has been collected. This data contains fuel consumption and 11 aspects of automobile design and performance for 32 automobiles. Variable description is given below.Dataset - 'mtcars.csv**

Sl No	Variables	Description	Categories
1	mpg	Miles/(US) gallon	
2	cyl	Number of cylinders	4, 6, 8
3	disp	Displacement (cu.in.)	
4	hp	Gross horsepower	
5	drat	Rear axle ratio	
6	wt	Weight (1000 lbs)	
7	qsec	Fastest time to travel 1/4 mile from standstill (in seconds)	
8	vs	Engine cylinder configuration	0, 1 (0 -V-shape; 1 - Straight Line)
9	am	Transmission type	0, 1 (0 - automatic; 1 - manual)
10	gear	Number of forward gears	3, 4 , 5 Manual transmissions have either 4 or 5 forward gears; Automatic either 3 or 4
11	carb	Number of carburettors	1, 2, 3, 4, 6, 8

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

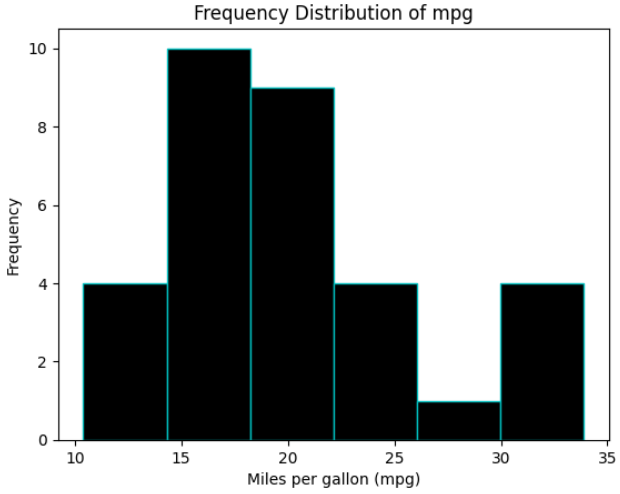
1. histogram to check the frequency distribution of the variable 'mpg' (Miles per gallon) and note down the interval having the highest frequency.
2. scatter plot to determine the relation between weight of the car and mpg
3. bar plot to check the frequency distribution of transmission type of cars.
4. Box and Whisker plot of mpg and interpret the five number summary.
5. Create a git repository and push source code to repo.

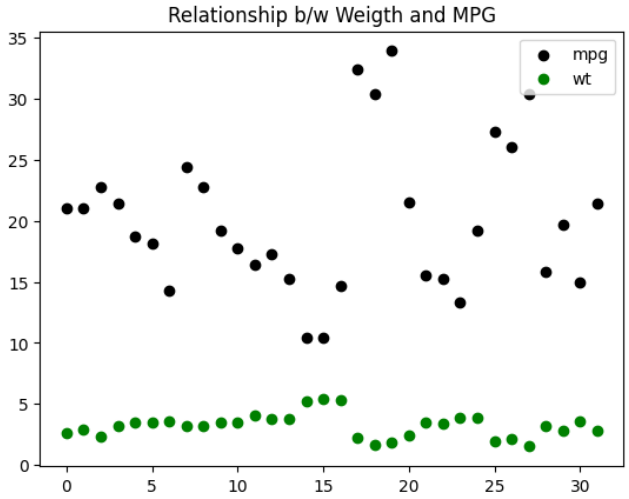
Code:

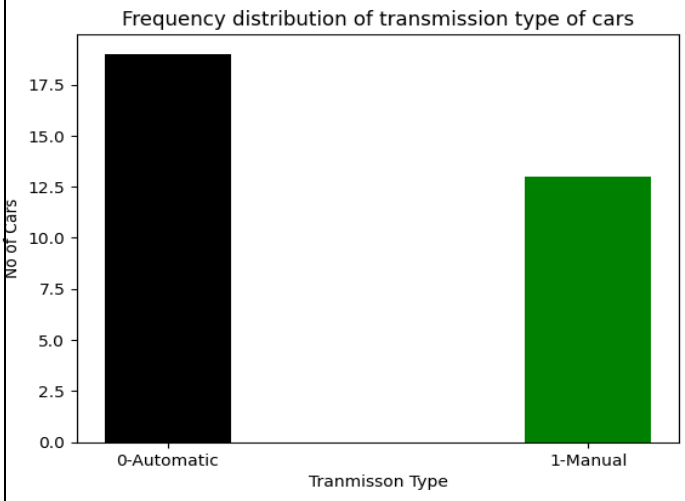
```
import pandas as p
import matplotlib.pyplot as m
import seaborn as s
# data as 32 Elements
data=p.read_csv("C:\\Users\\reddy\\Desktop\\Data\\mtcars.csv")

# HISTOGRAM
mpg=data['mpg']
m.hist(mpg,bins='auto',color='k',edgecolor='c')
m.xlabel('Miles per gallon (mpg)');m.ylabel('Frequency')
m.title('Frequency Distribution of mpg')
m.show()

# SCATTER
wt=data['wt']
iv=range(len(data))
m.scatter(iv,mpg,color='k',label='mpg')
m.scatter(iv,wt,color='g',label='wt')
m.title("Relationship b/w Weigth and MPG")
m.legend()
```

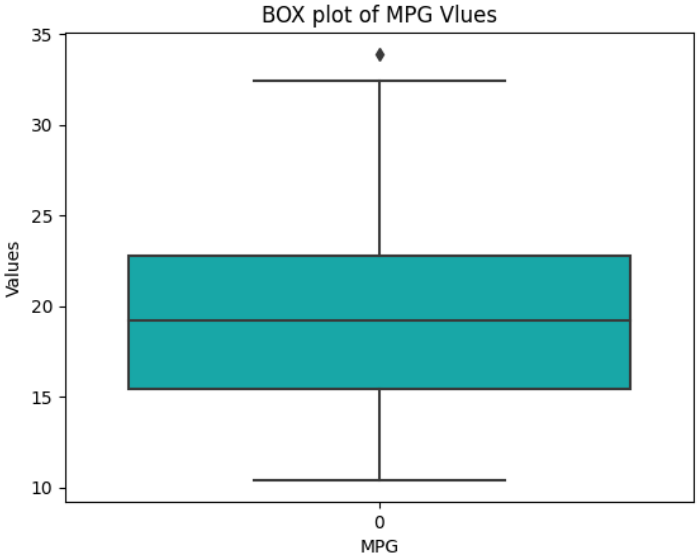
```
m.show()

# BAR PLOT
c=data['am'].value_counts()
co=['k','g']
m.bar(c.index,c.values,color=co,width=0.3)
m.xticks([0,1],['0-Automatic','1-Manual'])
m.xlabel("Tranmisson Type");m.ylabel("No of Cars")
m.title("Frequency distribution of transmission type of cars")
m.show()

# BOX PLOT
s.boxplot(mpg,color='c')
m.xlabel("MPG");m.ylabel("Values")
m.title("BOX plot of MPG Vlues")
```

Output:









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

1.Code to add 1000 steps to all the observations

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

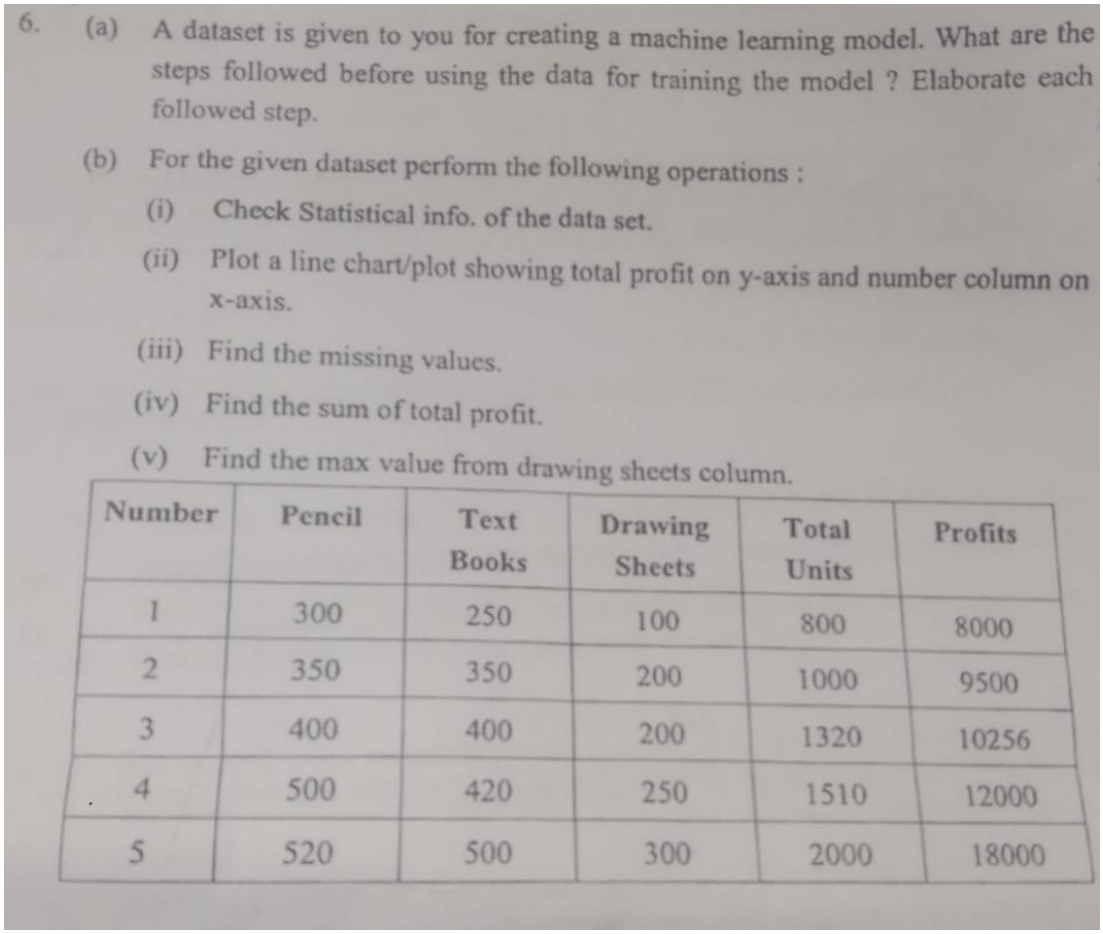
Code:

```
import pandas as p
import numpy as n

d={"Day":[1,2,3,4,5,6,7,8,9,10],
   "Steps":[4335,9552,7332,4504,5335,7552,8332,6504,8965,7689]}

dp=p.DataFrame(d)
dp["+1000 Steps"]=dp["Steps"]+1000

fi=dp[dp["+1000 Steps"]>7000]["Day"]

print("DataFrame:\n",dp)
print("\nDays on which Steps were >7000:\n",fi)
```

Output:

```
DataFrame:
    Day  Steps  +1000 Steps
0    1   4335         5335
1    2   9552        10552
2    3   7332         8332
3    4   4504         5504
4    5   5335         6335
5    6   7552         8552
6    7   8332         9332
7    8   6504         7504
8    9   8965         9965
9   10   7689         8689

Days on which Steps were >7000:
 1     2
2     3
5     6
6     7
7     8
8     9
9    10
Name: Day, dtype: int64
```

24.



6. (a) A dataset is given to you for creating a machine learning model. What are the steps followed before using the data for training the model ? Elaborate each followed step.

(b) For the given dataset perform the following operations :

(i) Check Statistical info. of the data set.

(ii) Plot a line chart/plot showing total profit on y-axis and number column on x-axis.

(iii) Find the missing values.

(iv) Find the sum of total profit.

(v) Find the max value from drawing sheets column.

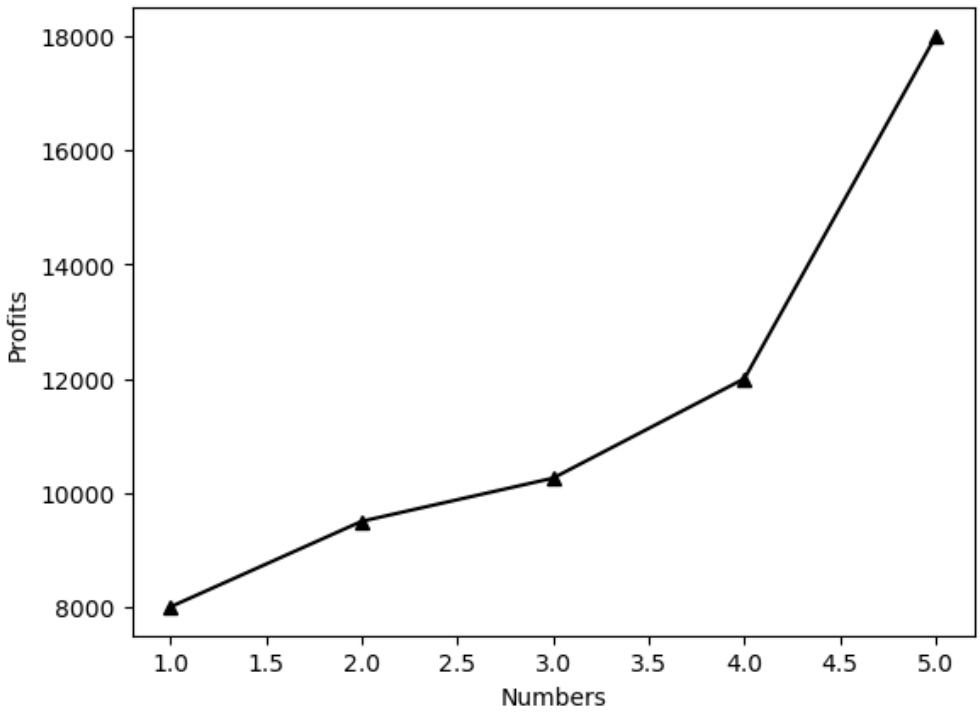
Number	Pencil	Text Books	Drawing Sheets	Total Units	Profits
1	300	250	100	800	8000
2	350	350	200	1000	9500
3	400	400	200	1320	10256
4	500	420	250	1510	12000
5	520	500	300	2000	18000

Code:

```
import numpy as n
import pandas as p
import matplotlib.pyplot as m
da={
    'n':[1,2,3,4,5],'Pencil':[300,350,400,500,520],'TextBooks':[250,350,400,420
,500],
    'Draw':[100,200,200,250,300],'Total':[800,1000,1320,1510,2000],"Profits":[8
000,9500,10256,12000,18000]
}
df=p.DataFrame(da)

sta=df.describe()
print("Statistics:\n",sta)

su=df['Profits'].sum()
print("\nSum of Profits:\n",su)

mi=df.isna()
print("\nMissing values:\n",mi)

print("\nMaximum Value:\n",df['Draw'].max())

m.plot(df['n'],df['Profits'],'^-',color='k')
m.xlabel("Numbers");m.ylabel("Profits")
m.show()
```

Output:

```
Statistics:
              n      Pencil   TextBooks        Draw        Total       Profits
count  5.000000    5.000000    5.000000    5.000000     5.000000      5.000000
mean   3.000000  414.000000  384.000000  210.000000  1326.000000  11551.200000
std    1.581139   94.762862   92.357999   74.161985   466.669048   3882.152393
min    1.000000  300.000000  250.000000  100.000000   800.000000   8000.000000
25%    2.000000  350.000000  350.000000  200.000000  1000.000000   9500.000000
50%    3.000000  400.000000  400.000000  200.000000  1320.000000  10256.000000
75%    4.000000  500.000000  420.000000  250.000000  1510.000000  12000.000000
max    5.000000  520.000000  500.000000  300.000000  2000.000000  18000.000000

Sum of Profits:
 57756

Missing values:
       n  Pencil  TextBooks   Draw  Total  Profits
0  False   False      False  False  False    False
1  False   False      False  False  False    False
2  False   False      False  False  False    False
3  False   False      False  False  False    False
4  False   False      False  False  False    False

Maximum Value:
 300
```



Week -04

25. **Linear Algebra [Vector , Scalar, Tensors, Matrix, Gradiant, Eigen Values and Vectors]**

Code:

```
import numpy as n
import tensorflow as t

s=50
print("Scalar : ",s)

m=n.array([[0, 2],[2, 3]])
v=n.array([[0, 2]])
print("\nMatrix:\n",m)
print("\nVector:\n",v)

print("\nTensor:")
fill_2d = t.fill([3, 3],4, '2d')
fill_string = t.fill([2, 2], "str", 'fill_tensor_string')
print("\nNumerics:\n",fill_2d)
print("\nString:\n",fill_string)
g=n.gradient(m)
print("\nGradient:\n",g)

w,v = n.linalg.eig(m)
mat_norm = n.linalg.norm(m)
print("\nEigen values:\n",w)
print("\nEigen vectors:\n",v)
print("\nMatrix norm:", mat_norm)
```

Output:

```
Scalar :  50

Matrix:
 [[0 2]
 [2 3]]

Vector:
 [[0 2]]

Tensor:

Numerics:
 tf.Tensor(
[[4 4 4]
 [4 4 4]
 [4 4 4]], shape=(3, 3), dtype=int32)

String:
 tf.Tensor(
[[b'str' b'str']
 [b'str' b'str']], shape=(2, 2), dtype=string)

Gradient:
 [array([[2., 1.],
        [2., 1.]]), array([[2., 2.],
        [1., 1.]])]

Eigen values:
 [-1.  4.]

Eigen vectors:
 [[-0.89442719 -0.4472136 ]
 [ 0.4472136  -0.89442719]]

Matrix norm: 4.123105625617661
```

26. Covariance and Co-relation

Code:

```
import pandas as pd
from sklearn import datasets

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=["sepal_length", "sepal_width",
"petal_length", "petal_width"])

df["class"] = iris.target

cov=df.iloc[:, 0:4].cov()
cor=df.iloc[:, 0:4].corr()

print("Covariance:\n",cov)
print("\nCorelation:\n",cor)
```

Output:

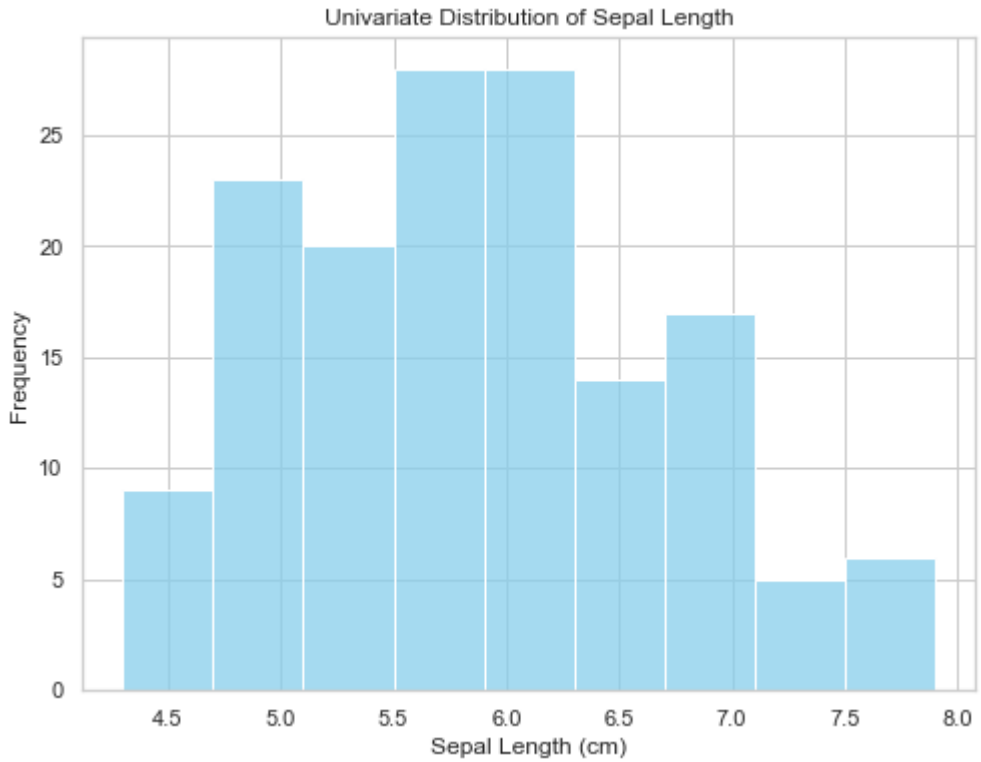
```
Covariance:
              sepal_length  sepal_width  petal_length  petal_width
sepal_length      0.685694    -0.042434      1.274315     0.516271
sepal_width      -0.042434     0.189979     -0.329656    -0.121639
petal_length      1.274315    -0.329656      3.116278     1.295609
petal_width       0.516271    -0.121639      1.295609     0.581006

Corelation:
              sepal_length  sepal_width  petal_length  petal_width
sepal_length      1.000000    -0.117570      0.871754     0.817941
sepal_width      -0.117570     1.000000     -0.428440    -0.366126
petal_length      0.871754    -0.428440      1.000000     0.962865
petal_width       0.817941    -0.366126      0.962865     1.000000
```

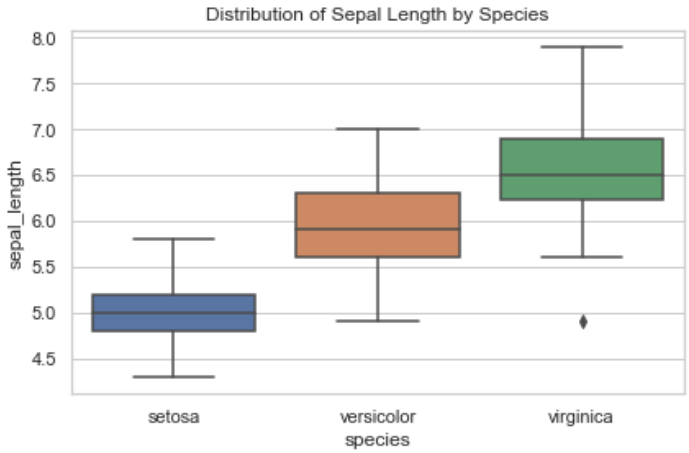
27. Univariate & Multivariate Distrubution Plot

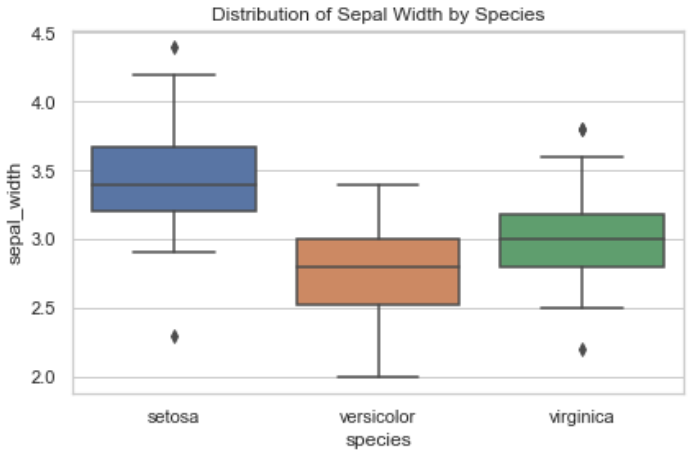
Code:

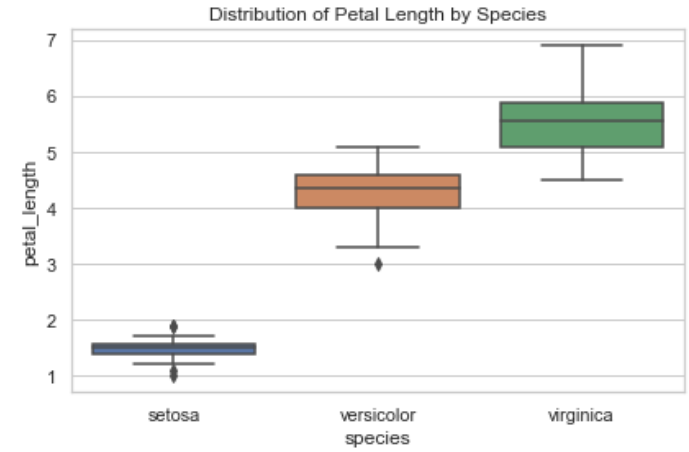
```
import seaborn as sns
import matplotlib.pyplot as plt

iris = sns.load_dataset("iris")

#Univariate
sepal_length_data = iris["sepal_length"]

plt.figure(figsize=(8, 6))

sns.histplot(sepal_length_data, color="skyblue")

plt.xlabel("Sepal Length (cm)")
plt.ylabel("Frequency")
plt.title("Univariate Distribution of Sepal Length")

plt.show()

#Multivariate

# Create subplots for each numeric variable
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# Plot sepal length distribution by species
sns.boxplot(x="species", y="sepal_length", data=iris, ax=axes[0, 0])
axes[0, 0].set_title("Distribution of Sepal Length by Species")

# Plot sepal width distribution by species
sns.boxplot(x="species", y="sepal_width", data=iris, ax=axes[0, 1])
axes[0, 1].set_title("Distribution of Sepal Width by Species")

# Plot petal length distribution by species
sns.boxplot(x="species", y="petal_length", data=iris, ax=axes[1, 0])
axes[1, 0].set_title("Distribution of Petal Length by Species")

# Plot petal width distribution by species
sns.boxplot(x="species", y="petal_width", data=iris, ax=axes[1, 1])
axes[1, 1].set_title("Distribution of Petal Width by Species")

plt.tight_layout()
plt.show()
```

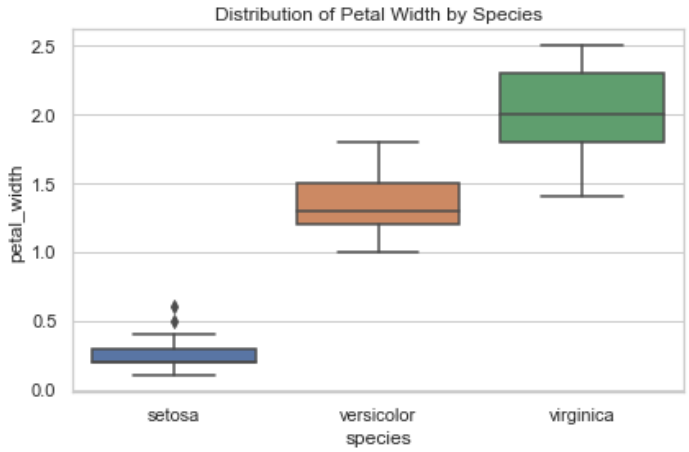
Output:


Univariate Distribution of Sepal Length
Frequency
Sepal Length (cm)
0
5
10
15
20
25
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


Distribution of Sepal Length by Species
sepal_length
species
setosa
versicolor
virginica


Distribution of Sepal Width by Species
sepal_width
species
setosa
versicolor
virginica


Distribution of Petal Length by Species
petal_length
species
setosa
versicolor
virginica

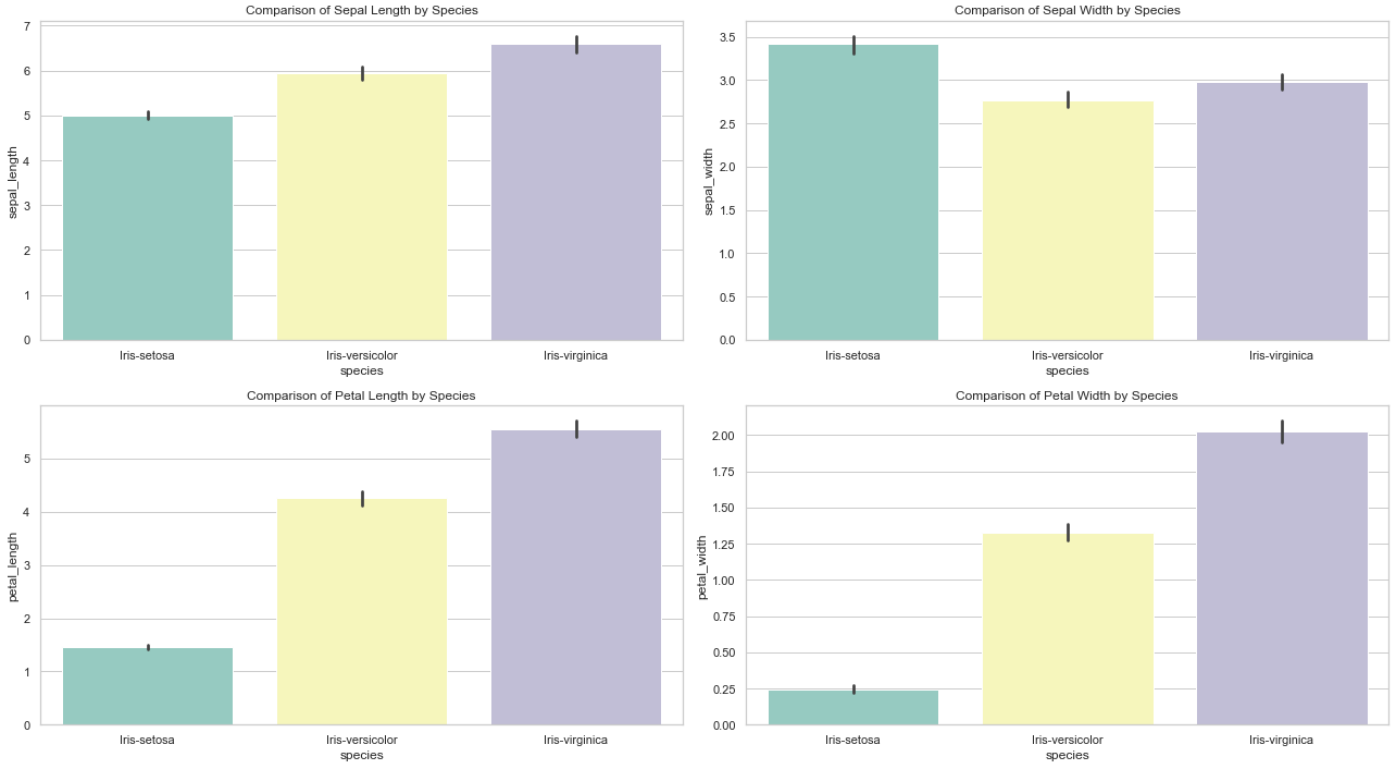

Distribution of Petal Width by Species
petal_width
species
setosa
versicolor
virginica

28. **Univariate & Multivariate Comparision Plots**

Code:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as p

iris = p.read_csv("C:\\Users\\reddy\\Desktop\\Data\\i.csv")

# Set the figure size
plt.figure(figsize=(18, 10))

# Create grouped bar plots for sepal length, sepal width, petal length, and
petal width by species
plt.subplot(2, 2, 1)
sns.barplot(x="species", y="sepal_length", data=iris, palette="Set3")
plt.title("Comparison of Sepal Length by Species")

plt.subplot(2, 2, 2)
sns.barplot(x="species", y="sepal_width", data=iris, palette="Set3")
plt.title("Comparison of Sepal Width by Species")

plt.subplot(2, 2, 3)
sns.barplot(x="species", y="petal_length", data=iris, palette="Set3")
plt.title("Comparison of Petal Length by Species")

plt.subplot(2, 2, 4)
sns.barplot(x="species", y="petal_width", data=iris, palette="Set3")
plt.title("Comparison of Petal Width by Species")

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```

Output:

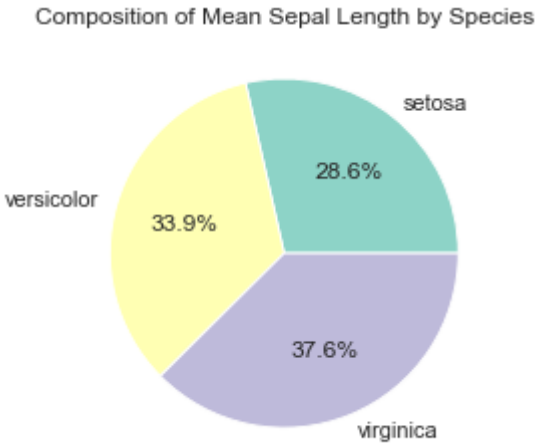

Comparison of Sepal Length by Species
sepal_length
Iris-setosa
Iris-versicolor
Iris-virginica
species
Comparison of Sepal Width by Species
sepal_width
Iris-setosa
Iris-versicolor
Iris-virginica
species
Comparison of Petal Length by Species
petal_length
Iris-setosa
Iris-versicolor
Iris-virginica
species
Comparison of Petal Width by Species
petal_width
Iris-setosa
Iris-versicolor
Iris-virginica
species

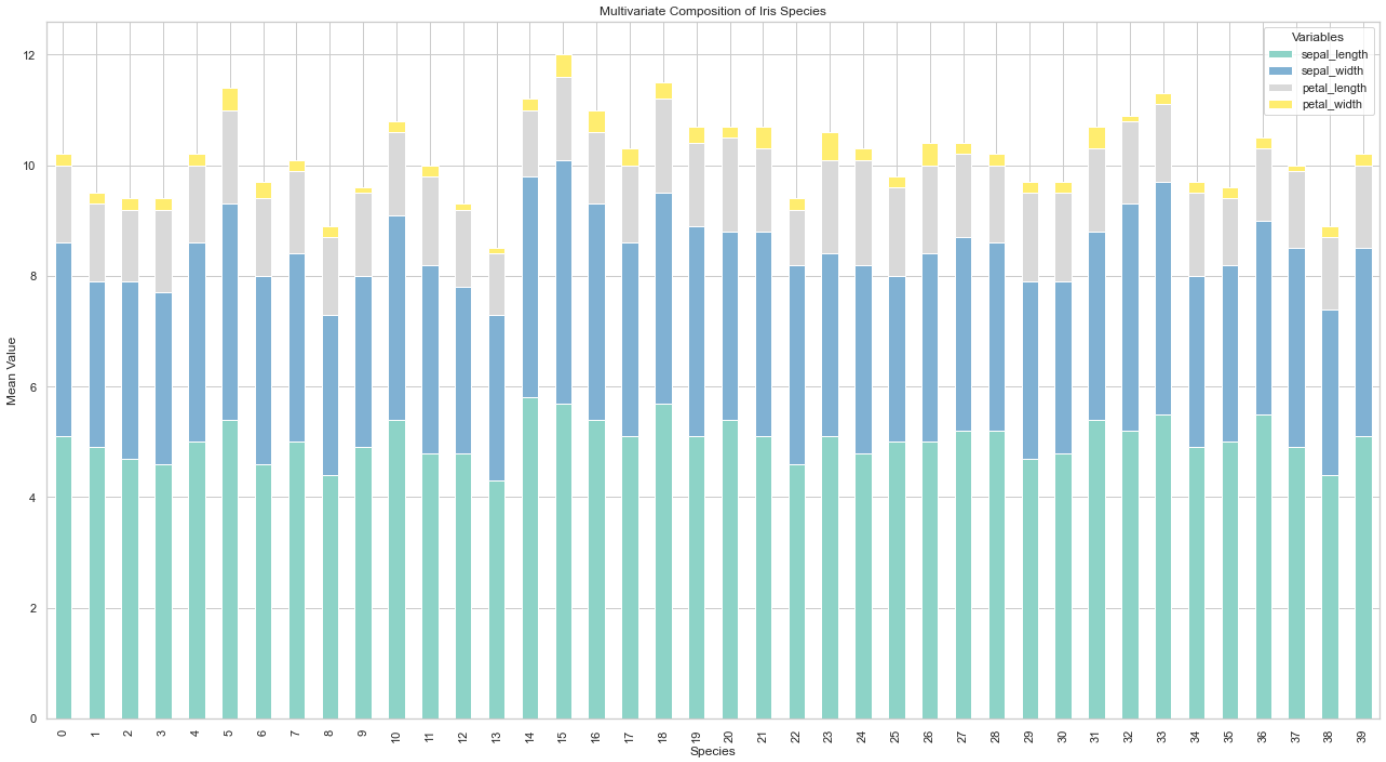
29. Univariate & Multivariate Composition Plot

Code:

```
import seaborn as sns
import matplotlib.pyplot as plt
# Load the Iris dataset
iris = sns.load_dataset("iris")

#univariate
sl= iris.groupby("species")["sepal_length"].mean()
plt.pie(sl, labels=sl.index, autopct='%1.1f%%',
colors=sns.color_palette("Set3"))
plt.title("Composition of Mean Sepal Length by Species")
plt.show()


#Multivariate
#Reducing the dataset to count of 40
iris = sns.load_dataset("iris").head(40)
# Group data by species and calculate the mean of numeric variables
species_data = iris.groupby("species")[["sepal_length", "sepal_width",
"petal_length", "petal_width"]]

# Create a stacked bar chart
species_data.plot(kind="bar", stacked=True, colormap="Set3", figsize=(18, 10))

# Set labels and title
plt.title("Multivariate Composition of Iris Species")
plt.xlabel("Species")
plt.ylabel("Mean Value")

# Show the plot
plt.legend(title="Variables", loc="upper right")
plt.tight_layout()
plt.show()
```

Output:


Composition of Mean Sepal Length by Species
setosa
28.6%
versicolor
33.9%
37.6%
virginica

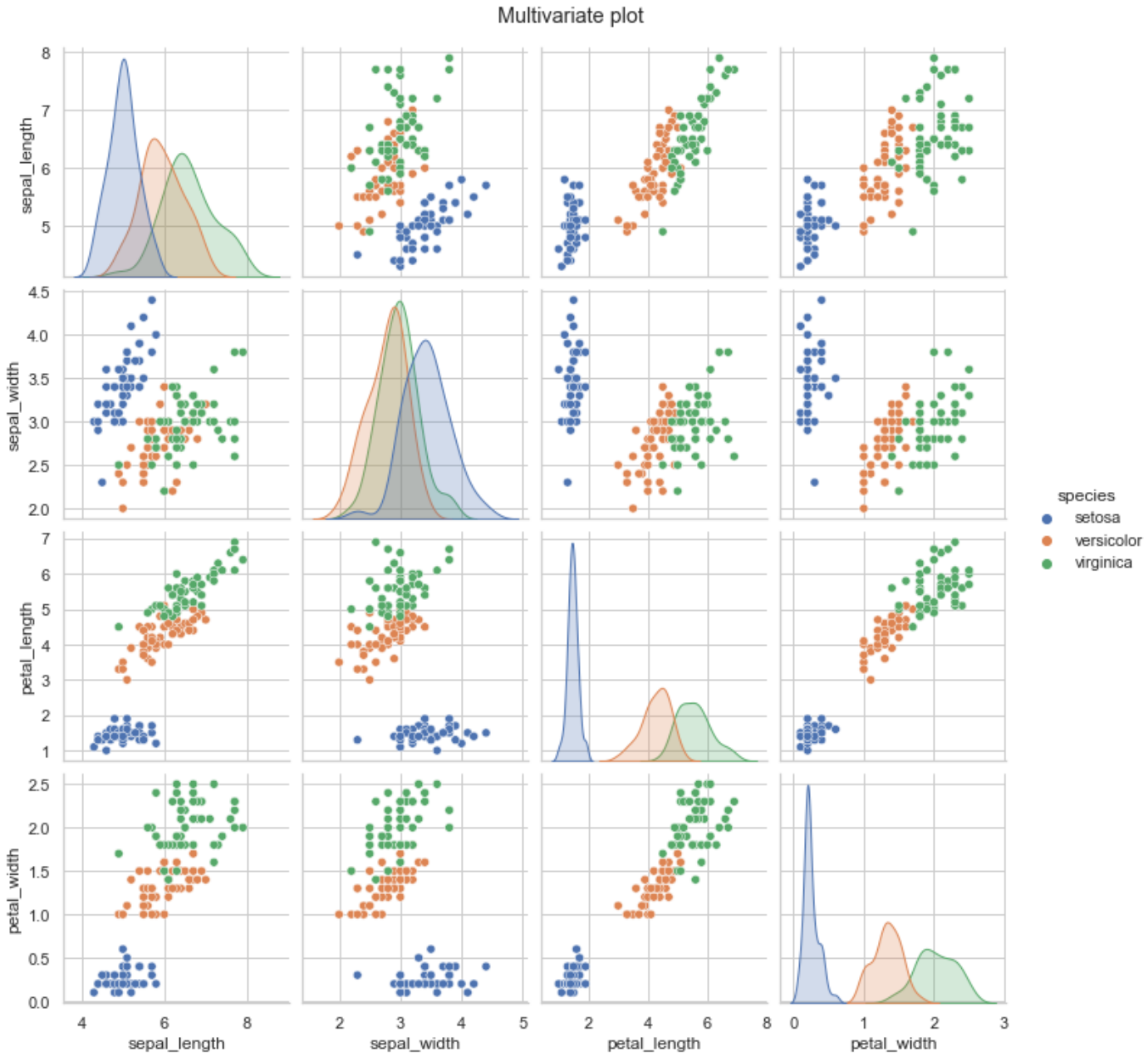

Multivariate Composition of Iris Species
Variables
sepal_length
sepal_width
petal_length
petal_width
Mean Value
Species

30. **Multivariate Relationship Plot**

Code:

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

df = sns.load_dataset('iris')

# plt.figure(figsize=(24,20))
sns.pairplot(df, hue="species")
plt.suptitle('Multivariate plot',y=1.02)
plt.show()
```

Output:



Week -05

31. **Detect missing values with pandas dataframe.**

functions: .info() and .isna()

Code:

```
import pandas as p

df=p.read_csv("C:\\Users\\Desktop\\Data\\Titanic-Dataset.csv")

info=df.info()

print("\n\n\nis_na:\n\n",df.isna().head(7))

is_null_su=df.isna().sum()
print("\n\n\nCount of all Missing values:\n\n",df.isna().sum())
```

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
```

```
is_na:

    PassengerId  Survived  Pclass   Name    Sex    Age  SibSp  Parch  Ticket  \
0        False     False   False  False  False  False  False  False   False
1        False     False   False  False  False  False  False  False   False
2        False     False   False  False  False  False  False  False   False
3        False     False   False  False  False  False  False  False   False
4        False     False   False  False  False  False  False  False   False
5        False     False   False  False  False   True  False  False   False
6        False     False   False  False  False  False  False  False   False

    Fare  Cabin  Embarked
0  False   True     False
1  False  False     False
2  False   True     False
3  False  False     False
4  False   True     False
5  False   True     False
6  False  False     False
```

```
Count of all Missing values:

 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
```

32. **Replace**

Code:

```
import numpy as n
df=p.read_csv("C:\\Users\\Desktop\\Data\\Titanic-Dataset.csv")

print(df.isna().sum())

#Replacing all NaN values with -1
df=df.replace({n.nan:-1})

print("\n\n\n")
print(df.isna().sum())
```

Output:

```
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



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

33. **Remove data objects with missing values**

Code:

```
df=p.read_csv("C:\\Users\\Desktop\\Data\\Titanic-Dataset.csv")
print("\nBefore Droping:\n")
df.info()

#drops Entier row data if as nan values in any coloumn
df=df.dropna()

#OR
#dp=dp.dropna(axis=0)

print('\n\nAfter Droping:\n')
df.info()
```

Output:

```
Before Droping:

<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
```

```
After Droping:

<class 'pandas.core.frame.DataFrame'>
Index: 183 entries, 1 to 889
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  183 non-null    int64
 1   Survived     183 non-null    int64
 2   Pclass       183 non-null    int64
 3   Name         183 non-null    object
 4   Sex          183 non-null    object
 5   Age          183 non-null    float64
 6   SibSp        183 non-null    int64
 7   Parch        183 non-null    int64
 8   Ticket       183 non-null    object
 9   Fare         183 non-null    float64
 10  Cabin        183 non-null    object
 11  Embarked     183 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 18.6+ KB
```

34. **Remove the attributes with missing values**

Code:

```
df=p.read_csv("C:\\Users\\Desktop\\Data\\Titanic-Dataset.csv")

print("\nBefore Droping:\n")
df.info()

df=df.dropna(axis=1)

#OR
#df=df.drop(columns=df.columns[df.isnull().any()])

print('\n\nAfter Droping:\n')
df.info()
```

Output:

```
Before Droping:

<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
```

```
After Droping:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 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   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Ticket       891 non-null    object
 8   Fare         891 non-null    float64
dtypes: float64(1), int64(5), object(3)
memory usage: 62.8+ KB
```

35. **Estimate and impute missing values**

Filling it with some Arbitrary value here it is 0

Code:

```
df=p.read_csv("C:\\Users\\Desktop\\Data\\Titanic-Dataset.csv")

print("Before Filling Null values:\n\n",df.isna().sum())

df=df.fillna(0)

print("\n\nAfter Filling Null Values:\n\n",df.isna().sum())
```

Output:

```
Before Filling Null values:

 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
```

```
After Filling Null Values:

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

36. **Replacing with Mean Value**

Code:

```
df=p.read_csv("C:\\Users\\Desktop\\Data\\Titanic-Dataset.csv")

print("Before Replacing:\n\n",df['Age'].head(7))

print("\nMean of Age Column:",df['Age'].mean())

dp=df['Age'].fillna(df['Age'].mean())
print("\nAfter Replacing with Mean:\n\n",dp.head(7))
```

Output:

```
Before Replacing:

 0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5     NaN
6    54.0
Name: Age, dtype: float64

Mean of Age Column: 29.69911764705882

After Replacing with Mean:

 0    22.000000
1    38.000000
2    26.000000
3    35.000000
4    35.000000
5    29.699118
6    54.000000
Name: Age, dtype: float64
```

37. **Replacing with Median Value**

Code:

```
df=p.read_csv("C:\\Users\\Desktop\\Data\\Titanic-Dataset.csv")

print("Before Replacing:\n\n",df['Age'].head(7))

print("\nMedian of Age Column:",df['Age'].median())

dp=df['Age'].fillna(df['Age'].median())
print("\nAfter Replacing with Mean:\n\n",dp.head(7))
```

Output:

```
Before Replacing:

 0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5     NaN
6    54.0
Name: Age, dtype: float64

Median of Age Column: 28.0

After Replacing with Mean:

 0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5    28.0
6    54.0
Name: Age, dtype: float64
```

38. **Replacing with Mode value**

Code:

```
df=p.read_csv("C:\\Users\\Desktop\\Data\\Titanic-Dataset.csv")

print("Before Replacing:\n\n",df['Age'].head(7))

print("\nMode of Age Column:",df['Age'].mode()[0])

dp=df['Age'].fillna(df['Age'].mode()[0])
print("\nAfter Replacing with Mode:\n\n",dp.head(7))
```

Output:

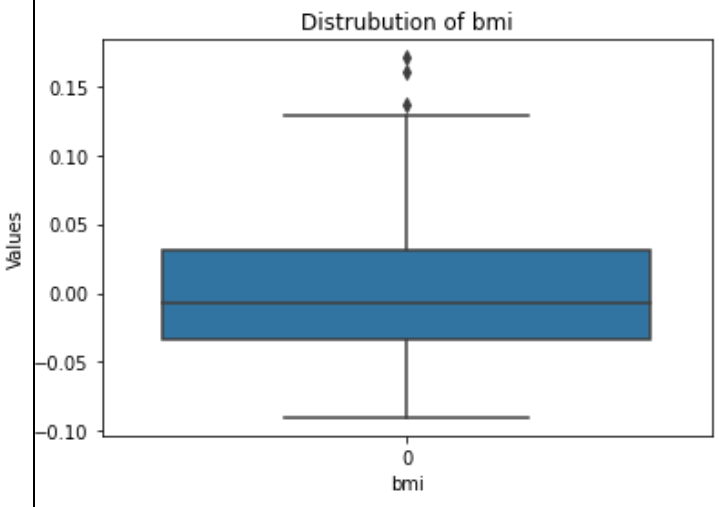
```
Before Replacing:

 0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5     NaN
6    54.0
Name: Age, dtype: float64

Mode of Age Column: 24.0

After Replacing with Mode:

 0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5    24.0
6    54.0
Name: Age, dtype: float64
```

39. **Univariate Outliers**

Code:

```
from sklearn.datasets import load_diabetes
import matplotlib.pyplot as m
import seaborn as s

dp=load_diabetes()
col_n =dp.feature_names
df= p.DataFrame(dp.data);df.columns = col_n

#Visualizing of Outliers
s.boxplot(df['bmi'])
m.ylabel('Values');m.xlabel('bmi');m.title('Distrubution of bmi')
m.show()

#IQR
q1=df['bmi'].quantile(0.25)
q3=df['bmi'].quantile(0.75)
iqr=q3-q1

#Floor and Capping
flo=q1-1.5*iqr
cap=q3+1.5*iqr
out=df[(df.bmi<=flo)|(df.bmi>=cap)]
print("Outliers:\n",out)
```

Output:



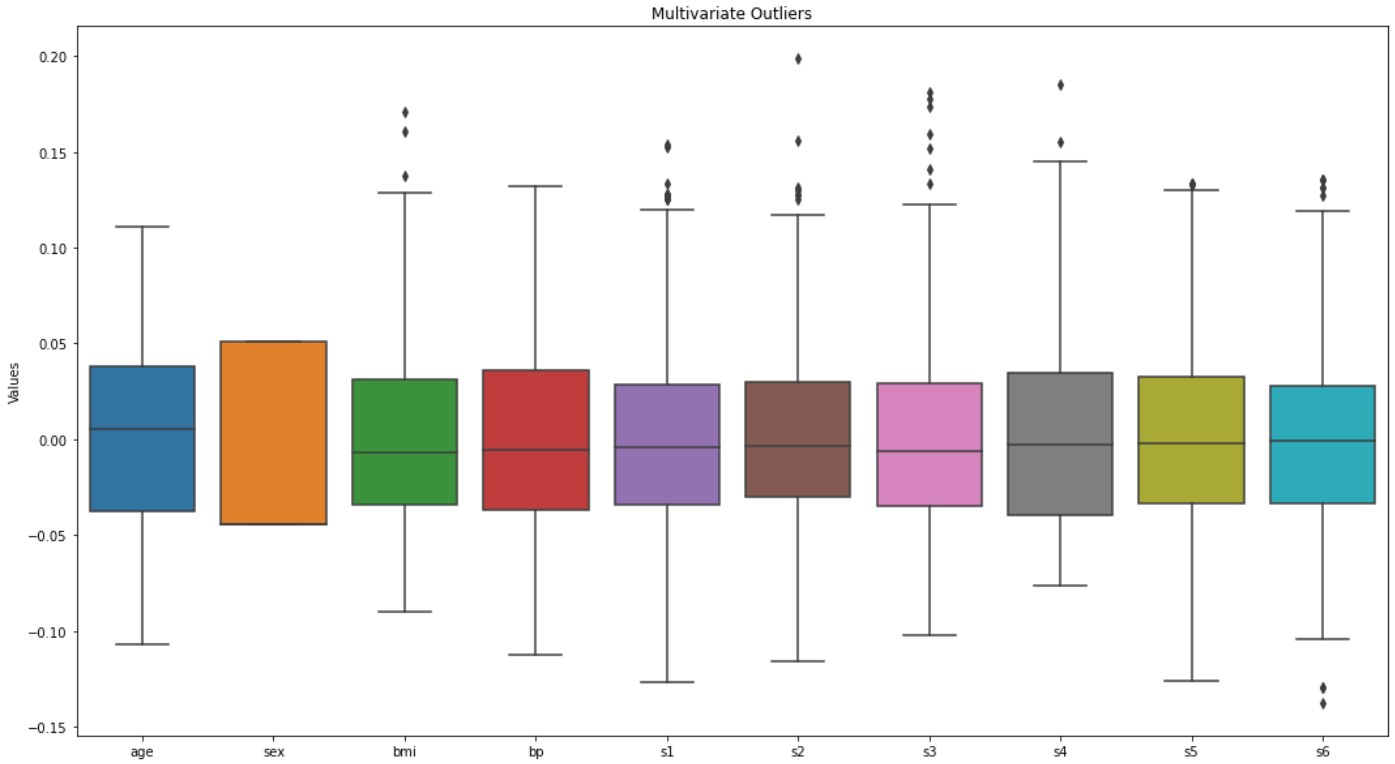

```
Outliers:
          age       sex       bmi        bp        s1        s2        s3  \
256 -0.049105 -0.044642  0.160855 -0.046985 -0.029088 -0.019790 -0.047082
366 -0.045472  0.050680  0.137143 -0.015999  0.041086  0.031880 -0.043401
367 -0.009147  0.050680  0.170555  0.014987  0.030078  0.033759 -0.021311

          s4        s5        s6
256  0.034309  0.028020  0.011349
366  0.071210  0.071019  0.048628
367  0.034309  0.033654  0.032059
```

40. **Multivariate Outliers**

Code:

```
from sklearn.datasets import load_diabetes
import matplotlib.pyplot as m
import seaborn as s

dp=load_diabetes()
col_n =dp.feature_names
df= p.DataFrame(dp.data)
df.columns = col_n

m.figure(figsize=(18,10))
s.boxplot(data=df)
m.title('Multivariate Outliers')
m.ylabel('Values')
m.show()
```

Output:

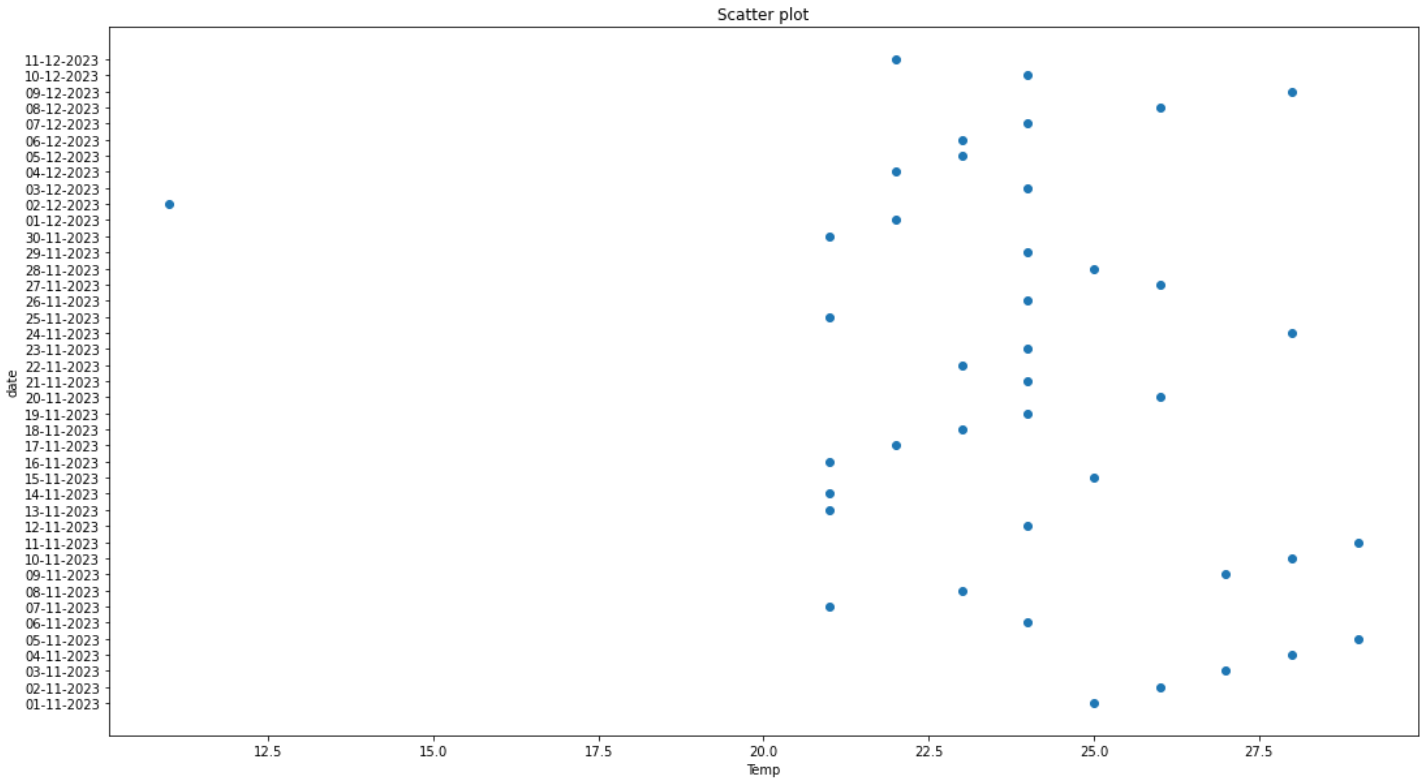


41. **Time series outlier detection**

Code:

```
import numpy as n
import pandas as p
import matplotlib.pyplot as m
df=p.read_csv("C:\\Users\\reddy\\Desktop\\Data\\te.csv")
x=df.temp
y=df.day
m.figure(figsize=(18,10))
m.scatter(x,y,label="values of x & y")
m.xlabel("Temp")
m.ylabel("date")
m.title("Scatter plot")
m.show()
```

Output:



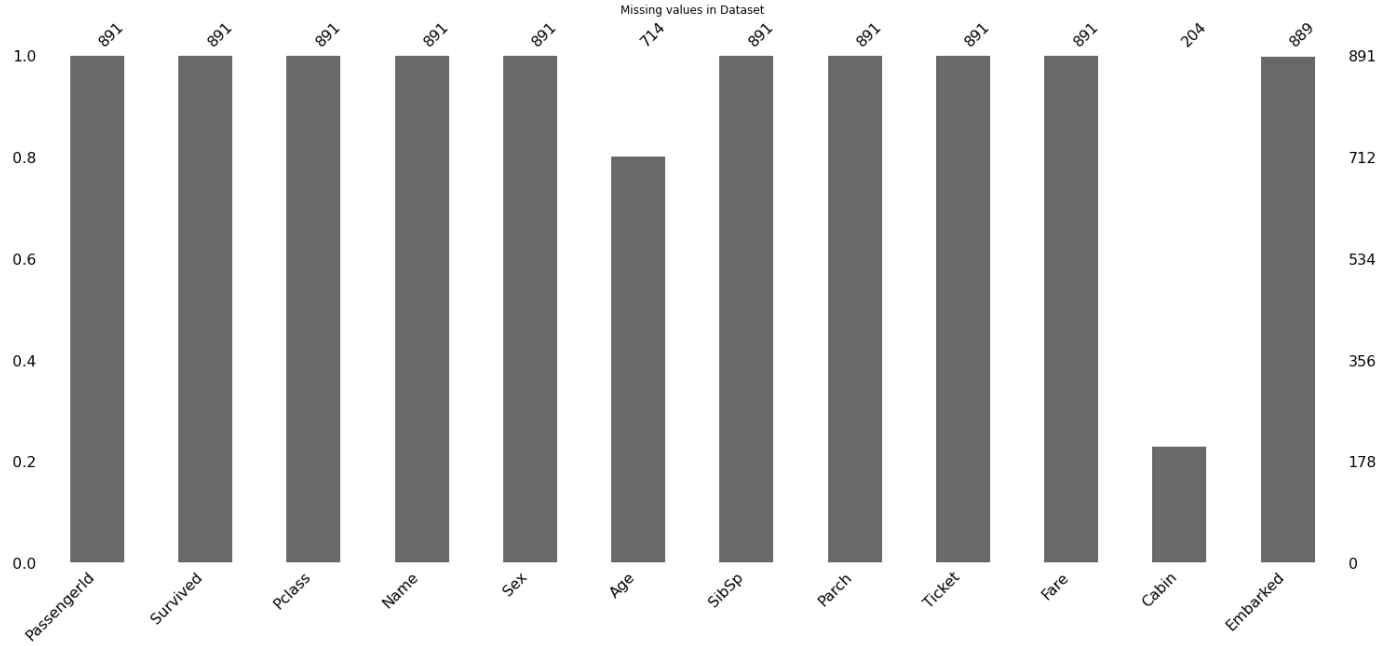
42. **Titanic Dataset Perform:**

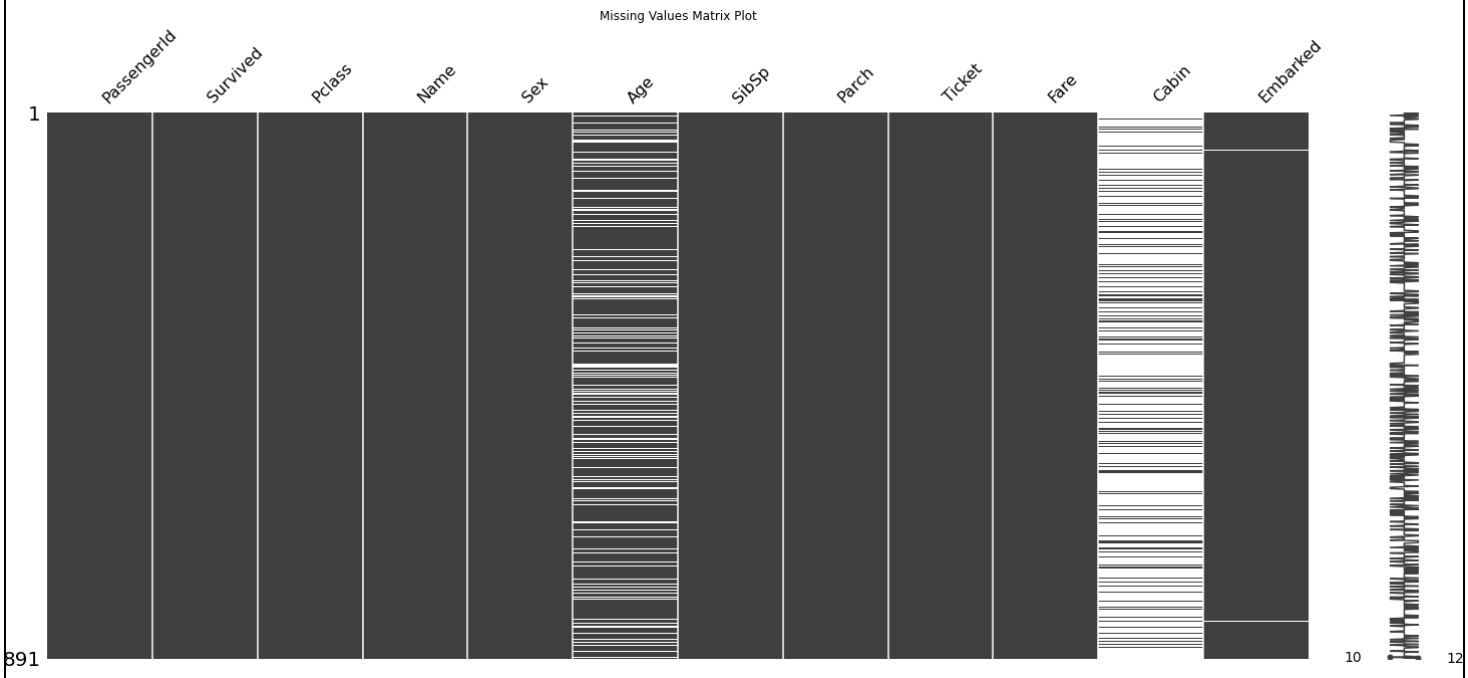
- **Visualize missing values as bar plot and matrix plot**
- **Handle Missing values by deleting data objects and attributes**
- **Impute the missing values**

Code:

```
import missingno as ms
ti_da=p.read_csv("C:\\Users\\Desktop\\Data\\Titanic-Dataset.csv")

#Box Plot
ms.bar(ti_da)
m.title("Missing values in Dataset")
m.show()

#Matrix Plot
ms.matrix(ti_da)
m.title('Missing Values Matrix Plot')
m.show()

#Removing Null Objects
print("Before Droping Objects:\n")
ti_da.info()
ti_d=ti_da.dropna(axis=0)
print("\n\nAfter Droping objects:\n")
ti_d.info()

#Removing Null Attributes
print("\nBefore Droping Attributes:\n")
ti_da.info()
ti=ti_da.dropna(axis=1)
print("\n\nAfter Droping Attributes:\n")
ti.info()

#Imputing Missing value of Age column through Mean
print("\n\nAge column before imputing:\n")
ti_da['Age'].info()
ti_ag=ti_da['Age'].fillna(ti_da['Age'].mean())
print("\n\nAfter Imputing:\n")
ti_ag.info()
```

Output:


Missing values in Dataset
891
891
891
891
891
714
891
891
891
891
204
889
1.0
0.8
0.6
0.4
0.2
0.0
891
712
534
356
178
0
PassengerId
Survived
Pclass
Name
Sex
Age
SibSp
Parch
Ticket
Fare
Cabin
Embarked


Missing Values Matrix Plot
PassengerId
Survived
Pclass
Name
Sex
Age
SibSp
Parch
Ticket
Fare
Cabin
Embarked
1
891
10
12

Before Droping Objects:

```
<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
```

After Droping objects:

```
<class 'pandas.core.frame.DataFrame'>
Index: 183 entries, 1 to 889
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  183 non-null    int64
 1   Survived     183 non-null    int64
 2   Pclass       183 non-null    int64
 3   Name         183 non-null    object
 4   Sex          183 non-null    object
 5   Age          183 non-null    float64
 6   SibSp        183 non-null    int64
 7   Parch        183 non-null    int64
 8   Ticket       183 non-null    object
 9   Fare         183 non-null    float64
 10  Cabin        183 non-null    object
 11  Embarked     183 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 18.6+ KB
```

Before Droping Attributes:

```
<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
```

After Droping Attributes:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 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   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Ticket       891 non-null    object
 8   Fare         891 non-null    float64
dtypes: float64(1), int64(5), object(3)
memory usage: 62.8+ KB
```

Age column before imputing:

```
<class 'pandas.core.series.Series'>
RangeIndex: 891 entries, 0 to 890
Series name: Age
Non-Null Count  Dtype
--------------  -----
714 non-null    float64
dtypes: float64(1)
memory usage: 7.1 KB
```

After Imputing:

```
<class 'pandas.core.series.Series'>
RangeIndex: 891 entries, 0 to 890
Series name: Age
Non-Null Count  Dtype
--------------  -----
891 non-null    float64
dtypes: float64(1)
memory usage: 7.1 KB
```

43. **For Credit dataset**

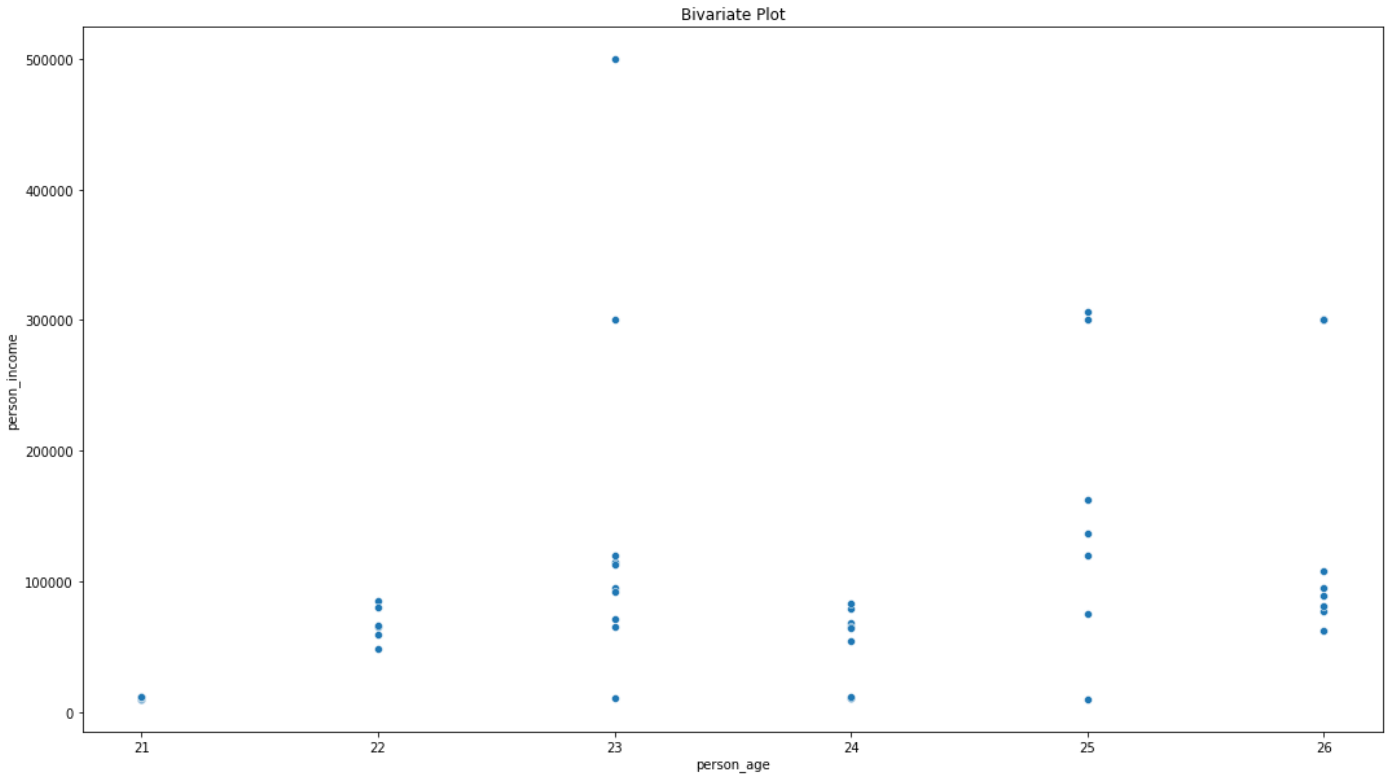
- **Spot outliers in income using bivariate plot**
- **Spot outliers in any feature using boxplot**
- **Detect outliers in any one feature using IQR method**
- **Treat outliers using Imputation [Mean, Median, Zero]**

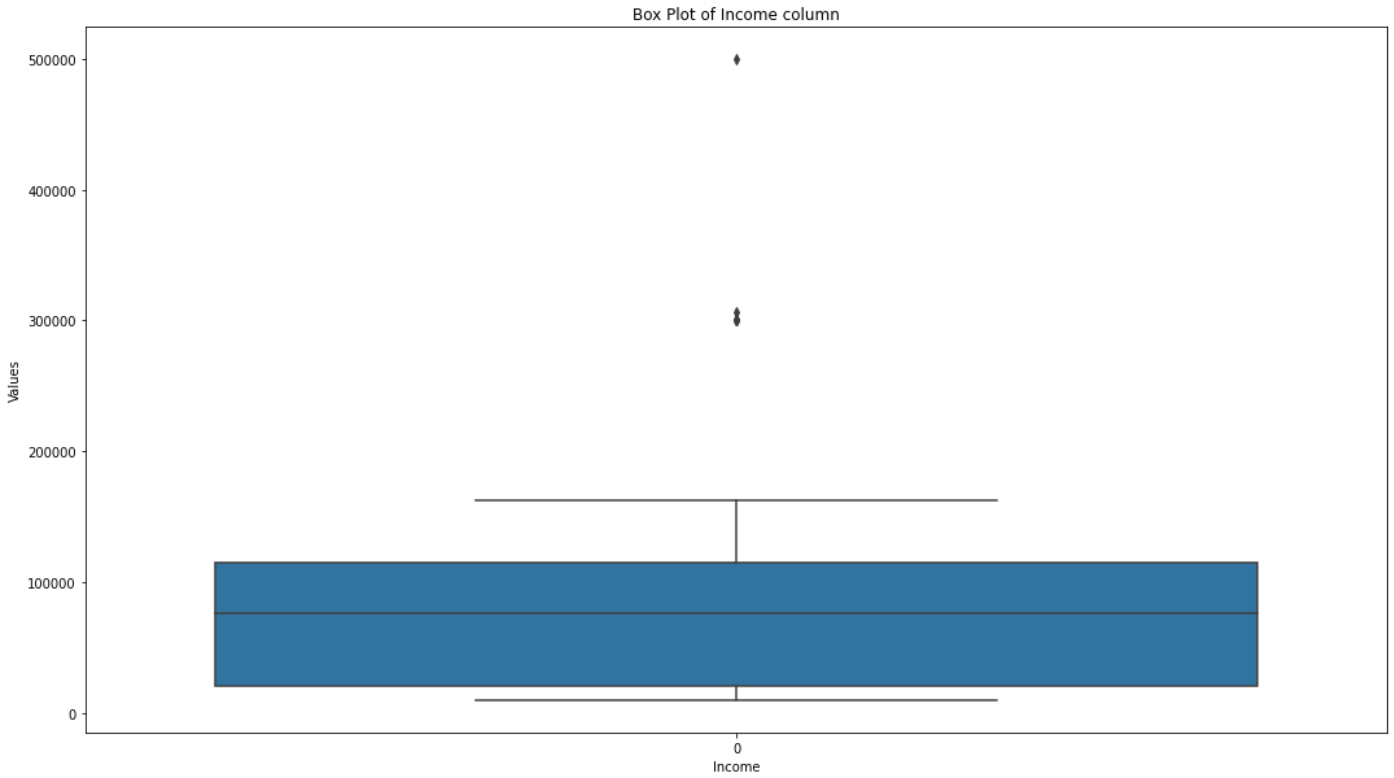
Code:

```
import pandas as p
import matplotlib.pyplot as m
import seaborn as s

da=p.read_csv("C:\\Users\\Desktop\\Data\\credit_risk_dataset.csv").head(5
0)

#Bivariate Plot
m.figure(figsize=(18,10))
s.scatterplot(x=da['person_age'],y=da['person_income'],data=da)
m.title("Bivariate Plot")
m.show()

#Box Plot
m.figure(figsize=(18,10))
s.boxplot(da['person_income'])
m.xlabel('Income');m.ylabel('Values')
m.title("Box Plot of Income column")
m.show()

#Detect Outliers using IQR Method
inc=da['person_income']
q1=inc.quantile(0.25)
q3=inc.quantile(0.75)
iqr=q3-q1
low=q1-1.5*iqr
hig=q3+1.5*iqr

out=(inc <= low) | (inc>= hig)
print("Outliers:\n",out.sum())

#Impute Outliers using Mean
mean_in=inc[(inc >= low) & (inc <= hig)].mean()
da.loc[out, 'person_income'] = mean_in

m.figure(figsize=(18,10))
m.boxplot(da['person_income'])
m.title("After Imputing with mean")
m.show()

# #Impute with Median
da=p.read_csv("C:\\Users\\Desktop\\Data\\credit_risk_dataset.csv").head(5
0)
```

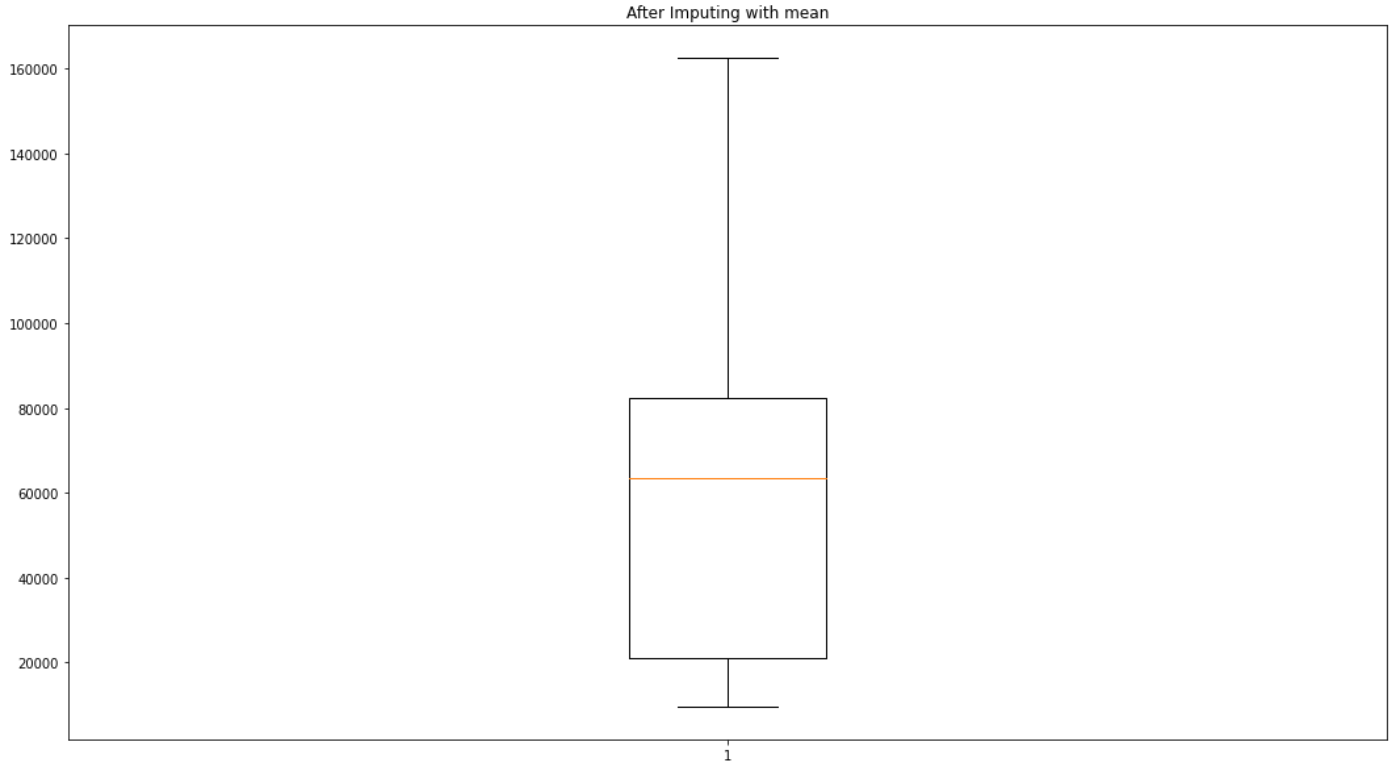
```
inc=da['person_income']
out1= (inc <= low) | (inc >= hig)
print("Outliers:\n",out1.sum())

medi=inc[(inc>= low) & (inc<= hig)].median()
da.loc[out1, 'person_income'] = medi

m.figure(figsize=(18,10))
m.boxplot(da['person_income'])
m.title("After Imputing with median")
m.show()

#Impute with Zero
da=p.read_csv("C:\\Users\\Desktop\\Data\\credit_risk_dataset.csv").head(5
0)
inc=da['person_income']
out2=(inc <= low) | (inc>= hig)
print("Outliers:\n",out2.sum())

da.loc[out2, 'person_income'] = 0

m.figure(figsize=(18,10))
m.boxplot(da['person_income'])
m.title("After Imputing with Zero [0]")
m.show()
```

Output:

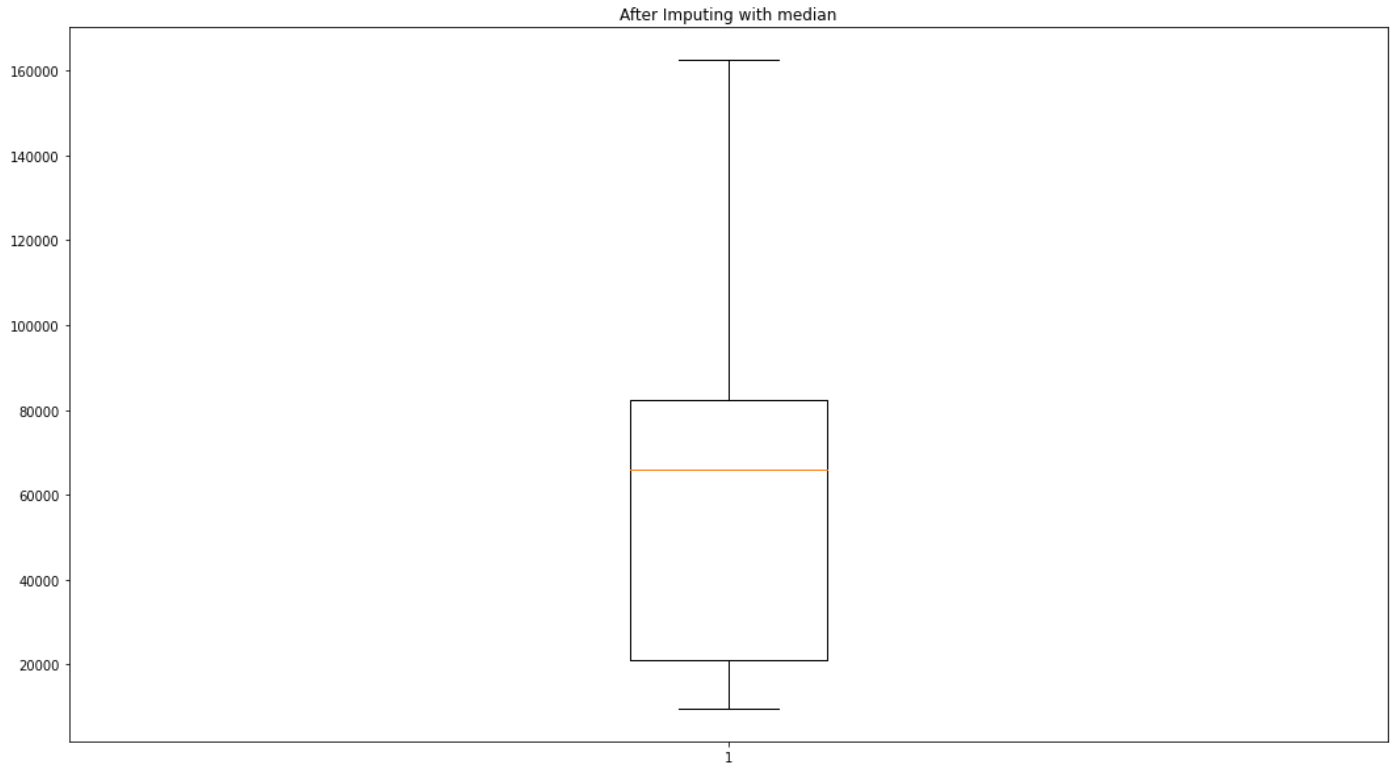




Outliers: 8



Outliers: 8


After Imputing with median
160000
140000
120000
100000
80000
60000
40000
20000
1

Outliers: 8


After Imputing with Zero [0]
160000
140000
120000
100000
80000
60000
40000
20000
0
1

Week-06

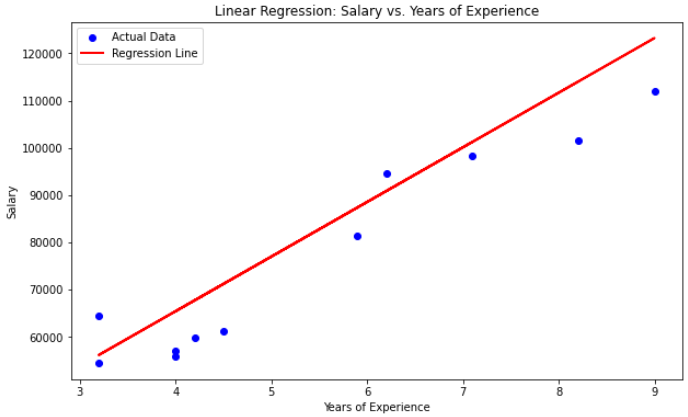
44. Salary Prediction according to Experience.

Code:

```
import pandas as p
import matplotlib.pyplot as m
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

da=p.read_csv("C:\\Users\\Desktop\\Data\\Salary_LR.csv")
da.info()
da.dropna(axis=0,inplace=True)

x=da['YearsExperience'].values.reshape(-1,1)
y=da['Salary']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
random_state=42)
model = LinearRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

m.figure(figsize=(10, 6))
m.scatter(x_test, y_test, color='blue', label='Actual Data')
m.plot(x_test, y_pred, color='red', linewidth=2, label='Regression Line')
m.title('Linear Regression: Salary vs. Years of Experience')
m.xlabel('Years of Experience');m.ylabel('Salary')
m.legend();m.show()

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\nMean Squared Error:", mse)
print("\nR-squared:", r2)

Expe= [[3]]  # Replace with the desired experience value
Sal = model.predict(Expe)
print(f"Predicted Salary for {Expe[0][0]} Year Experience : {Sal[0]}")
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36 entries, 0 to 35
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   YearsExperience  36 non-null     float64
 1   Salary           35 non-null     float64
dtypes: float64(2)
memory usage: 708.0 bytes


Mean Squared Error: 67193846.55620855

R-squared: 0.8414757489217368
Predicted Salary for 3 Year Experience : 53828.25216352839
```



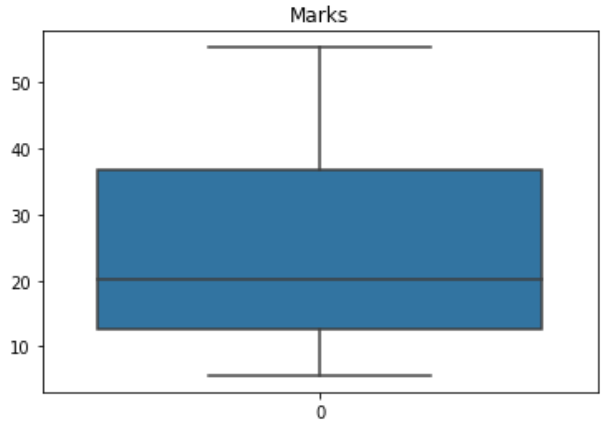
45. **Marks Prediction according to Study Hours.**

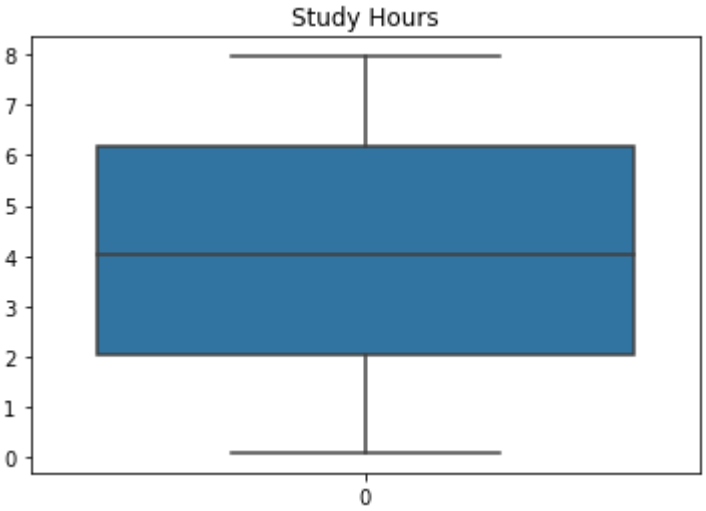
Code:

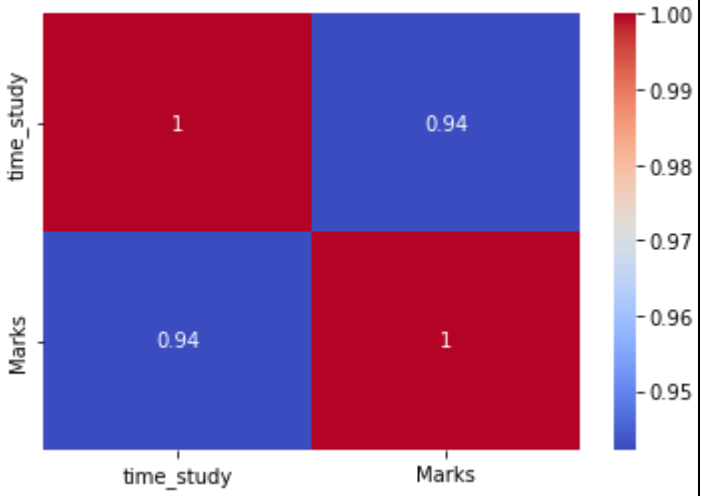
```
import pandas as p
import matplotlib.pyplot as m
import seaborn as s
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

da=p.read_csv("C:\\Users\\Desktop\\Data\\Stud_marks.csv")
print(da.isna().sum())

s.boxplot(da['Marks']);m.title('Marks')
m.show()
s.boxplot(da['time_study']);m.title('Study Hours')
m.show()
s.heatmap(data=da.corr(), annot=True, cmap='coolwarm')
m.show()

X = da['time_study'].values.reshape(-1,1)
y = da['Marks']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R-squared:", r2)

m.figure(figsize=(10, 6))
m.scatter(X_test, y_test, color='blue', label='Actual Data')
m.plot(X_test, y_pred, color='red', linewidth=2, label='Regression Line')
m.title('Linear Regression: Salary vs. Years of Experience')
m.xlabel('Years of Experience');m.ylabel('Salary')
m.legend()
m.show()

Hours= [[20]]# Replace with the desired experience value
predicted_marks = model.predict(Hours)
print(f"Predicted Marks for {Hours[0][0]} Hours : {predicted_marks[0]}")
```

Output:

```
time_study    0
Marks         0
dtype: int64
```








```
Mean Squared Error: 25.23674562363223
R-squared: 0.9040228286990537
```


```
Predicted Marks for 20 Hours : 109.62199613197404
```

46. **Boston housing price**

a. Preprocessing & Exploration

Code:

```
import pandas as p
import seaborn as s
import matplotlib.pyplot as m

da=p.read_csv("C:\\Users\\Desktop\\Data\\boston.csv")
da.info()

print(da.describe())

m.figure(figsize=(18,10))
s.heatmap(data=da.corr(),annot=True,cmap='coolwarm')
m.show()

m.figure(figsize=(18,10))
s.histplot(data=da,x='MEDV',bins=30,kde=True)
m.title('Distribution of Pricing')
m.show()

m.figure(figsize=(18,10))
s.scatterplot(data=da,x='RM',y='MEDV')
m.title('Number of rooms to pricing')
m.show()

m.figure(figsize=(18,10))
s.scatterplot(data=da,x='LSTAT',y='MEDV')
m.title('Lower status Population and Pricing')
m.show()
```

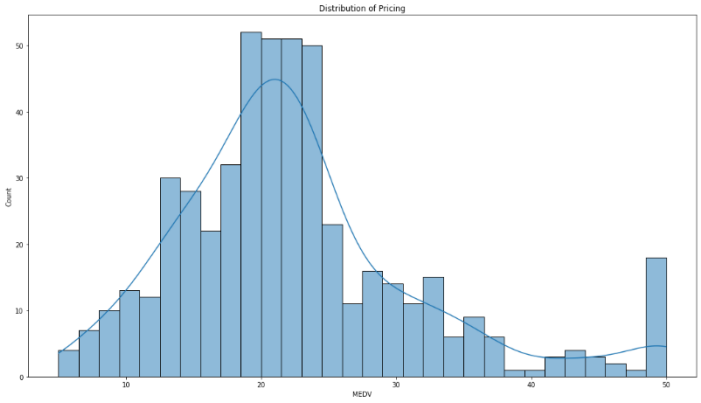
Output:

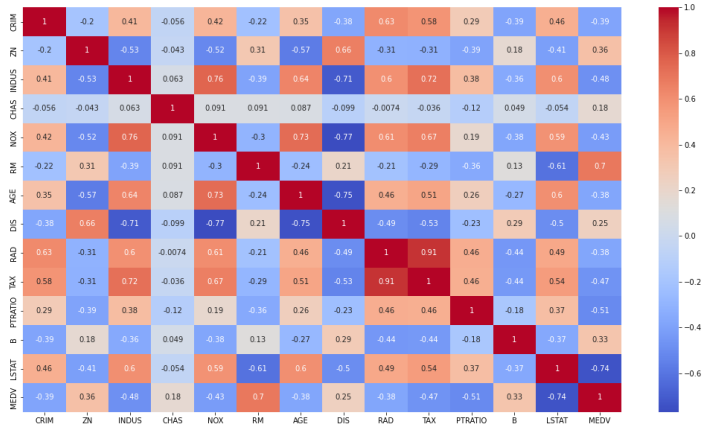
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     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    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

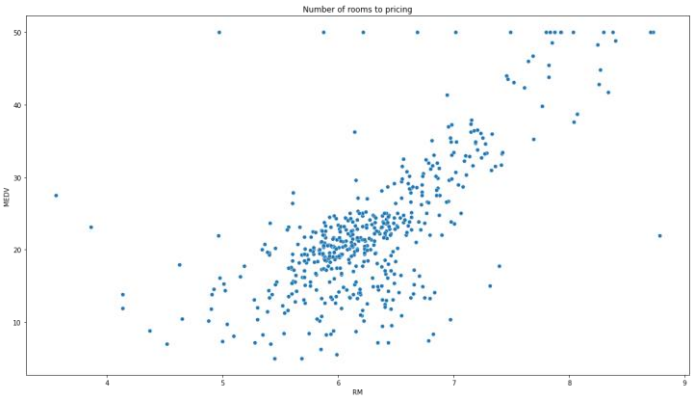
	CRIM	ZN	INDUS	CHAS	NOX	RM
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000

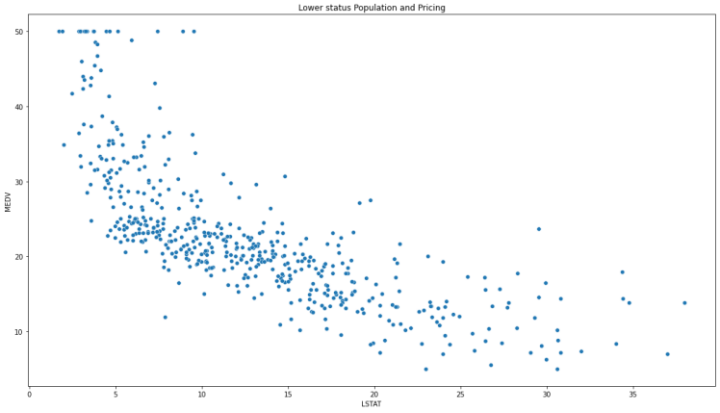
	AGE	DIS	RAD	TAX	PTRATIO	B
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032
std	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864
min	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000
25%	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500
50%	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000
75%	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000
max	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000

	LSTAT	MEDV
count	506.000000	506.000000
mean	12.653063	22.532806
std	7.141062	9.197104
min	1.730000	5.000000
25%	6.950000	17.025000
50%	11.360000	21.200000
75%	16.955000	25.000000
max	37.970000	50.000000









b. **Splitting**

```
import numpy as n
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

X = da.drop('MEDV', axis=1)
y = da['MEDV']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
print("Train set:", X_train.shape, y_train.shape)
print("Test set:", X_test.shape, y_test.shape)

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

rmse = n.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r2)
```

Output:

```
Train set: (354, 13) (354,)
Test set: (152, 13) (152,)
Root Mean Squared Error: 4.638689926172827
R-squared: 0.7112260057484925
```

47. Cricket match result

a. Data Preprocessing

Code:

```
import pandas as p
da=p.read_csv("C:\\Users\\Desktop\\Data\\matches.csv")
da.info()

#PRE_PROCESSING
miss=da.isna().sum()
print("\nMissing Values:\n",miss)

da.drop('umpire3', axis=1, inplace=True)
da.dropna(axis=0,inplace=True)
# da.drop('date',axis=1,inplace=True)

print("\nMissing Values:\n",da.isna().sum())
```

Output:

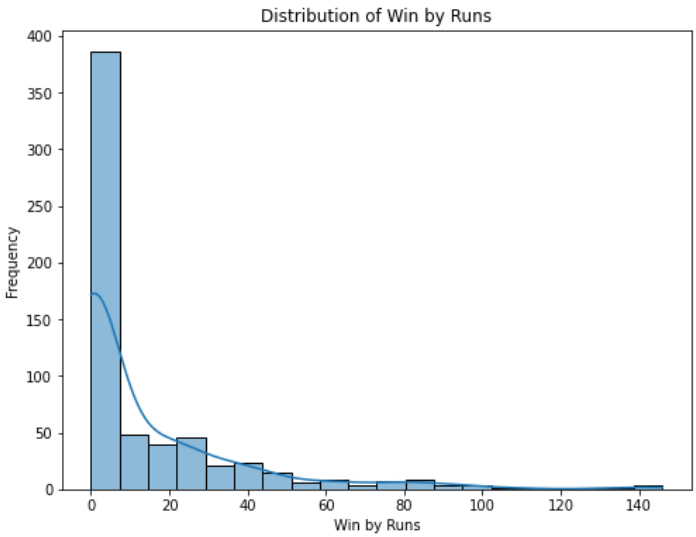
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 636 entries, 0 to 635
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               636 non-null    int64
 1   season           636 non-null    int64
 2   city             629 non-null    object
 3   date             636 non-null    object
 4   team1            636 non-null    object
 5   team2            636 non-null    object
 6   toss_winner      636 non-null    object
 7   toss_decision    636 non-null    object
 8   result           636 non-null    object
 9   dl_applied       636 non-null    int64
 10  winner           633 non-null    object
 11  win_by_runs      636 non-null    int64
 12  win_by_wickets   636 non-null    int64
 13  player_of_match  633 non-null    object
 14  venue            636 non-null    object
 15  umpire1          635 non-null    object
 16  umpire2          635 non-null    object
 17  umpire3          0 non-null      float64
dtypes: float64(1), int64(5), object(12)
memory usage: 89.6+ KB
Missing Values:
 id                 0
season             0
city               7
date               0
team1              0
team2              0
toss_winner        0
toss_decision      0
result             0
dl_applied         0
winner             3
win_by_runs        0
win_by_wickets     0
player_of_match    3
venue              0
umpire1            1
umpire2            1
umpire3          636
dtype: int64
Missing Values:
 id                 0
season             0
city               0
date               0
team1              0
team2              0
toss_winner        0
toss_decision      0
result             0
dl_applied         0
winner             0
win_by_runs        0
win_by_wickets     0
player_of_match    0
venue              0
umpire1            0
umpire2            0
dtype: int64
```

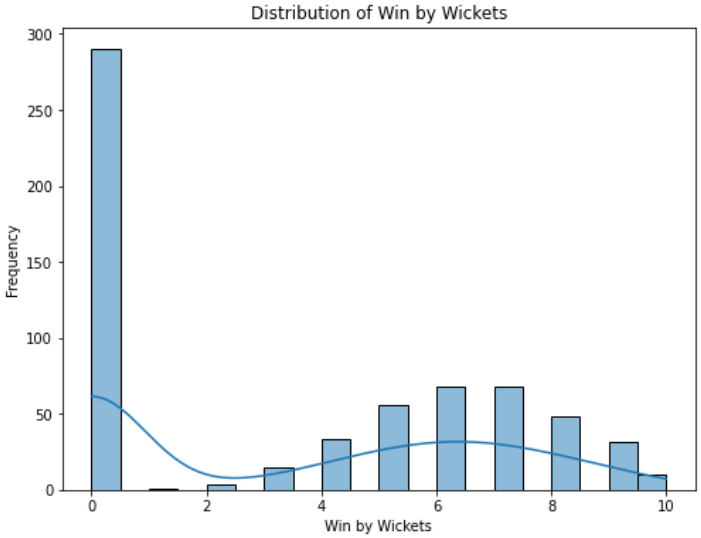
b. Data Exploration

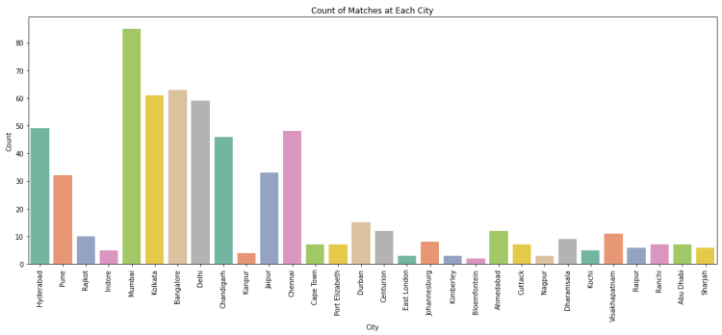
Code:

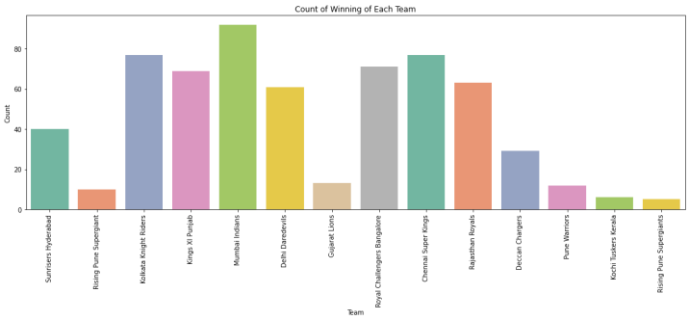
```
import matplotlib.pyplot as m
import seaborn as s

# Function to create a histogram
def histo(da, col, title, xlabel):
    m.figure(figsize=(8, 6))
    s.histplot(da[col], bins=20, kde=True)
    m.title(title)
    m.xlabel(xlabel)
    m.ylabel('Frequency')
    m.show()

# Function to create a count plot
def count_plo(da, x, title, xlabel):
    m.figure(figsize=(15, 7))
    s.countplot(x=x, data=da, palette='Set2')
    m.title(title)
    m.xlabel(xlabel)
    m.ylabel('Count')
    m.xticks(rotation=90)
    m.tight_layout()
    m.show()

# Function to create a stacked bar chart
def sta_bar(da, x, y, title, xlabel, ylabel, legend_title=None):
    dp = da.groupby([x, y]).size().unstack()
    dp.plot(kind='bar', stacked=True, figsize=(10, 6))
    m.title(title)
    m.xlabel(xlabel)
    m.ylabel(ylabel)
    if legend_title:
        m.legend(title=legend_title)
    m.show()


#DATA_EXPLORATION
histo(da, 'win_by_runs', 'Distribution of Win by Runs', 'Win by Runs')
histo(da, 'win_by_wickets', 'Distribution of Win by Wickets', 'Win by Wickets')
count_plo(da, 'city', 'Count of Matches at Each City', 'City')
count_plo(da, 'winner', 'Count of Winning of Each Team', 'Team')
sta_bar(da, 'toss_decision', 'result', 'Toss Decision vs. Match Result', 'Toss
Decision', 'Count', legend_title='Match Result')
sta_bar(da, 'season', 'toss_decision', 'Season-wise Toss Decision', 'Season',
'Count', legend_title='Toss Decision')
```

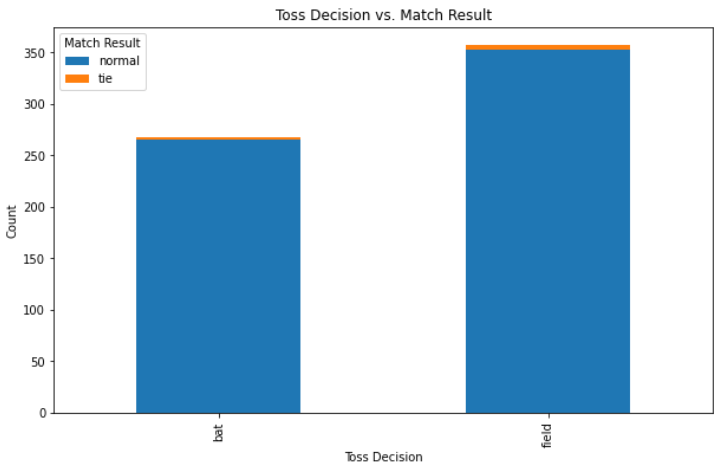
Output:

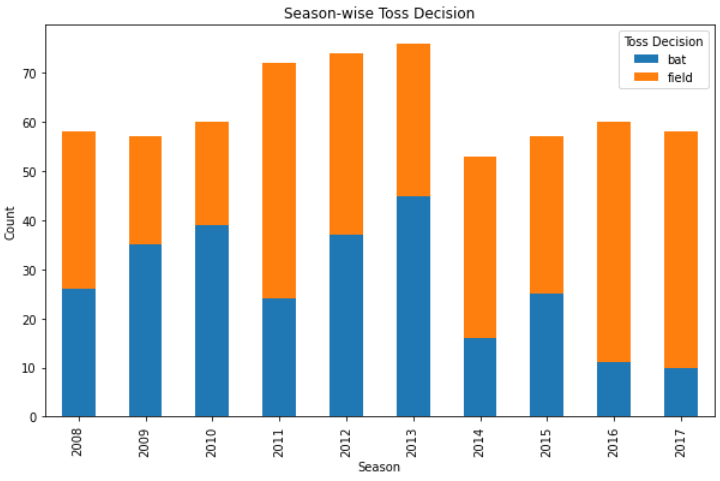

Distribution of Win by Runs
Frequency
Win by Runs


Distribution of Win by Wickets
Frequency
Win by Wickets


Count of Matches at Each City
Count
City


Count of Winning of Each Team
Count
Team


Toss Decision vs. Match Result
Match Result
normal
tie
Count
bat
field
Toss Decision


Season-wise Toss Decision
Toss Decision
bat
field
Count
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
Season

c. **Splitting**

Code:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

#Transformation
categorical_columns = ['city', 'team1', 'team2', 'toss_winner',
'toss_decision', 'result', 'player_of_match', 'venue', 'umpire1',
'umpire2','winner','date']
label_encoders = {}

for column in categorical_columns:
    label_encoders[column] = LabelEncoder()
    da[column] = label_encoders[column].fit_transform(da[column])

X = da.drop('winner', axis=1)
y = da['winner']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
print("Train set:", X_train.shape, y_train.shape)
print("Test set:", X_test.shape, y_test.shape)
```

Output:

```
Train set: (437, 16) (437,)
Test set: (188, 16) (188,)
```

48. **Performance of a cricket player**

a. **Preprocessing**

Code:

```
import pandas as p
da=p.read_csv("C:\\Users\\Desktop\\Data\\Ms_Dhoni.csv")
da.info()

#PRE_PROCESSING
miss=da.isna().sum()
print("\nMissing Values:\n",miss)
da.dropna(inplace=True)
print("\nMissing Values:\n",da.isna().sum())
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16 entries, 0 to 15
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Year           15 non-null     float64
 1   Matches        15 non-null     float64
 2   Innings        15 non-null     float64
 3   N.O.           15 non-null     float64
 4   Runs           15 non-null     float64
 5   Highest Score  15 non-null     float64
 6   Average        15 non-null     float64
 7   Strike Rate    15 non-null     float64
 8   100            15 non-null     float64
 9   50             15 non-null     float64
 10  Fours          15 non-null     float64
 11  Sixes          15 non-null     float64
 12  Catches Taken  15 non-null     float64
 13  Stumpings      15 non-null     float64
dtypes: float64(14)
memory usage: 1.9 KB
```

```
Missing Values:
 Year             1
Matches           1
Innings           1
N.O.              1
Runs              1
Highest Score     1
Average           1
Strike Rate       1
100               1
50                1
Fours             1
Sixes             1
Catches Taken     1
Stumpings         1
dtype: int64
```

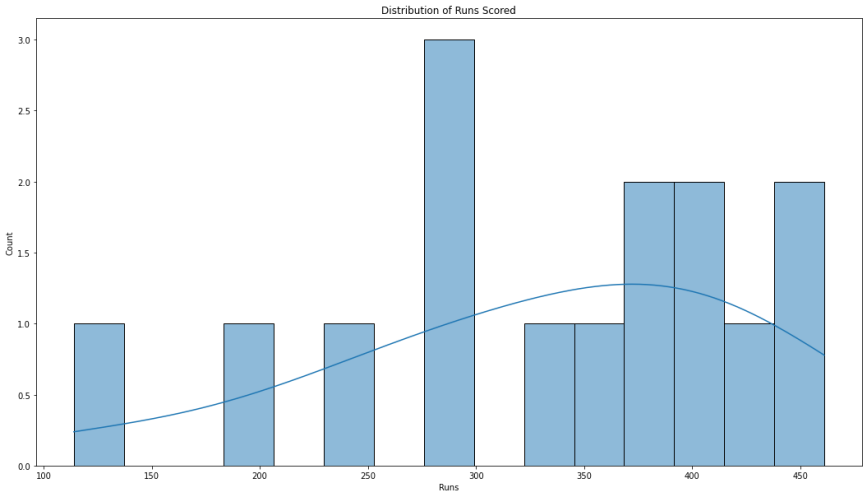
```
Missing Values:
 Year             0
Matches           0
Innings           0
N.O.              0
Runs              0
Highest Score     0
Average           0
Strike Rate       0
100               0
50                0
Fours             0
Sixes             0
Catches Taken     0
Stumpings         0
dtype: int64
```

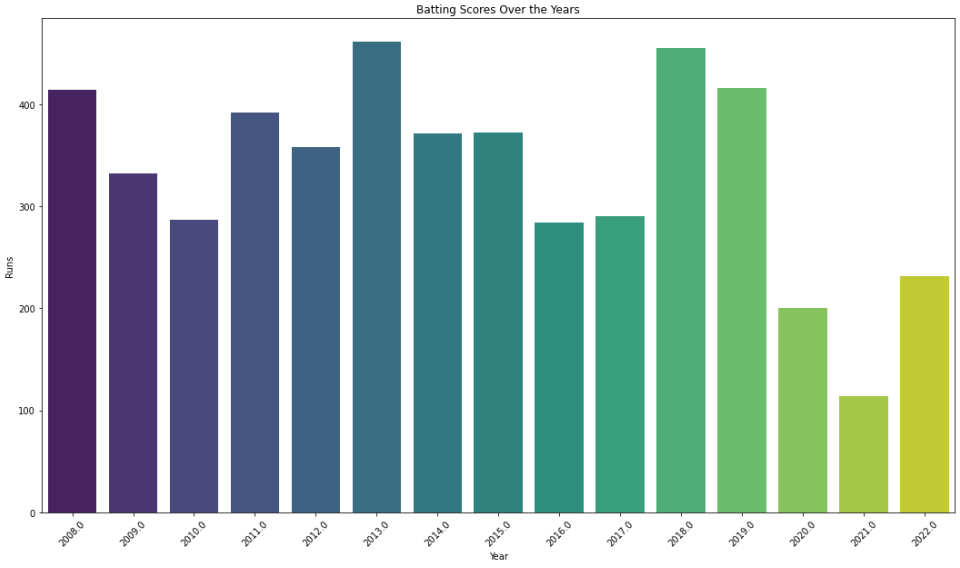
b. **Exploration**

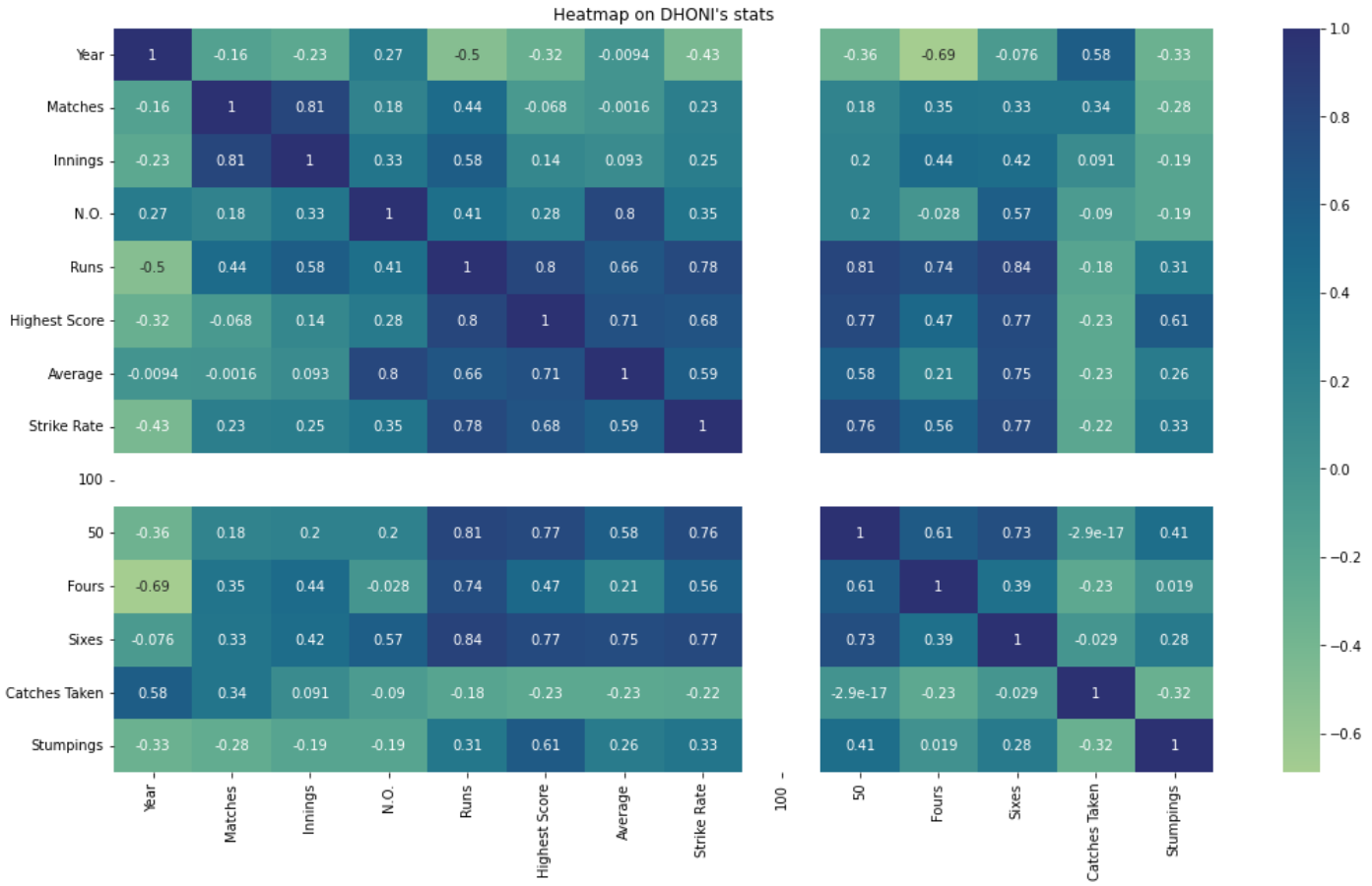
Code:

```
import matplotlib.pyplot as m
import seaborn as s
m.figure(figsize=(18,10))
s.histplot(da['Runs'], bins=15, kde=True)
m.title('Distribution of Runs Scored')
m.show()
m.figure(figsize=(18,10))
s.barplot(data=da,x='Year',y='Runs',palette='viridis')
m.title('Batting Scores Over the Years')
m.xlabel('Year');m.ylabel('Runs')
m.xticks(rotation=45);m.show()
m.figure(figsize=(18,10))
s.heatmap(data=da.corr(),annot=True,cmap='crest')
m.title("Heatmap on DHONI's stats")
m.show()
```

Output:


Distribution of Runs Scored
Count
Runs


Batting Scores Over the Years
Runs
Year


Heatmap on DHONI's stats
Year 1 -0.16 -0.23 0.27 -0.5 -0.32 -0.0094 -0.43 -0.36 -0.69 -0.076 0.58 -0.33
Matches -0.16 1 0.81 0.18 0.44 -0.068 -0.0016 0.23 0.18 0.35 0.33 0.34 -0.28
Innings -0.23 0.81 1 0.33 0.58 0.14 0.093 0.25 0.2 0.44 0.42 0.091 -0.19
N.O. 0.27 0.18 0.33 1 0.41 0.28 0.8 0.35 0.2 -0.028 0.57 -0.09 -0.19
Runs -0.5 0.44 0.58 0.41 1 0.8 0.66 0.78 0.81 0.74 0.84 -0.18 0.31
Highest Score -0.32 -0.068 0.14 0.28 0.8 1 0.71 0.68 0.77 0.47 0.77 -0.23 0.61
Average -0.0094 -0.0016 0.093 0.8 0.66 0.71 1 0.59 0.58 0.21 0.75 -0.23 0.26
Strike Rate -0.43 0.23 0.25 0.35 0.78 0.68 0.59 1 0.76 0.56 0.77 -0.22 0.33
100
50 -0.36 0.18 0.2 0.2 0.81 0.77 0.58 0.76 1 0.61 0.73 -2.9e-17 0.41
Fours -0.69 0.35 0.44 -0.028 0.74 0.47 0.21 0.56 0.61 1 0.39 -0.23 0.019
Sixes -0.076 0.33 0.42 0.57 0.84 0.77 0.75 0.77 0.73 0.39 1 -0.029 0.28
Catches Taken 0.58 0.34 0.091 -0.09 -0.18 -0.23 -0.23 -0.22 -2.9e-17 -0.23 -0.029 1 -0.32
Stumpings -0.33 -0.28 -0.19 -0.19 0.31 0.61 0.26 0.33 0.41 0.019 0.28 -0.32 1
Year Matches Innings N.O. Runs Highest Score Average Strike Rate 100 50 Fours Sixes Catches Taken Stumpings

c. **Splitting**

Code:

```
from sklearn.model_selection import train_test_split

X = da.drop('Average', axis=1)
y = da['Average']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
print("Train set:", X_train.shape, y_train.shape)
print("Test set:", X_test.shape, y_test.shape)
```

Output:

```
Train set: (10, 13) (10,)
Test set: (5, 13) (5,)
```

49. **Crop yield**

a. **Preprocessing**

Code:

```
import pandas as p
da=p.read_csv("C:\\Users\\Desktop\\Data\\crop_production.csv")
da.info()

#PRE_PROCESSING
miss=da.isna().sum()
print("\nMissing Values:\n",miss)
da.dropna(inplace=True)
print("\nMissing Values:\n",da.isna().sum())
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 246091 entries, 0 to 246090
Data columns (total 7 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   State_Name     246091 non-null  object
 1   District_Name  246091 non-null  object
 2   Crop_Year      246091 non-null  int64
 3   Season         246091 non-null  object
 4   Crop           246091 non-null  object
 5   Area           246091 non-null  float64
 6   Production     242361 non-null  float64
dtypes: float64(2), int64(1), object(4)
memory usage: 13.1+ MB
```

```
Missing Values:
 State_Name          0
District_Name        0
Crop_Year            0
Season               0
Crop                 0
Area                 0
Production        3730
dtype: int64

Missing Values:
 State_Name        0
District_Name     0
Crop_Year         0
Season            0
Crop              0
Area              0
Production        0
dtype: int64
```

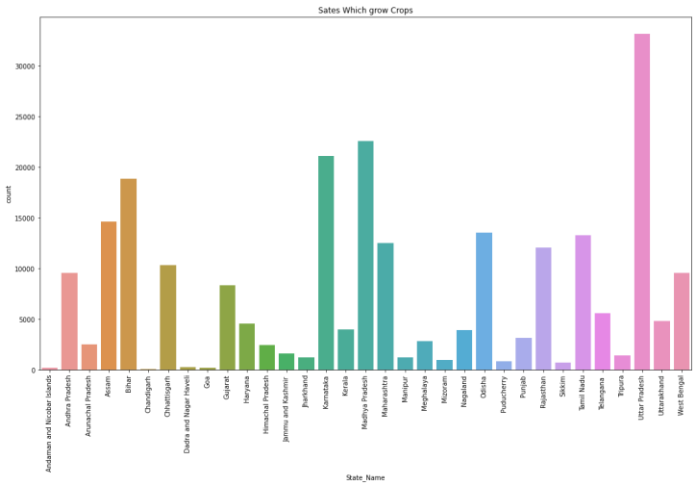
b. **Data Exploration**

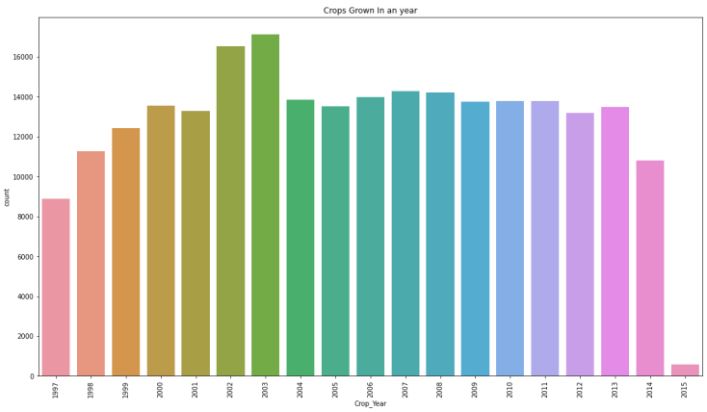
Code:

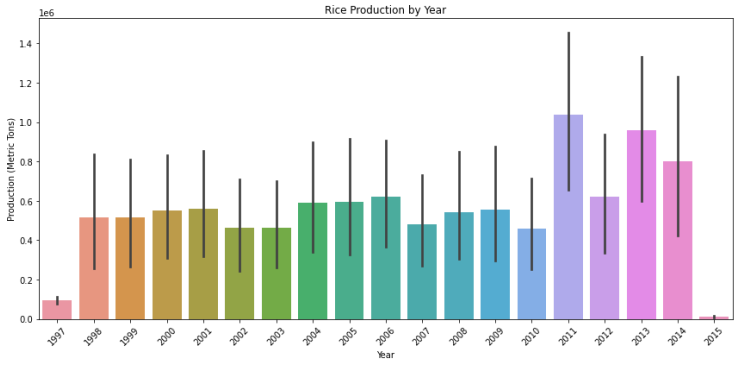
```
import matplotlib.pyplot as m
import seaborn as s

m.figure(figsize=(18,10))
s.countplot(data=da, x='State_Name')
m.title("Sates Which grow Crops")
m.xticks(rotation=90)
m.show()
```

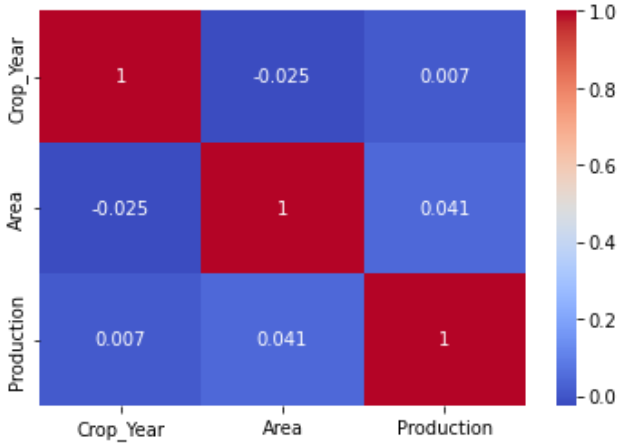
```
m.figure(figsize=(18,10))
s.countplot(data=da,x='Crop_Year')
m.title('Crops Grown In an year')
m.xticks(rotation=90)
m.show()

m.figure(figsize=(12, 6))
s.barplot(x=da['Crop_Year'], y=da['Production'],data=da)
m.title('Rice Production by Year')
m.xlabel('Year')
m.ylabel('Production (Metric Tons)')
m.xticks(rotation=45)
m.tight_layout()
m.show()

sn=da.select_dtypes(exclude=['object'])
s.heatmap(data=sn.corr(),annot=True,cmap='coolwarm')
m.show()
```

Output:









c. **Splitting**

Code:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

#Transformation
categorical_columns = ['State_Name','District_Name','Season','Crop']
label_encoders = {}

for column in categorical_columns:
    label_encoders[column] = LabelEncoder()
    da[column] = label_encoders[column].fit_transform(da[column])

X = da.drop('Production', axis=1)
y = da['Production']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
print("Train set:", X_train.shape, y_train.shape)
print("Test set:", X_test.shape, y_test.shape)
```

Output:

```
Train set: (169652, 6) (169652,)
Test set: (72709, 6) (72709,)
```

Week -07

50. Regression algorithms

- **Decision Tree Regressor**
- **Random Forest Regressor**
- **Support Vector Regression**

Code:

```
import pandas as p
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

# Breast cancer dataset
data = p.read_csv("C:\\Users\\Desktop\\Data\\brca1.csv")
data.info()
data.drop(["id"],axis=1,inplace=True)
M=data[data.diagnosis=="M"]
B=data[data.diagnosis=="B"]

data.diagnosis=[1 if i == "M" else 0 for i in data.diagnosis]
x=data.drop(["diagnosis"],axis=1)
y=data.diagnosis.values

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_s
tate=42)

from sklearn.tree import DecisionTreeRegressor
model=DecisionTreeRegressor()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)

print("\nAccuracy of the model using Decision tree regression algorithm
is ",r2_score(y_test,y_pred))

from sklearn.ensemble import RandomForestRegressor
model1 = RandomForestRegressor()
model1.fit(x_train,y_train)
y_pred1 = model1.predict(x_test)
print("\nAccuracy of the model using Random forest regression algorithm
is ",r2_score(y_test,y_pred1))

from sklearn.svm import SVR
model2 = SVR(kernel='rbf')
model2.fit(x_train,y_train)
y_pred2 = model2.predict(x_test)
print("\nAccuracy of the model using Support vector regression algorithm
is ",r2_score(y_test,y_pred2))
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
```

```
Accuracy of the model using Decision tree regression algorithm is  0.6481481481481481

Accuracy of the model using Random forest regression algorithm is  0.8534185185185185

Accuracy of the model using Support vector regression algorithm is  0.8114302960086689
```

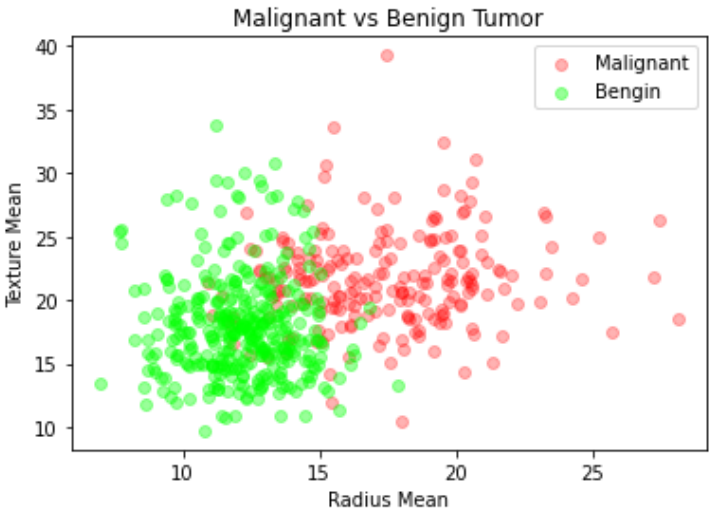
51. **Build decision tree-based model for Breast Cancer Wisconsin (diagnostic) dataset.[Classifier]**

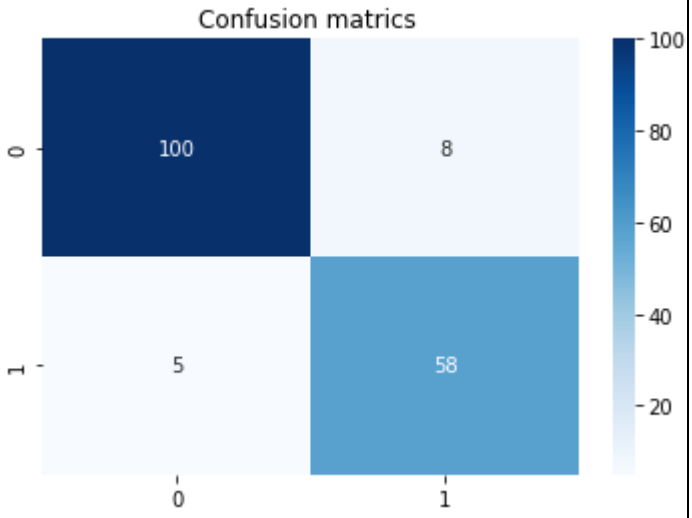
Code:

```
import numpy as n
import pandas as p
import seaborn as s
import matplotlib.pyplot as m
from sklearn.metrics import r2_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix

data = p.read_csv("C:\\Users\\Desktop\\Data\\brca1.csv")
data.info()
data.drop(["id"],axis=1,inplace=True)
M=data[data.diagnosis=="M"]
B=data[data.diagnosis=="B"]

m.title("Malignant vs Benign Tumor")
m.xlabel("Radius Mean"); m.ylabel("Texture Mean")
m.scatter(M.radius_mean,M.texture_mean,color='red',label='Malignant',alph
a=0.3)
m.scatter(B.radius_mean,B.texture_mean,color='lime',label='Bengin',alpha=
0.4)
m.legend(); m.show()

data.diagnosis=[1 if i == "M" else 0 for i in data.diagnosis]
x=data.drop(["diagnosis"],axis=1)
y=data.diagnosis.values

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_s
tate=42)

dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
y_pred=dt.predict(x_test)

#Confusion Matrix
cm = confusion_matrix(y_test,y_pred)
s.heatmap(cm,annot=True,fmt='d',cmap="Blues")
m.title("Confusion matrics ")
m.show()
print("Accuracy of the classifier model
is",accuracy_score(y_test,y_pred))
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      569 non-null    int64
 1   diagnosis               569 non-null    object
 2   radius_mean             569 non-null    float64
 3   texture_mean            569 non-null    float64
 4   perimeter_mean          569 non-null    float64
 5   area_mean               569 non-null    float64
 6   smoothness_mean         569 non-null    float64
 7   compactness_mean        569 non-null    float64
 8   concavity_mean          569 non-null    float64
 9   concave points_mean     569 non-null    float64
 10  symmetry_mean           569 non-null    float64
 11  fractal_dimension_mean  569 non-null    float64
 12  radius_se               569 non-null    float64
 13  texture_se              569 non-null    float64
 14  perimeter_se            569 non-null    float64
 15  area_se                 569 non-null    float64
 16  smoothness_se           569 non-null    float64
 17  compactness_se          569 non-null    float64
 18  concavity_se            569 non-null    float64
 19  concave points_se       569 non-null    float64
 20  symmetry_se             569 non-null    float64
 21  fractal_dimension_se    569 non-null    float64
 22  radius_worst            569 non-null    float64
 23  texture_worst           569 non-null    float64
 24  perimeter_worst         569 non-null    float64
 25  area_worst              569 non-null    float64
 26  smoothness_worst        569 non-null    float64
 27  compactness_worst       569 non-null    float64
 28  concavity_worst         569 non-null    float64
 29  concave points_worst    569 non-null    float64
 30  symmetry_worst          569 non-null    float64
 31  fractal_dimension_worst 569 non-null    float64
```






```
Accuracy of the classifier model is 0.9239766081871345
```

Week – 08

52. Logistic regression [using Fish prediction dataset]

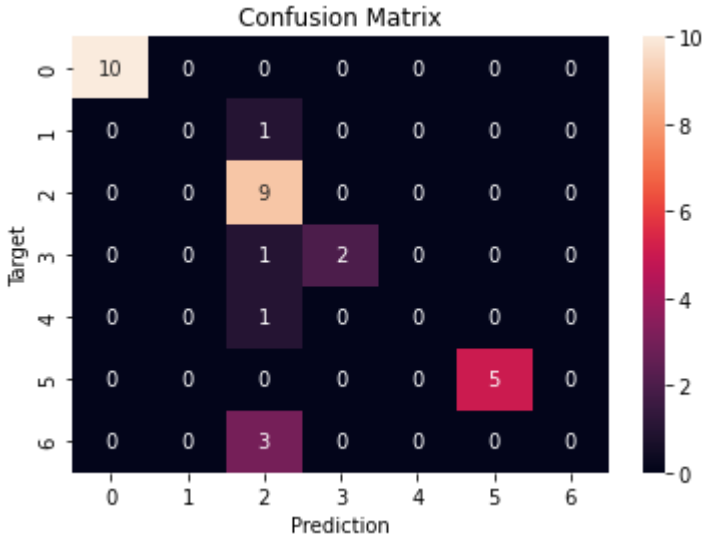
Code:

```
import pandas as p
import seaborn as s
import matplotlib.pyplot as m

da=p.read_csv("C:\\Users\\Desktop\\Data\\fish_prediction.csv")
print(da.isna().sum())

x=da.iloc[:,1:]
y=da.loc[:,'Species']

#Scaling
from sklearn.preprocessing import MinMaxScaler
sca=MinMaxScaler()
sca.fit(x)
x_sca=sca.transform(x)

#Transformation of Y
from sklearn.preprocessing import LabelEncoder
lb=LabelEncoder()
y=lb.fit_transform(y)

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_sca,y,test_size=0.2,rand
om_state=42)

from sklearn.linear_model import LogisticRegression
mod=LogisticRegression()
mod.fit(x_train,y_train)
y_pred=mod.predict(x_test)

from sklearn.metrics import accuracy_score
print("\nAccuracy:",accuracy_score(y_test,y_pred))

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
s.heatmap(cm,annot=True)
m.xlabel("Prediction"); m.ylabel('Target')
m.title('Confusion Matrix'); m.show()
```

Output:

```
Species    0
Weight     0
length1    0
length2    0
length3    0
Height     0
Width      0
dtype: int64
```

```
Accuracy: 0.8125
```

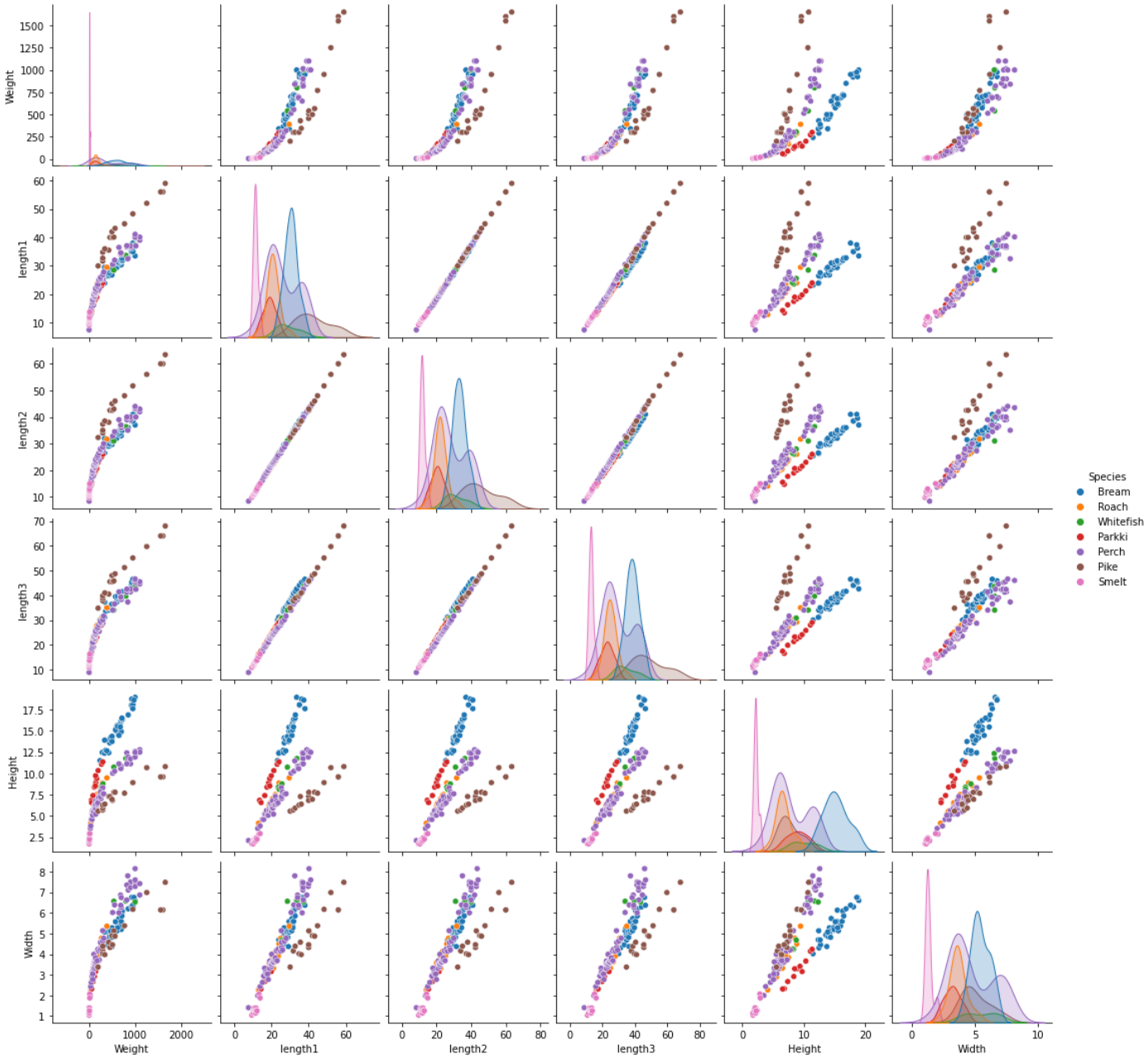


53. SVM Model [Using Fish prediction dataset]

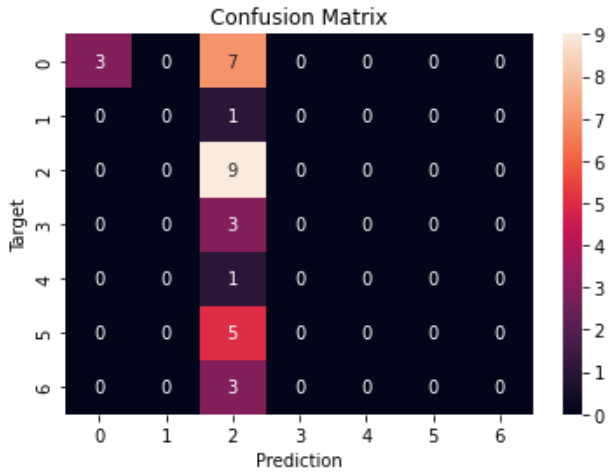
Code:

```
import pandas as p
import seaborn as s
import matplotlib.pyplot as m

da=p.read_csv("C:\\Users\\Desktop\\Data\\fish_prediction.csv")
s.pairplot(data=da,hue='Species')
m.show()

x=da.iloc[:,1:]
y=da.loc[:,'Species']

#Transformation of Y
from sklearn.preprocessing import LabelEncoder
lb=LabelEncoder()
y=lb.fit_transform(y)

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_sca,y,test_size=0.2,rand
om_state=42)

from sklearn.svm import SVC
mod=SVC(kernel='rbf',random_state=1,gamma='auto')
mod.fit(x_train,y_train)
y_pred=mod.predict(x_test)

from sklearn.metrics import accuracy_score
print("\nAccuracy:",accuracy_score(y_test,y_pred)*100)

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
s.heatmap(cm,annot=True)
m.xlabel("Prediction"); m.ylabel('Target')
m.title('Confusion Matrix'); m.show()
```

Output:




```
Accuracy: 37.5
```



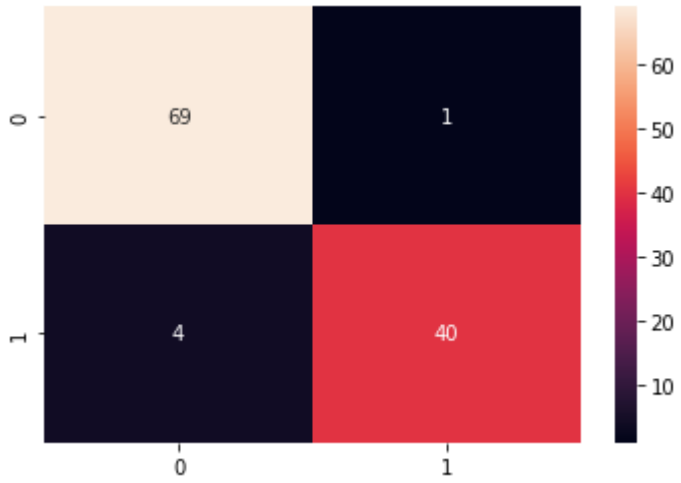
54. **Random Forest Classifier model [Using Breast Cancer dataset]**

Code:

```
import pandas as p
import seaborn as s
import matplotlib.pyplot as m
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

da=p.read_csv("C:\\Users\\Desktop\\Data\\brca1.csv")
da.drop(['id'],axis=1,inplace=True)
da.diagnosis=[1 if i=='M' else 0 for i in da.diagnosis]

x=da.drop(['diagnosis'],axis=1)
y=da.diagnosis.values

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_s
tate=30)
model1=RandomForestClassifier()
model1.fit(x_train,y_train)
y_pred1=model1.predict(x_test)

print("\nAccuracy of the model using Random Forest Regression alogorithm
is",accuracy_score(y_test,y_pred1))

cm=confusion_matrix(y_test,y_pred1)
s.heatmap(cm,annot=True)
m.show()
```

Output:

```
Accuracy of the model using Random Forest Regression alogorithm is 0.956140350877193
```



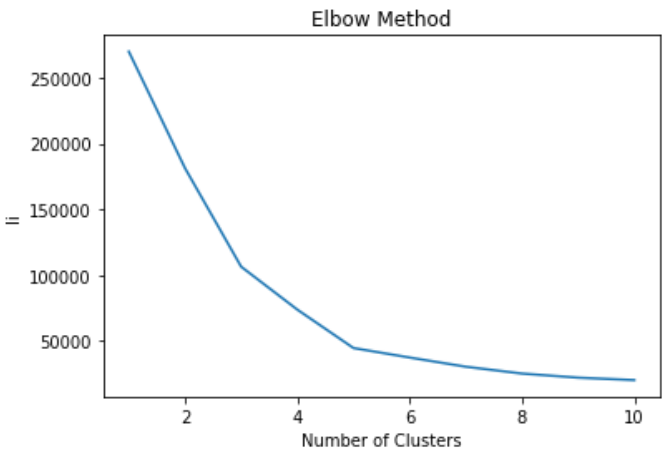
Week - 09

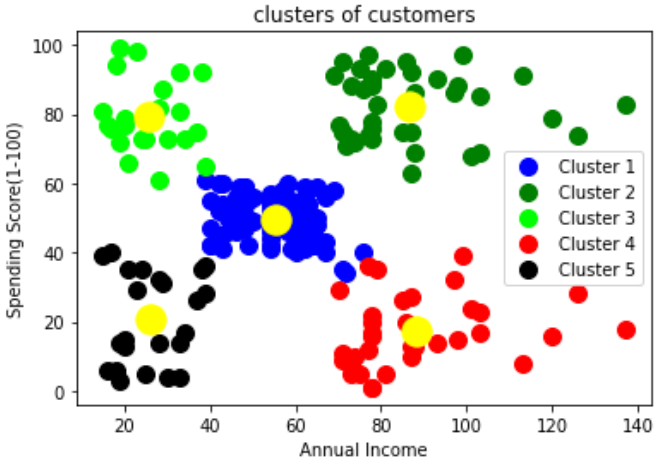
55. K-means [using Mall Customers Data]

Code:

```
import pandas as p
import matplotlib.pyplot as m
da=p.read_csv("C:\\Users\\Desktop\\Data\\Mall_Customers.csv")
x=da.iloc[:,[3,4]].values
from sklearn.cluster import KMeans
li=[]

for i in range(1,11):
    mod=KMeans(n_clusters=i,init='k-means++', random_state=42,n_init=10)
    mod.fit(x)
    li.append(mod.inertia_)

m.plot(range(1,11), li)
m.title("Elbow Method")
m.xlabel('Number of Clusters'); m.ylabel('li'); m.show()

mod=KMeans(n_clusters=5,init='k-means++', random_state=42,n_init=10)
y_pred=mod.fit_predict(x)

m.scatter(x[y_pred==0,0],x[y_pred==0,1],s=100, c='blue', label='Cluster 1')
m.scatter(x[y_pred==1,0],x[y_pred==1,1],s=100,c='g', label='Cluster 2')
m.scatter(x[y_pred==2,0],x[y_pred==2,1],s=100, c='lime', label='Cluster 3')
m.scatter(x[y_pred==3,0],x[y_pred==3,1],s=100, c='red', label='Cluster 4')
m.scatter(x[y_pred==4,0],x[y_pred==4,1],s=100, c='k', label='Cluster 5')
m.scatter
(mod.cluster_centers_[:,0],mod.cluster_centers_[:,1],s=300,c='yellow')
m.title("clusters of customers")
m.xlabel("Annual Income"); m.ylabel("Spending Score(1-100)")
m.legend();m.show()
```

Output:





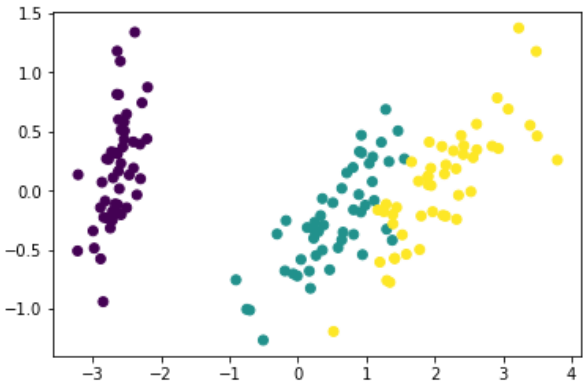
56. **Dimensionality Reduction [PCA] using iris Dataset**

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA

iris=datasets.load_iris()
x=iris.data
y=iris.target
print(x.shape)
print(y.shape)

pca=PCA(n_components=2)
pca.fit(x)
print(pca.components_)

x=pca.transform(x)
print(x.shape)
plt.scatter (x[:,0],x[:,1],c=y)

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

x_train, x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
res=DecisionTreeClassifier()
res.fit(x_train,y_train)

y_predict=res.predict(x_test)
print(accuracy_score (y_test,y_predict))
```

Output:

```
(150, 4)
(150,)
[[ 0.36138659 -0.08452251  0.85667061  0.3582892 ]
 [ 0.65658877  0.73016143 -0.17337266 -0.07548102]]
(150, 2)
0.9666666666666667
```



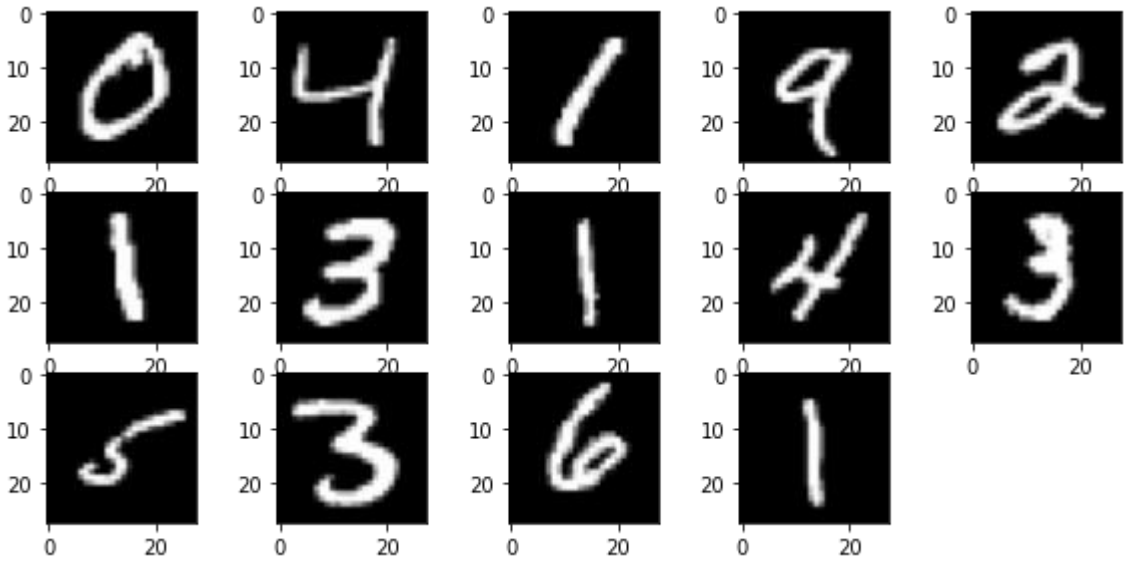
Week - 10

57. **Shallow Neural Network**

Code:

```
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.utils import to_categorical
layers = tf.keras.layers

from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# one-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

plt.figure(figsize=(10,8))
for i in range(1, 15):
    # image in ith position of grid
    plt.subplot(5, 5, i)
    plt.imshow(x_train[i], cmap=plt.get_cmap('gray'))
plt.show()

model = tf.keras.Sequential()
model.add(layers.Flatten())
model.add(layers.Dense(5, activation='sigmoid'))
model.add(layers.Dense(10, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=32,
validation_data=(x_test, y_test))

loss, accuracy = model.evaluate(x_test, y_test)
print('\n\nTest loss', loss)
print('Test accuracy', accuracy)
```

Output:



```
Epoch 1/10
1875/1875 [==============================] - 8s 4ms/step - loss: 1.4526 - accuracy: 0.5931 - val_loss: 1.0243 - val_accuracy: 0.7557
Epoch 2/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.8635 - accuracy: 0.7800 - val_loss: 0.7250 - val_accuracy: 0.8135
Epoch 3/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.6821 - accuracy: 0.8089 - val_loss: 0.6251 - val_accuracy: 0.8249
Epoch 4/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.6089 - accuracy: 0.8218 - val_loss: 0.5780 - val_accuracy: 0.8343
Epoch 5/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.5670 - accuracy: 0.8329 - val_loss: 0.5476 - val_accuracy: 0.8441
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.5363 - accuracy: 0.8429 - val_loss: 0.5239 - val_accuracy: 0.8494
Epoch 7/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.5115 - accuracy: 0.8510 - val_loss: 0.5060 - val_accuracy: 0.8531
Epoch 8/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.4898 - accuracy: 0.8579 - val_loss: 0.4863 - val_accuracy: 0.8610
Epoch 9/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.4724 - accuracy: 0.8639 - val_loss: 0.4795 - val_accuracy: 0.8615
Epoch 10/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.4587 - accuracy: 0.8658 - val_loss: 0.4662 - val_accuracy: 0.8657
313/313 [==============================] - 1s 3ms/step - loss: 0.4662 - accuracy: 0.8657
```

```
Test loss 0.4661898910999298
Test accuracy 0.8657000064849854
```

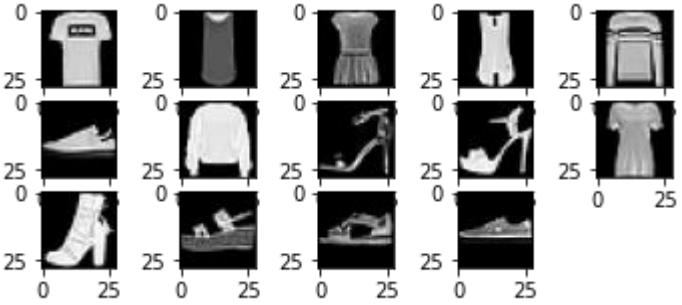
58. **Deep Neural Network**

Code:

```
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.utils import to_categorical
from keras.datasets import fashion_mnist

layers = tf.keras.layers
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)


print('X_train: ' + str(x_train.shape))
print('Y_train: ' + str(y_train.shape))
print('X_test:  '  + str(x_test.shape))
print('Y_test:  '  + str(x_test.shape))

for i in range(1, 15):
    plt.subplot(5, 5, i)
    plt.imshow(x_train[i], cmap=plt.get_cmap('gray'))
plt.show()

model = tf.keras.Sequential()
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['
accuracy'])
model.fit(x_train, y_train, epochs=10)

loss, accuracy = model.evaluate(x_test, y_test)
print('\n\nTest loss', loss)
print('Test accuracy', accuracy)
```

Output:

```
X_train: (60000, 28, 28)
Y_train: (60000, 10)
X_test:  (10000, 28, 28)
Y_test:  (10000, 28, 28)
```



```
Epoch 1/10
1875/1875 [==============================] - 12s 5ms/step - loss: 1.0180 - accuracy: 0.7400
Epoch 2/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.5336 - accuracy: 0.8119
Epoch 3/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.4713 - accuracy: 0.8318
Epoch 4/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.4366 - accuracy: 0.8428
Epoch 5/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.4120 - accuracy: 0.8492
Epoch 6/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.3943 - accuracy: 0.8556
Epoch 7/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.3826 - accuracy: 0.8603
Epoch 8/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.3711 - accuracy: 0.8639
Epoch 9/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.3617 - accuracy: 0.8676
Epoch 10/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.3538 - accuracy: 0.8705
313/313 [==============================] - 2s 4ms/step - loss: 0.4548 - accuracy: 0.8404
```

```
Test loss 0.454790323972702
Test accuracy 0.840399980545044
```

Week-11

59. **Tokenization [Sentence & Word]**

Code:

```
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize

f=open("C:\\Users\\Desktop\\Data\\da.txt")
text=f.read()
print(text)

sent=sent_tokenize(text)
print("Number of sentences:",len(sent))
for i in range(len(sent)):
    print("\nSentence:",i+1,"\n",sent[i])

w=word_tokenize(text)
print("\nTotal Words:",len(w))
print(w)
```

Output:

```
Smartphones have become an essential part of our daily lives, offering communication, entertainment, and productivity all in one device. The rapid advancements
in smartphone technology have led to features like facial recognition, high-resolution cameras, and 5G connectivity.

Number of sentences: 2

Sentence: 1
 Smartphones have become an essential part of our daily lives, offering communication, entertainment, and productivity all in one device.

Sentence: 2
 The rapid advancements in smartphone technology have led to features like facial recognition, high-resolution cameras, and 5G connectivity.

Total Words: 44
['Smartphones', 'have', 'become', 'an', 'essential', 'part', 'of', 'our', 'daily', 'lives', ',', 'offering', 'communication', ',', 'entertainment', ',', 'and',
'productivity', 'all', 'in', 'one', 'device', '.', 'The', 'rapid', 'advancements', 'in', 'smartphone', 'technology', 'have', 'led', 'to', 'features', 'like',
'facial', 'recognition', ',', 'high-resolution', 'cameras', ',', 'and', '5G', 'connectivity', '.']
```

60. **N-Grams**

Code:

```
from nltk.util import ngrams
from nltk.tokenize import word_tokenize

f=open("C:\\Users\\Desktop\\da.txt")
text=f.read()
w=word_tokenize(text)

print("Bi-Grams:\n\n",list(ngrams(w,2)))
print("\n\nTri-Grams:\n\n",list(ngrams(w,3)))
```

Output:

```
Bi-Grams:

 [('Smartphones', 'have'), ('have', 'become'), ('become', 'an'), ('an', 'essential'), ('essential', 'part'), ('part', 'of'), ('of', 'our'), ('our', 'daily'),
('daily', 'lives'), ('lives', ','), (',', 'offering'), ('offering', 'communication'), ('communication', ','), (',', 'entertainment'), ('entertainment', ','),
(',', 'and'), ('and', 'productivity'), ('productivity', 'all'), ('all', 'in'), ('in', 'one'), ('one', 'device'), ('device', '.'), ('.', 'The'), ('The', 'rapid'),
('rapid', 'advancements'), ('advancements', 'in'), ('in', 'smartphone'), ('smartphone', 'technology'), ('technology', 'have'), ('have', 'led'), ('led', 'to'),
('to', 'features'), ('features', 'like'), ('like', 'facial'), ('facial', 'recognition'), ('recognition', ','), (',', 'high-resolution'), ('high-resolution',
'cameras'), ('cameras', ','), (',', 'and'), ('and', '5G'), ('5G', 'connectivity'), ('connectivity', '.')]


Tri-Grams:

 [('Smartphones', 'have', 'become'), ('have', 'become', 'an'), ('become', 'an', 'essential'), ('an', 'essential', 'part'), ('essential', 'part', 'of'), ('part',
'of', 'our'), ('of', 'our', 'daily'), ('our', 'daily', 'lives'), ('daily', 'lives', ','), ('lives', ',', 'offering'), (',', 'offering', 'communication'),
('offering', 'communication', ','), ('communication', ',', 'entertainment'), (',', 'entertainment', ','), ('entertainment', ',', 'and'), (',', 'and',
'productivity'), ('and', 'productivity', 'all'), ('productivity', 'all', 'in'), ('all', 'in', 'one'), ('in', 'one', 'device'), ('one', 'device', '.'), ('device',
'.', 'The'), ('.', 'The', 'rapid'), ('The', 'rapid', 'advancements'), ('rapid', 'advancements', 'in'), ('advancements', 'in', 'smartphone'), ('in', 'smartphone',
'technology'), ('smartphone', 'technology', 'have'), ('technology', 'have', 'led'), ('have', 'led', 'to'), ('led', 'to', 'features'), ('to', 'features', 'like'),
('features', 'like', 'facial'), ('like', 'facial', 'recognition'), ('facial', 'recognition', ','), ('recognition', ',', 'high-resolution'), (',', 'high-
resolution', 'cameras'), ('high-resolution', 'cameras', ','), ('cameras', ',', 'and'), (',', 'and', '5G'), ('and', '5G', 'connectivity'), ('5G', 'connectivity',
'.')]
```

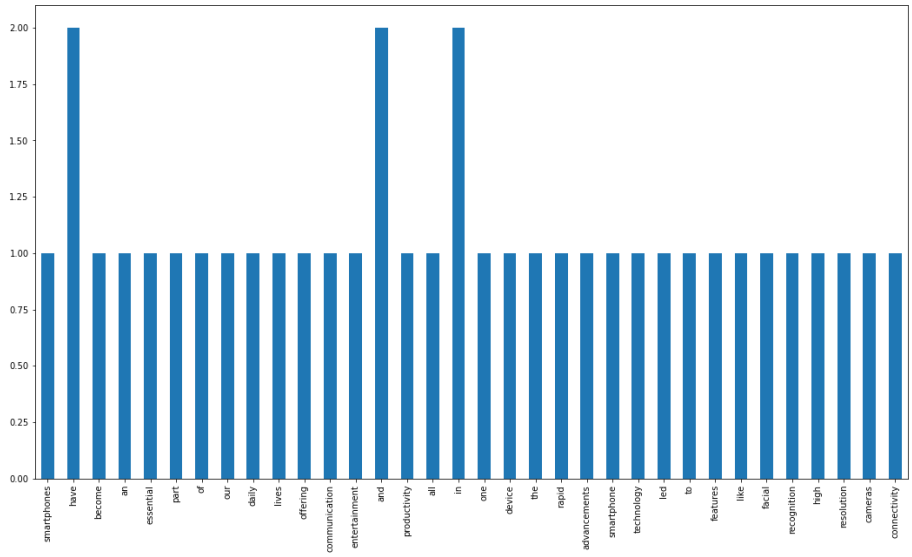
61. Frequency Distribution of Words

Code:

```
import pandas as p
import matplotlib.pyplot as m
from nltk.probability import FreqDist

freq=FreqDist(w)
print("Count of and word:",freq['and'])

freq=p.Series(dict(freq))
m.figure(figsize=(18,10))
freq.plot(kind='bar')
m.show()
```

Output:

```
Count of and word: 2
```



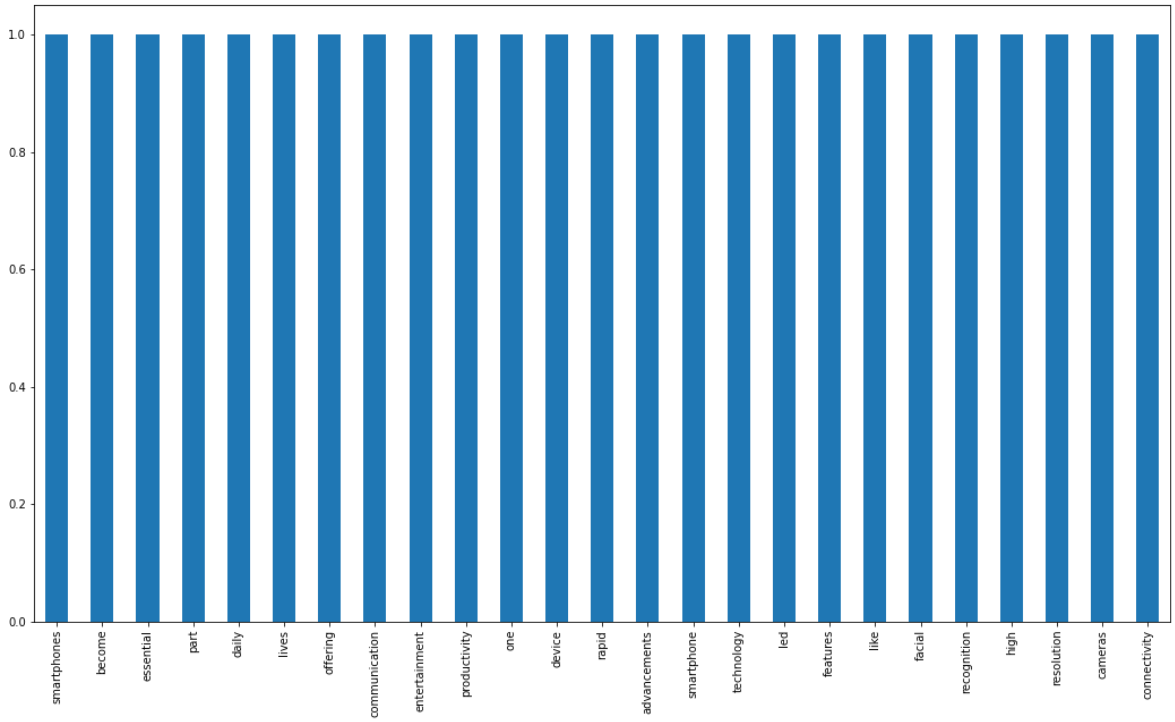
62. **Removing Stop-words.**

Code:

```
import nltk
import matplotlib.pyplot as m
import pandas as p
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist

w = word_tokenize(text)
print("Before Removal of Stopwords words count:",len(w))
stop_w = nltk.corpus.stopwords.words('english')
removed_stopw = []

for i in w:
    if i not in stop_w:
        removed_stopw.append(i)

print("\nAfter Removal of StopWords:",len(removed_stopw))
freq = FreqDist(removed_stopw)
freq_series = p.Series(dict(freq))

m.figure(figsize=(18, 10))
freq_series.plot(kind='bar')
m.show()
```

Output:

```
Before Removal of Stopwords words count: 37

After Removal of StopWords: 25
```



63. **Word-cloud**

Code:

```
import matplotlib.pyplot as m
f=open("C:\\Users\\Desktop\\da.txt")
text=f.read()

from wordcloud import WordCloud, STOPWORDS
stop_w=set(STOPWORDS)

wc=WordCloud(width=800,height=800,
             background_color='black',colormap='plasma',
             stopwords=stop_w,
             min_font_size=10).generate(text)

m.figure(figsize=(8,8),facecolor=None)
m.imshow(wc)
m.axis('off')
m.tight_layout(pad=0)
m.show()
```

Output:

64. **Stemming & Lemmatization**

Code:

```
import re
from nltk.tokenize import word_tokenize

f=open("C:\\Users\\Desktop\\da.txt")
text=f.read()
text=text.lower()

text=re.sub('[^A-Za-z0-9]+',' ',text)
text=re.sub("\S*\d\S*","",text).strip()
print(text)

w=word_tokenize(text,preserve_line=True)

from nltk.stem import PorterStemmer
ps=PorterStemmer()
ps_st=[ps.stem(i) for i in w]
print("\nStemming:\n\n",ps_st)

from nltk import WordNetLemmatizer
wnl=WordNetLemmatizer()
lema=[wnl.lemmatize(u) for u in w]

print("\n Lemmatization:\n\n",lema)
```

Output:

```
smartphones have become an essential part of our daily lives offering communication entertainment and productivity all in one device the rapid advancements in smartphone technology have led to features like facial recognition high resolution cameras and  connectivity

Stemming:

 ['smartphon', 'have', 'becom', 'an', 'essenti', 'part', 'of', 'our', 'daili', 'live', 'offer', 'commun', 'entertain', 'and', 'product', 'all', 'in', 'one', 'devic', 'the', 'rapid', 'advanc', 'in', 'smartphon', 'technolog', 'have', 'led', 'to', 'featur', 'like', 'facial', 'recognit', 'high', 'resolut', 'camera', 'and', 'connect']

 Lemmatization:

 ['smartphones', 'have', 'become', 'an', 'essential', 'part', 'of', 'our', 'daily', 'life', 'offering', 'communication', 'entertainment', 'and', 'productivity', 'all', 'in', 'one', 'device', 'the', 'rapid', 'advancement', 'in', 'smartphone', 'technology', 'have', 'led', 'to', 'feature', 'like', 'facial', 'recognition', 'high', 'resolution', 'camera', 'and', 'connectivity']
```