230701036

ARUN MC

II CSE A

In [1]:
```
import pandas
```

In [2]:
```
x={'cars':['BMW','volvo','toyota'],'Manufacture_count':[1500,1254,2143],'Sold_co
b=pandas.DataFrame(x)
```

In [3]:
```
print(b)
```

```
    cars  Manufacture_count  Sold_count
0    BMW               1500         752
1   volvo               1254         954
2  toyota               2143        1103
```

In [4]:
```
b.loc[0]
```

Out[4]:
```
cars                BMW
Manufacture_count  1500
Sold_count          752
Name: 0, dtype: object
```

In [5]:
```
#2nd dataframe
```

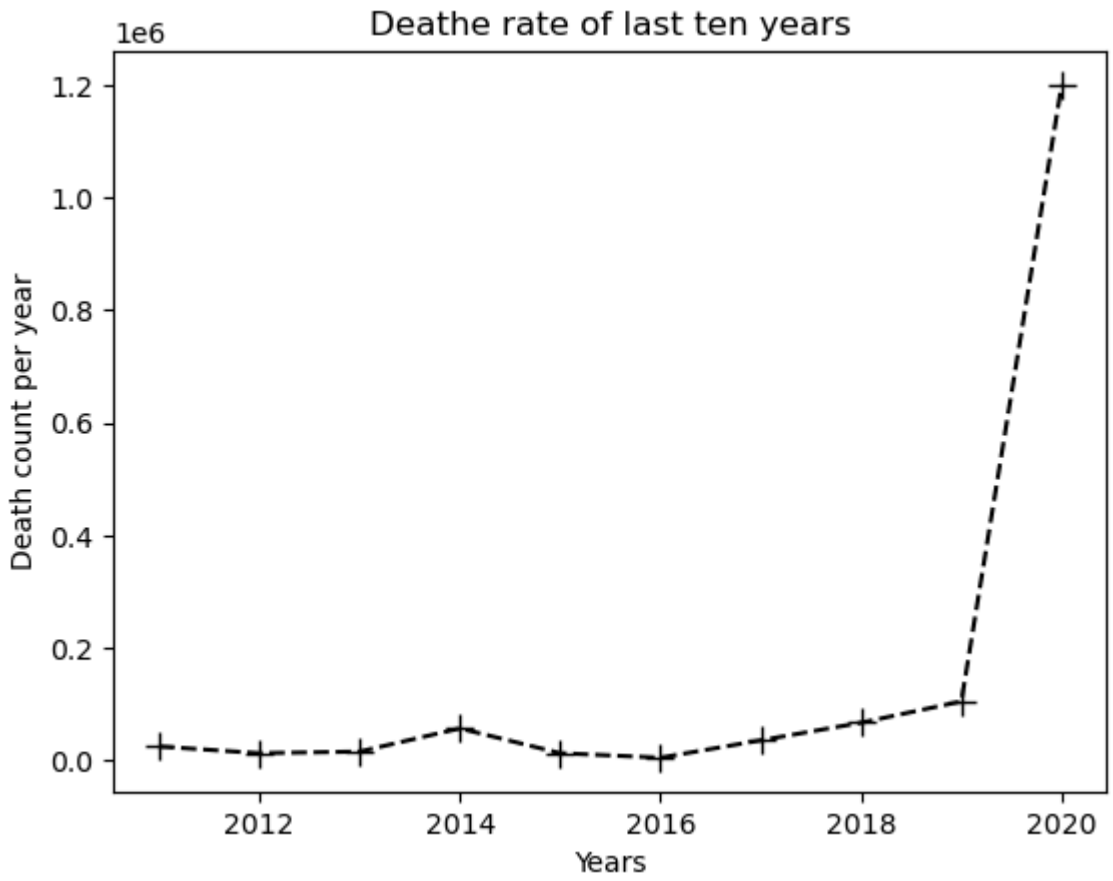
In [6]:
```
data={'Year':list(range(2011,2021)),'Death_rate':[24513,12543,15423,56612,12564,
c=pandas.DataFrame(data)
```

In [7]:
```
print(c)
```

```
   Year  Death_rate
0  2011       24513
1  2012       12543
2  2013       15423
3  2014       56612
4  2015       12564
5  2016        4571
6  2017       35412
7  2018       66742
8  2019      104523
9  2020     1200451
```

In [8]:
```
import matplotlib.pyplot as plt
```

In [9]:
```
plt.plot(c['Year'],c['Death_rate'],marker='+',ms=10,linestyle='--',color='black'
plt.title("Deathe rate of last ten years")
plt.xlabel('Years')
plt.ylabel('Death count per year')
plt.show()
```



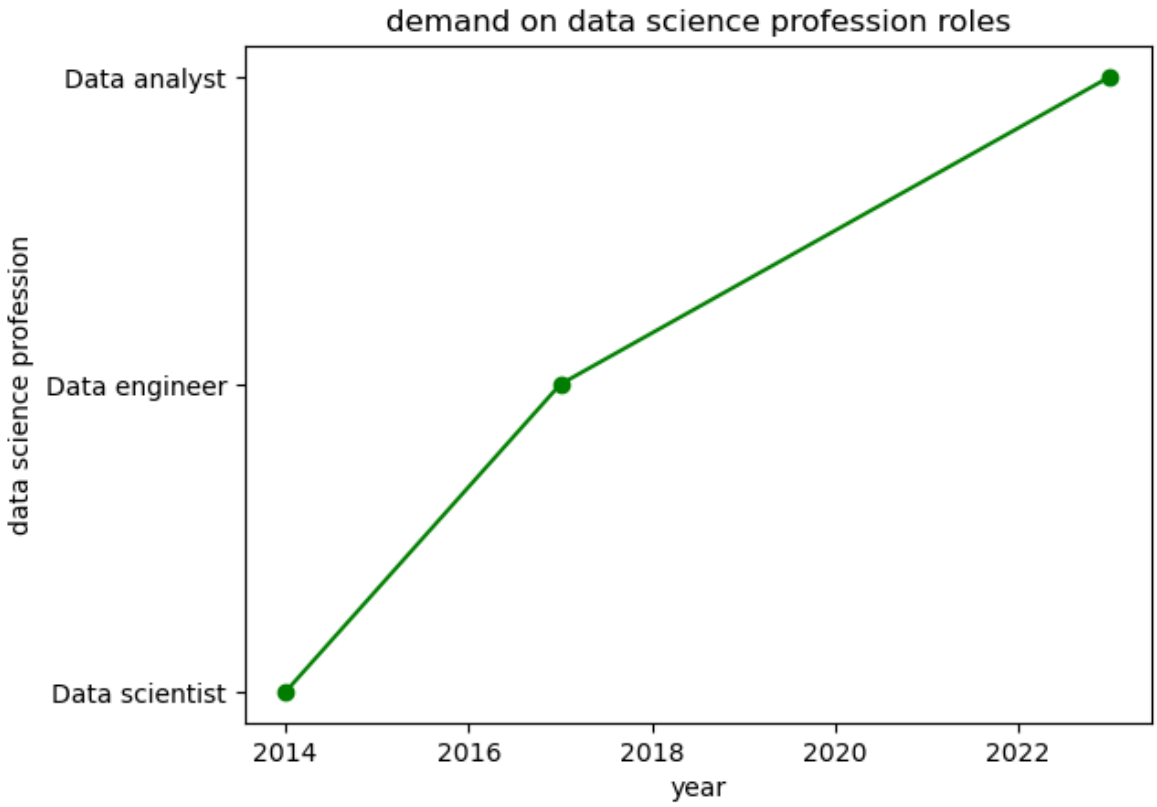
In [10]:
```
#3.analyze and visualizationof various data science roles
```

In [11]:
```
roles=['Data scientist','Data engineer','Data analyst',]
demand_rate=[70,80,94]
year=[2014,2017,2023]
data1={'roles':roles,'demand_rate':demand_rate,'year':year}
```

In [12]:
```
print(pandas.DataFrame(data1))
```

```
           roles  demand_rate  year
0  Data scientist           70  2014
1   Data engineer           80  2017
2    Data analyst           94  2023
```

In [13]:
```
plt.title("demand on data science profession roles")
plt.plot(data1['year'],data1['roles'],marker='o',color='green')
plt.xlabel('year')
plt.ylabel('data science profession')
plt.show()
```



In [14]:
```
#unstructured dataframes
unstruct1="Hi I am MC Arun.\nThis is the example for unstructured data."
print(unstruct1)
```

```
Hi I am MC Arun.
This is the example for unstructured data.
```

In [15]:
```
#structured data frames
struct1=pandas.DataFrame({
    'sl_no':['1','2','3','4','5','6','7','8','9','10'],'Name':['Arun','Aravinda'
})
struct1
```

Out[15]:

	sl_no	Name	Reg_no
0	1	Arun	036
1	2	Aravinda	032
2	3	Arul	035
3	4	Anu	030
4	5	Akshay	021
5	6	Ashish	040
6	7	Sam	026
7	8	Ajay	017
8	9	Aravindan	030
9	10	darshan	061

In [16]:
```
#read csv file
```

In [17]:
```
import pandas as pd
ds=pd.read_csv('dataset1.csv')
ds
```

Out[17]:

	sl_no	Register_no	Name	sem-1 Grade	Department
0	1	11	Arun	A+	CSE
1	2	12	Sam	O	CSE
2	3	13	Arul	A+	CSE
3	4	14	Akshay	A	CSE
4	5	15	Ajay	B+	CSE
5	6	16	Anu	A+	CSE
6	7	17	Aravinda	A+	CSE
7	8	18	Ajeesh	O	CSE
8	9	19	Reshmiajeesh	O	CSE
9	10	20	Ashish	O	CSE
10	11	21	Arivind	B+	CSE
11	12	22	Kumar	A+	CSE
12	13	23	Darshan	A	CSE
13	14	24	Ashna	A	CSE
14	15	25	Govind	O	CSE
15	16	26	Rakesh	A+	CSE
16	17	27	Terila	O	CSE
17	18	28	NaN	B+	CSE
18	19	29	Arayan	B+	CSE
19	20	30	Arya	O	CSE

In [18]:
```
#display first 10 rows
ds.head(10)
```

Out[18]:

	sl_no	Register_no	Name	sem-1 Grade	Department
0	1	11	Arun	A+	CSE
1	2	12	Sam	O	CSE
2	3	13	Arul	A+	CSE
3	4	14	Akshay	A	CSE
4	5	15	Ajay	B+	CSE
5	6	16	Anu	A+	CSE
6	7	17	Aravinda	A+	CSE
7	8	18	Ajeesh	O	CSE
8	9	19	Reshmiajeesh	O	CSE
9	10	20	Ashish	O	CSE

In [19]:
```
#displaying last 5 rows
ds.tail()
```

Out[19]:

	sl_no	Register_no	Name	sem-1 Grade	Department
15	16	26	Rakesh	A+	CSE
16	17	27	Terila	O	CSE
17	18	28	NaN	B+	CSE
18	19	29	Arayan	B+	CSE
19	20	30	Arya	O	CSE

In [20]:
```
#filling the null place
ds.fillna('Raja',inplace=True)
ds
```

Out[20]:

	sl_no	Register_no	Name	sem-1 Grade	Department
0	1	11	Arun	A+	CSE
1	2	12	Sam	O	CSE
2	3	13	Arul	A+	CSE
3	4	14	Akshay	A	CSE
4	5	15	Ajay	B+	CSE
5	6	16	Anu	A+	CSE
6	7	17	Aravinda	A+	CSE
7	8	18	Ajeesh	O	CSE
8	9	19	Reshmiajeesh	O	CSE
9	10	20	Ashish	O	CSE
10	11	21	Arivind	B+	CSE
11	12	22	Kumar	A+	CSE
12	13	23	Darshan	A	CSE
13	14	24	Ashna	A	CSE
14	15	25	Govind	O	CSE
15	16	26	Rakesh	A+	CSE
16	17	27	Terila	O	CSE
17	18	28	Raja	B+	CSE
18	19	29	Arayan	B+	CSE
19	20	30	Arya	O	CSE

In [21]:
```
ds['sl_no']='Hello'
ds[0]='nothing'
ds
```

Out[21]:

	sl_no	Register_no	Name	sem-1 Grade	Department	0
0	Hello	11	Arun	A+	CSE	nothing
1	Hello	12	Sam	O	CSE	nothing
2	Hello	13	Arul	A+	CSE	nothing
3	Hello	14	Akshay	A	CSE	nothing
4	Hello	15	Ajay	B+	CSE	nothing
5	Hello	16	Anu	A+	CSE	nothing
6	Hello	17	Aravinda	A+	CSE	nothing
7	Hello	18	Ajeesh	O	CSE	nothing
8	Hello	19	Reshmiajeesh	O	CSE	nothing
9	Hello	20	Ashish	O	CSE	nothing
10	Hello	21	Arivind	B+	CSE	nothing
11	Hello	22	Kumar	A+	CSE	nothing
12	Hello	23	Darshan	A	CSE	nothing
13	Hello	24	Ashna	A	CSE	nothing
14	Hello	25	Govind	O	CSE	nothing
15	Hello	26	Rakesh	A+	CSE	nothing
16	Hello	27	Terila	O	CSE	nothing
17	Hello	28	Raja	B+	CSE	nothing
18	Hello	29	Arayan	B+	CSE	nothing
19	Hello	30	Arya	O	CSE	nothing

In [22]:
```
del ds
```

In [23]:
```
ds=pd.read_csv('dataset1.csv')
ds
```

Out[23]:

	sl_no	Register_no	Name	sem-1 Grade	Department
0	1	11	Arun	A+	CSE
1	2	12	Sam	O	CSE
2	3	13	Arul	A+	CSE
3	4	14	Akshay	A	CSE
4	5	15	Ajay	B+	CSE
5	6	16	Anu	A+	CSE
6	7	17	Aravinda	A+	CSE
7	8	18	Ajeesh	O	CSE
8	9	19	Reshmiajeesh	O	CSE
9	10	20	Ashish	O	CSE
10	11	21	Arivind	B+	CSE
11	12	22	Kumar	A+	CSE
12	13	23	Darshan	A	CSE
13	14	24	Ashna	A	CSE
14	15	25	Govind	O	CSE
15	16	26	Rakesh	A+	CSE
16	17	27	Terila	O	CSE
17	18	28	NaN	B+	CSE
18	19	29	Arayan	B+	CSE
19	20	30	Arya	O	CSE

In [24]:
```
x=ds['sl_no'].mean()
x
```

Out[24]: 10.5

In [25]:
```
y=ds['sl_no'].median()
y
```

Out[25]: 10.5

In [26]:
```
#summary statistics
ds.describe()
```

Out[26]:

	sl_no	Register_no
count	20.00000	20.00000
mean	10.50000	20.50000
std	5.91608	5.91608
min	1.00000	11.00000
25%	5.75000	15.75000
50%	10.50000	20.50000
75%	15.25000	25.25000
max	20.00000	30.00000

In [27]:
```
#identifying the null valuse and displays its count
print(ds.isnull().sum())
```

```
sl_no          0
Register_no    0
Name           1
sem-1 Grade    0
Department     0
dtype: int64
```

In [28]:
```
f=ds['sem-1 Grade'].duplicated()
f
```

Out[28]:
```
0     False
1     False
2      True
3     False
4     False
5      True
6      True
7      True
8      True
9      True
10     True
11     True
12     True
13     True
14     True
15     True
16     True
17     True
18     True
19     True
Name: sem-1 Grade, dtype: bool
```

In [29]:
```
#slicing
ds[0:20:2]
```

Out[29]:

	sl_no	Register_no	Name	sem-1 Grade	Department
0	1	11	Arun	A+	CSE
2	3	13	Arul	A+	CSE
4	5	15	Ajay	B+	CSE
6	7	17	Aravinda	A+	CSE
8	9	19	Reshmiajeesh	O	CSE
10	11	21	Arivind	B+	CSE
12	13	23	Darshan	A	CSE
14	15	25	Govind	O	CSE
16	17	27	Terila	O	CSE
18	19	29	Arayan	B+	CSE

In [30]:
```
#slicing particular field
ds[['Register_no','sem-1 Grade'][0:20]]
```

Out[30]:

	Register_no	sem-1 Grade
0	11	A+
1	12	O
2	13	A+
3	14	A
4	15	B+
5	16	A+
6	17	A+
7	18	O
8	19	O
9	20	O
10	21	B+
11	22	A+
12	23	A
13	24	A
14	25	O
15	26	A+
16	27	O
17	28	B+
18	29	B+
19	30	O

In [31]:
```
#new data set based on sales
import pandas as pd
import matplotlib.pyplot as plt
```

In [32]:
```
sales_data_set=pd.read_csv('sales_data.csv')
sales_data_set
```

Out[32]:

	Date	Product	Sales	Quantity	Region
0	01-01-2023	Product A	200	4	North
1	02-01-2023	Product B	150	3	South
2	03-01-2023	Product A	220	5	North
3	04-01-2023	Product C	300	6	East
4	05-01-2023	Product B	180	4	West
5	06-01-2023	Product A	210	5	North
6	07-01-2023	Product C	320	7	East
7	08-01-2023	Product B	160	3	South
8	09-01-2023	Product A	230	6	North
9	10-01-2023	Product C	310	7	East
10	11-01-2023	Product B	190	4	West
11	12-01-2023	Product A	240	6	North
12	13-01-2023	Product C	330	8	East
13	14-01-2023	Product B	170	3	South
14	15-01-2023	Product A	250	7	North
15	16-01-2023	Product C	340	8	East

In [33]:
```
sales_data_set.describe()
```

Out[33]:

	Sales	Quantity
count	16.000000	16.000000
mean	237.500000	5.375000
std	64.031242	1.746425
min	150.000000	3.000000
25%	187.500000	4.000000
50%	225.000000	5.500000
75%	302.500000	7.000000
max	340.000000	8.000000

In [34]:
```
sales_data_set.columns
```

Out[34]:
```
Index(['Date', 'Product', 'Sales', 'Quantity', 'Region'], dtype='object')
```

In [35]:
```
sales_data_set.isnull()
```

Out[35]:

	Date	Product	Sales	Quantity	Region
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
5	False	False	False	False	False
6	False	False	False	False	False
7	False	False	False	False	False
8	False	False	False	False	False
9	False	False	False	False	False
10	False	False	False	False	False
11	False	False	False	False	False
12	False	False	False	False	False
13	False	False	False	False	False
14	False	False	False	False	False
15	False	False	False	False	False

In [36]:
```
sales_data_set.dropna(inplace=True)
```

In [37]:
```
sales_data_set_order=sales_data_set.sort_values('Sales',ascending=False)
sales_data_set_order
```

Out[37]:

	Date	Product	Sales	Quantity	Region
15	16-01-2023	Product C	340	8	East
12	13-01-2023	Product C	330	8	East
6	07-01-2023	Product C	320	7	East
9	10-01-2023	Product C	310	7	East
3	04-01-2023	Product C	300	6	East
14	15-01-2023	Product A	250	7	North
11	12-01-2023	Product A	240	6	North
8	09-01-2023	Product A	230	6	North
2	03-01-2023	Product A	220	5	North
5	06-01-2023	Product A	210	5	North
0	01-01-2023	Product A	200	4	North
10	11-01-2023	Product B	190	4	West
4	05-01-2023	Product B	180	4	West
13	14-01-2023	Product B	170	3	South
7	08-01-2023	Product B	160	3	South
1	02-01-2023	Product B	150	3	South

In [38]:
```
sales_data_set_order.Product.unique()
```

Out[38]:
```
array(['Product C', 'Product A', 'Product B'], dtype=object)
```

In [39]:
```
print(sales_data_set['Sales'].mean())
```

```
237.5
```

In [40]:
```
# average of product A,B,C
average_of_A=sales_data_set_order['Sales'].head(5).mean()
print(average_of_A)
average_of_B=sales_data_set_order['Sales'].iloc[5:11].mean()
print(average_of_B)
average_of_C=sales_data_set_order['Sales'].tail().mean()
print(average_of_C)
```

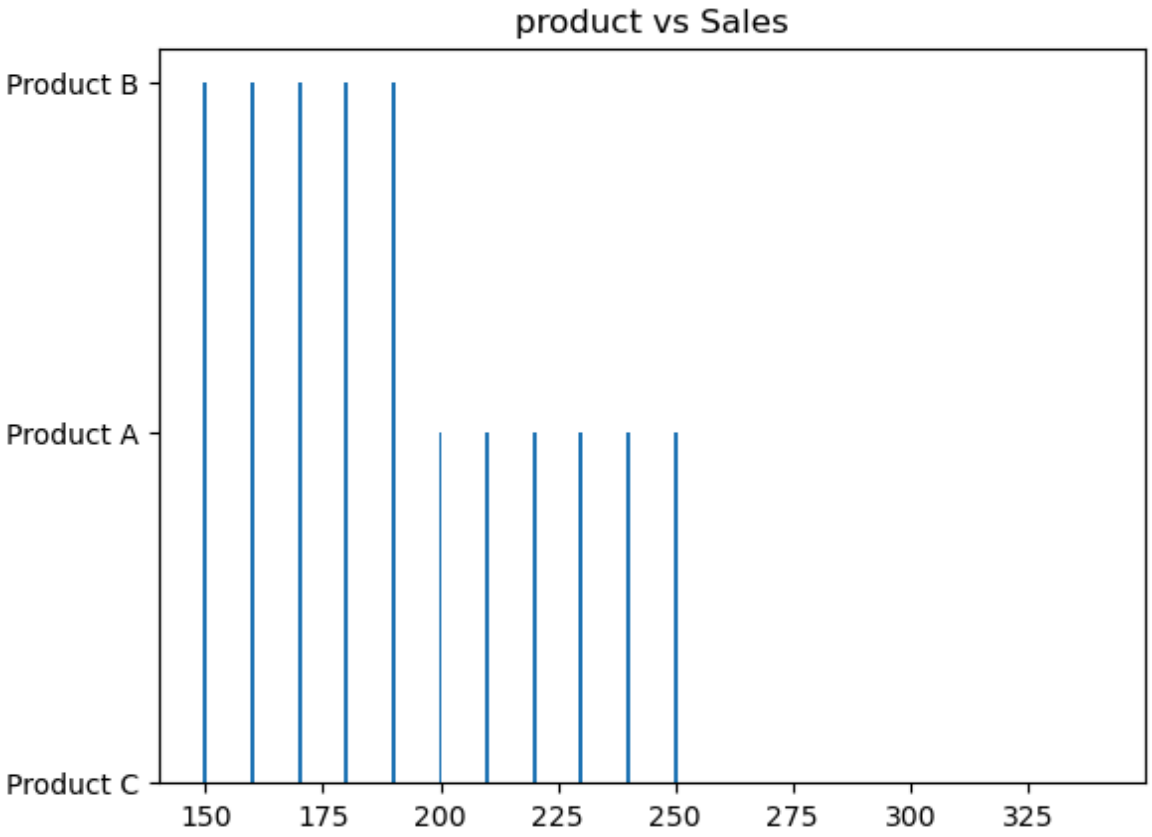
```
320.0
225.0
170.0
```

In [41]:
```
# data frame of average of each product
avg_of_product=pd.DataFrame({'Product_name':['Porduct A','Product B','Product C'
avg_of_product
```

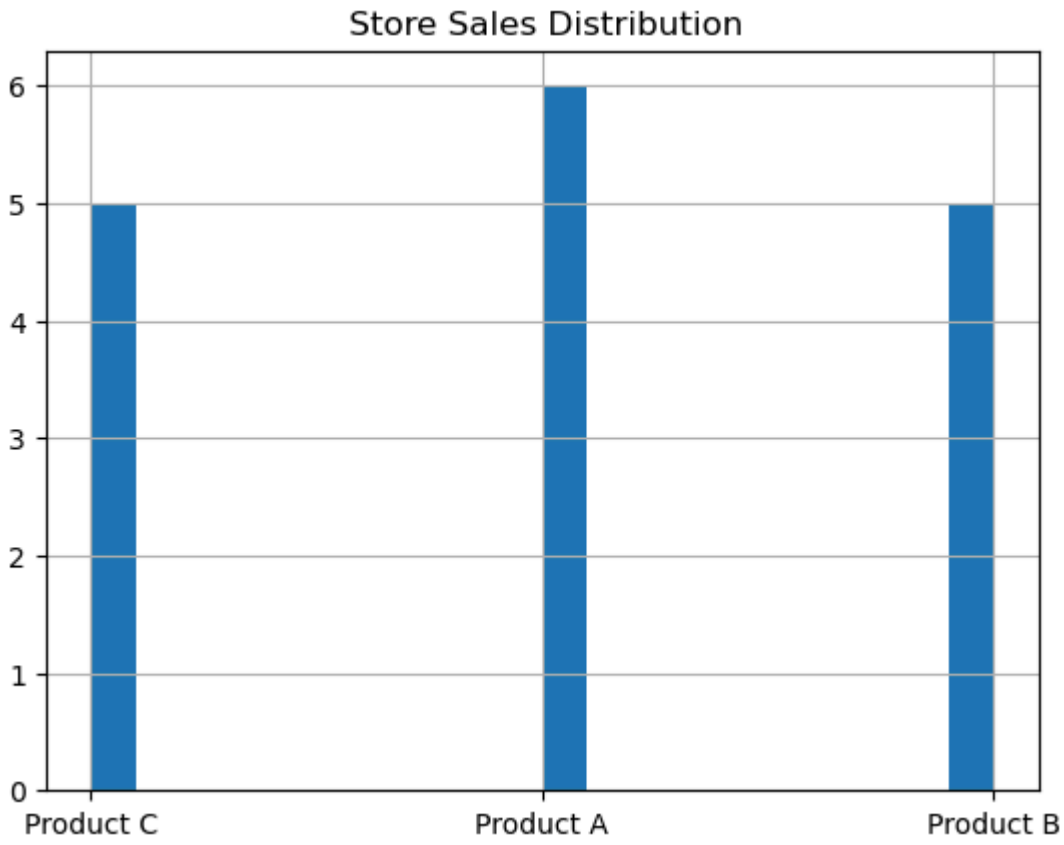
Out[41]:

	Product_name	Average
0	Porduct A	320.0
1	Product B	225.0
2	Product C	170.0

In [42]:
```
#ploting the graph based on product vs average of each
plt.title('product vs Sales')
plt.bar(sales_data_set_order['Sales'], sales_data_set_order['Product'])
plt.show()
```



In [43]:
```
sales_data_set_order['Product'].hist(bins=20)
plt.title(' Store Sales Distribution')
plt.show()
```



In [ ]:

In [101…
```
import pandas as pd
import matplotlib.pyplot as plt
```

In [102…
```
hotel_data=pd.read_csv('Hotel_Dataset.csv')
hotel_data
```

Out[102…

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	E
0	1	20-25	4	Ibis	veg	1300	2	
1	2	30-35	5	LemonTree	Non-Veg	2000	3	
2	3	25-30	6	RedFox	Veg	1322	2	
3	4	20-25	-1	LemonTree	Veg	1234	2	
4	5	35+	3	Ibis	Vegetarian	989	2	
5	6	35+	3	Ibys	Non-Veg	1909	2	
6	7	35+	4	RedFox	Vegetarian	1000	-1	
7	8	20-25	7	LemonTree	Veg	2999	-10	
8	9	25-30	2	Ibis	Non-Veg	3456	3	
9	9	25-30	2	Ibis	Non-Veg	3456	3	
10	10	30-35	5	RedFox	non-Veg	-6755	4	

In [103…
```
len(hotel_data)
```

Out[103…
```
11
```

In [104…
```
hotel_data.duplicated()
```

Out[104…
```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9      True
10    False
dtype: bool
```

In [105…
```
hotel_data.drop_duplicates(inplace=True)
hotel_data
```

Out[105…

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	E
0	1	20-25	4	Ibis	veg	1300	2	
1	2	30-35	5	LemonTree	Non-Veg	2000	3	
2	3	25-30	6	RedFox	Veg	1322	2	
3	4	20-25	-1	LemonTree	Veg	1234	2	
4	5	35+	3	Ibis	Vegetarian	989	2	
5	6	35+	3	Ibys	Non-Veg	1909	2	
6	7	35+	4	RedFox	Vegetarian	1000	-1	
7	8	20-25	7	LemonTree	Veg	2999	-10	
8	9	25-30	2	Ibis	Non-Veg	3456	3	
10	10	30-35	5	RedFox	non-Veg	-6755	4	

In [106…
```
hotel_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10 entries, 0 to 10
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   CustomerID       10 non-null     int64 
 1   Age_Group        10 non-null     object
 2   Rating(1-5)      10 non-null     int64 
 3   Hotel            10 non-null     object
 4   FoodPreference   10 non-null     object
 5   Bill             10 non-null     int64 
 6   NoOfPax          10 non-null     int64 
 7   EstimatedSalary  10 non-null     int64 
 8   Age_Group.1      10 non-null     object
dtypes: int64(5), object(4)
memory usage: 800.0+ bytes
```

In [107…

```
import numpy as np
len(hotel_data)
index=np.array(list(range(0,len(hotel_data))))
index
```

Out[107…

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [108…

```
hotel_data.set_index(index,inplace=True)
hotel_data
```

Out[108…

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	Es
0	1	20-25	4	Ibis	veg	1300	2	
1	2	30-35	5	LemonTree	Non-Veg	2000	3	
2	3	25-30	6	RedFox	Veg	1322	2	
3	4	20-25	-1	LemonTree	Veg	1234	2	
4	5	35+	3	Ibis	Vegetarian	989	2	
5	6	35+	3	Ibys	Non-Veg	1909	2	
6	7	35+	4	RedFox	Vegetarian	1000	-1	
7	8	20-25	7	LemonTree	Veg	2999	-10	
8	9	25-30	2	Ibis	Non-Veg	3456	3	
9	10	30-35	5	RedFox	non-Veg	-6755	4	

In [112…

```
hotel_data.loc[hotel_data.Rating(1-5)<0]=np.nan
hotel_data
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[112], line 1
----> 1 hotel_data.loc[hotel_data.Rating(1-5)<0]=np.nan
      2 hotel_data

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:5902, in NDFrame.__getattr__(self, name)
   5895 if (
   5896     name not in self._internal_names_set
   5897     and name not in self._metadata
   5898     and name not in self._accessors
   5899     and self._info_axis._can_hold_identifiers_and_holds_name(name)
   5900 ):
   5901     return self[name]
-> 5902 return object.__getattribute__(self, name)

AttributeError: 'DataFrame' object has no attribute 'Rating'
```

In [110…

```
hotel_data.Hotel.replace('Ibys','Ibis',inplace=True)
hotel_data
```

Out[110…

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	Es
0	1	20-25	4	Ibis	veg	1300	2	
1	2	30-35	5	LemonTree	Non-Veg	2000	3	
2	3	25-30	6	RedFox	Veg	1322	2	
3	4	20-25	-1	LemonTree	Veg	1234	2	
4	5	35+	3	Ibis	Vegetarian	989	2	
5	6	35+	3	Ibis	Non-Veg	1909	2	
6	7	35+	4	RedFox	Vegetarian	1000	-1	
7	8	20-25	7	LemonTree	Veg	2999	-10	
8	9	25-30	2	Ibis	Non-Veg	3456	3	
9	10	30-35	5	RedFox	non-Veg	-6755	4	

In [ ]:

In [ ]:

In [ ]:

230701036

ARUN MC

II CSE A

In [1]:
```
import seaborn as sns#to compare 2 visualization in matplotlib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline#specially for jupyter notebook
```

In [2]:
```
tips=sns.load_dataset('tips')#inbuilt dataset
tips
```

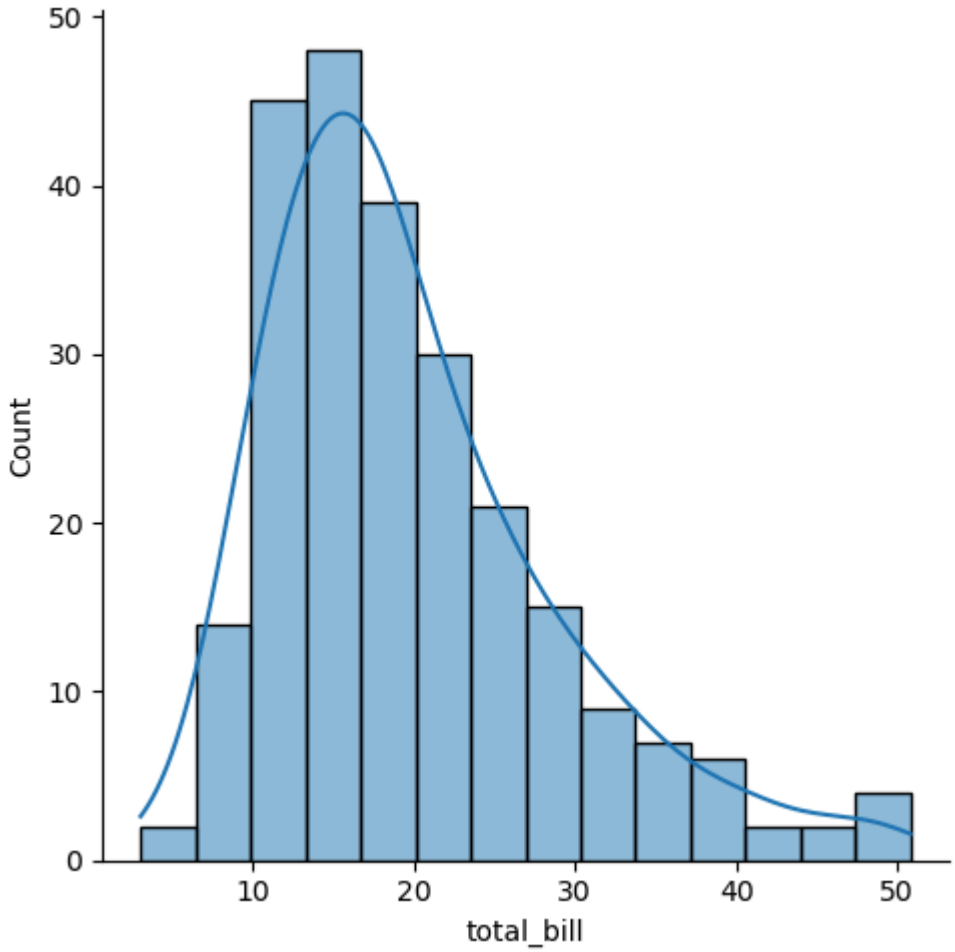
Out[2]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

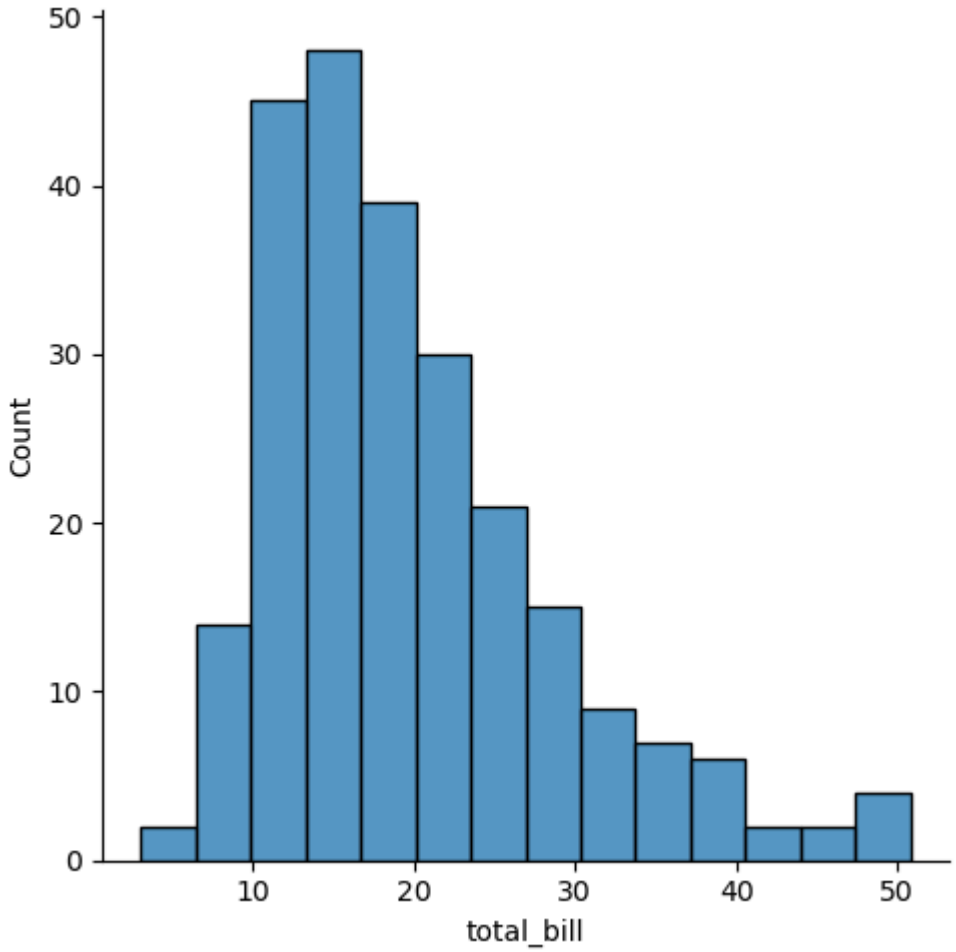
In [3]:
```
sns.displot(tips.total_bill,kde=True)
```

Out[3]:
```
<seaborn.axisgrid.FacetGrid at 0x2b02b92fca0>
```



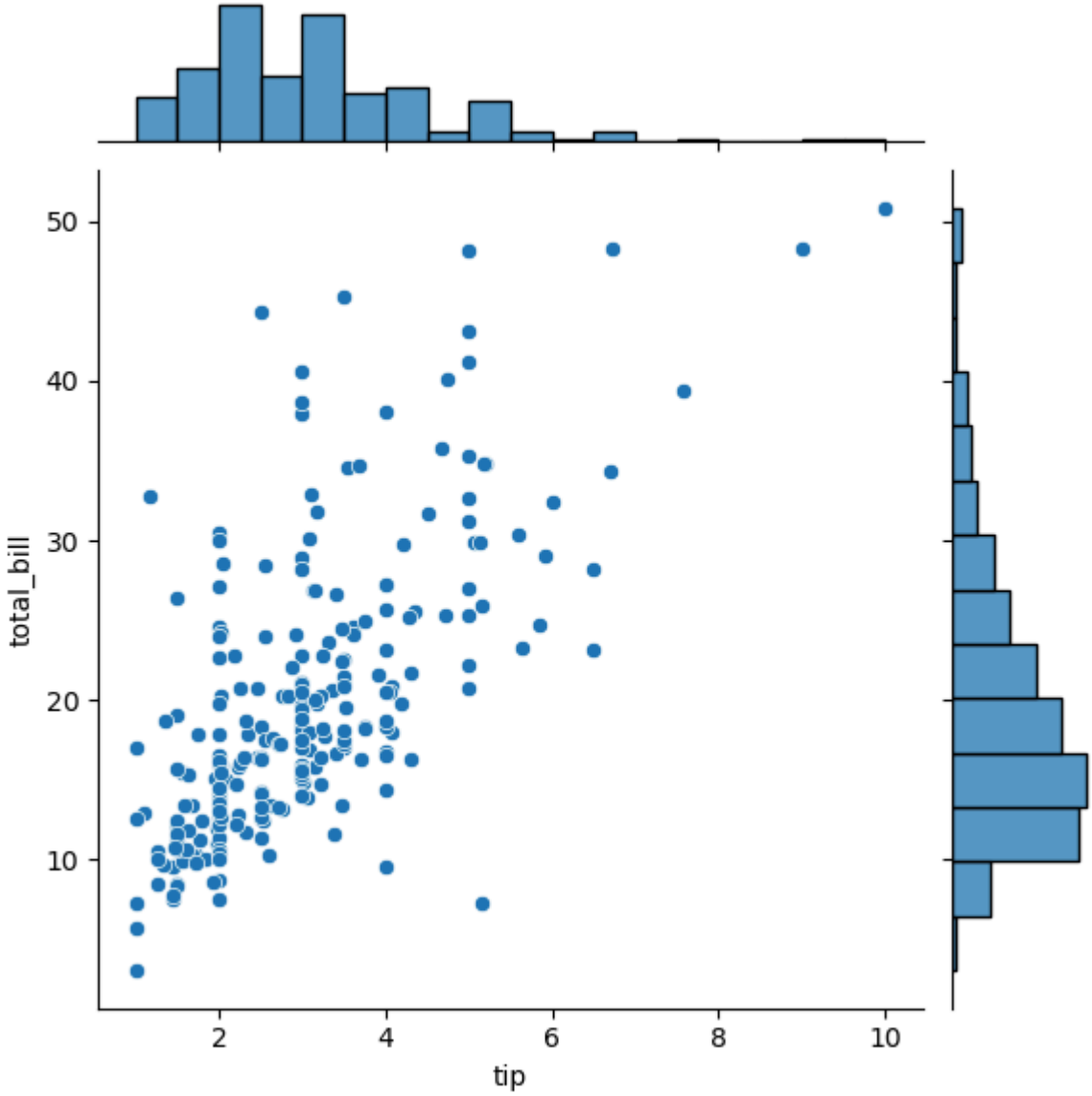
In [4]:
```
sns.displot(tips.total_bill,kde=False)
```

Out[4]:
```
<seaborn.axisgrid.FacetGrid at 0x2b02b92e410>
```



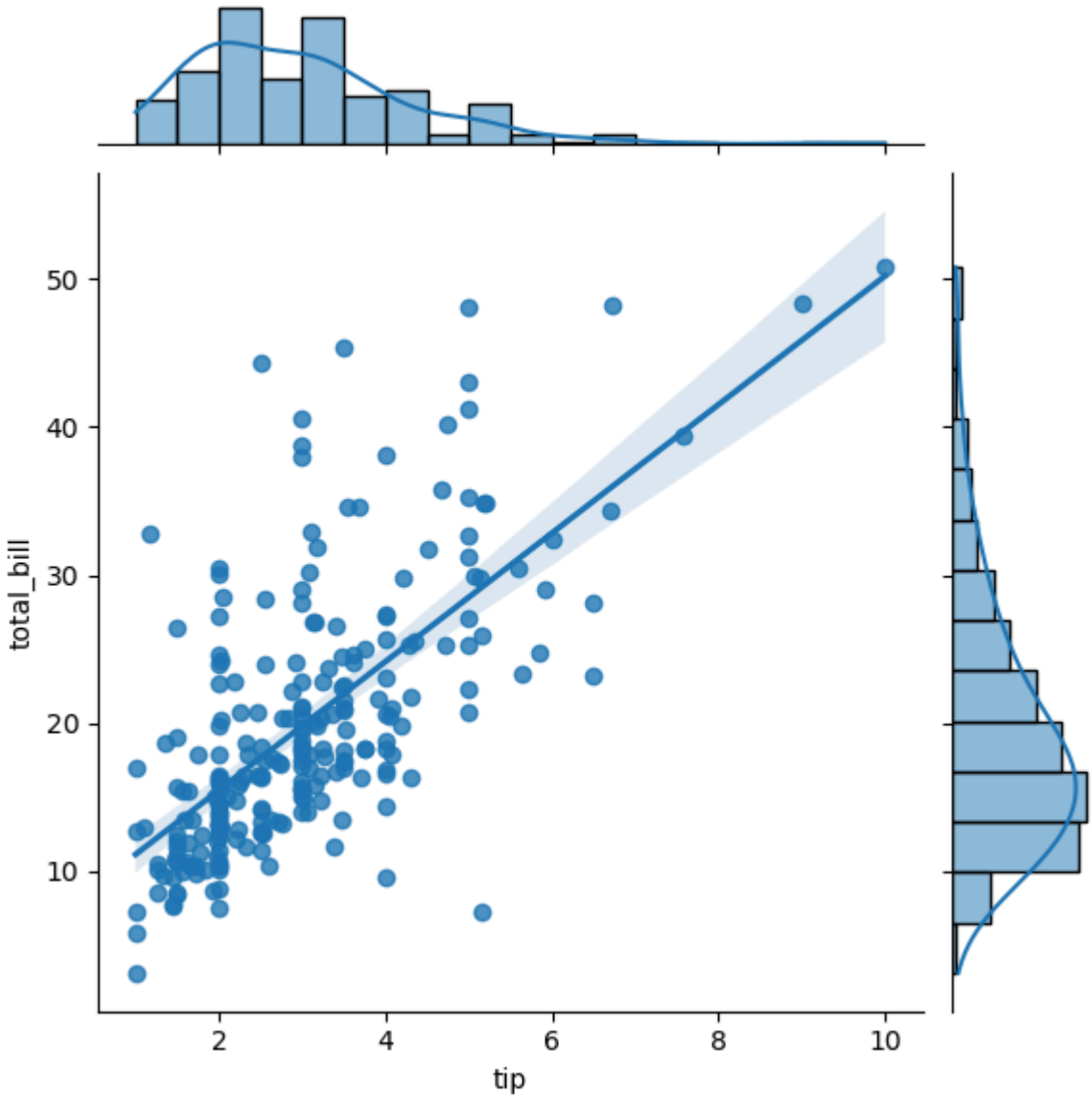
In [5]:
```
sns.jointplot(x=tips.tip,y=tips.total_bill)
#jointplot is used to shpw the correlation between tips
```

Out[5]:
```
<seaborn.axisgrid.JointGrid at 0x2b03321b6d0>
```



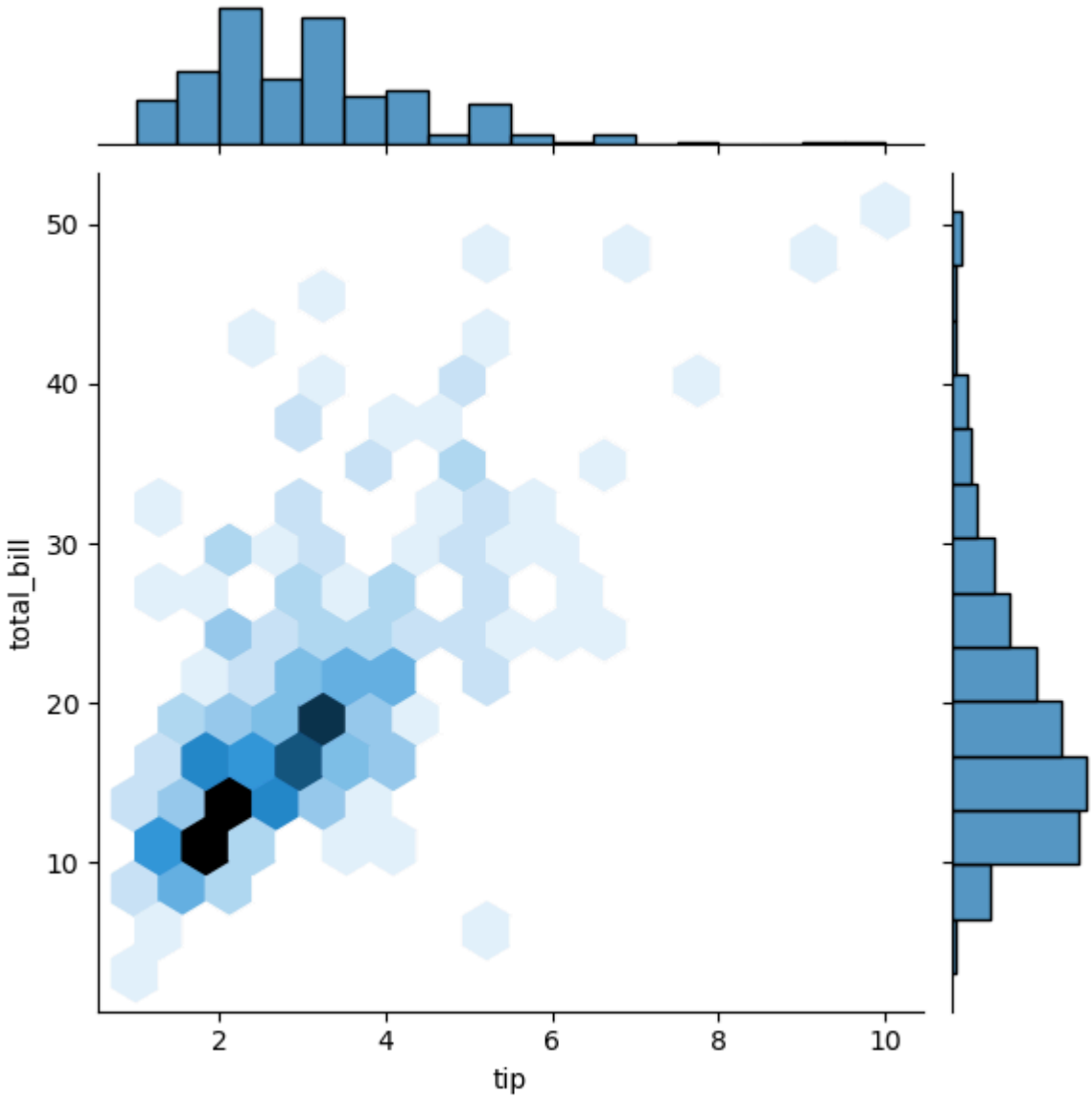
In [6]:
```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind='reg')
# reg is regression graph
```

Out[6]: <seaborn.axisgrid.JointGrid at 0x2b033328b50>



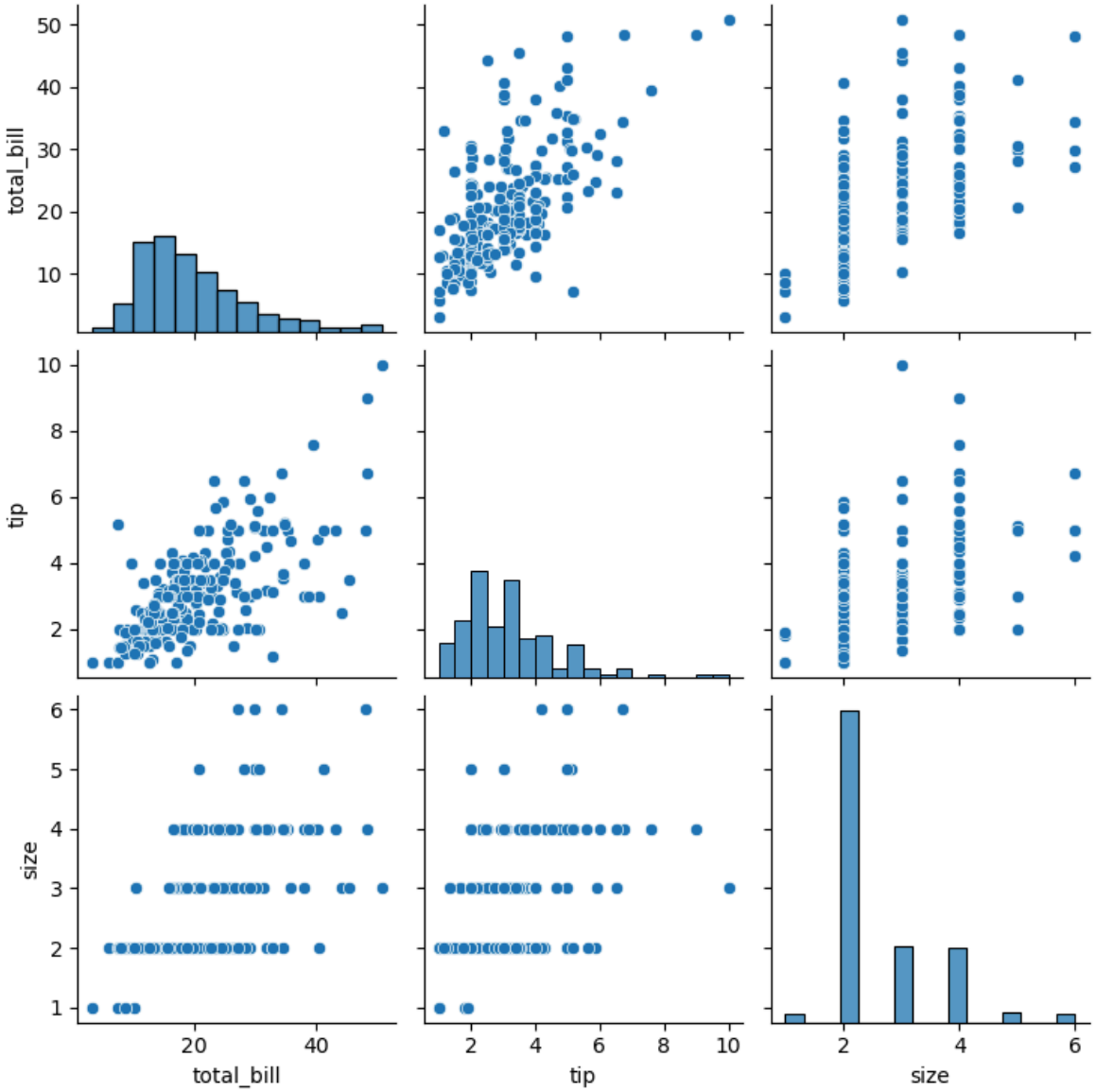
In [9]:
```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind='hex')
```

Out[9]: <seaborn.axisgrid.JointGrid at 0x2b03191e050>



In [10]:
```
sns.pairplot(tips)
```

Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x2b034b836d0>
```

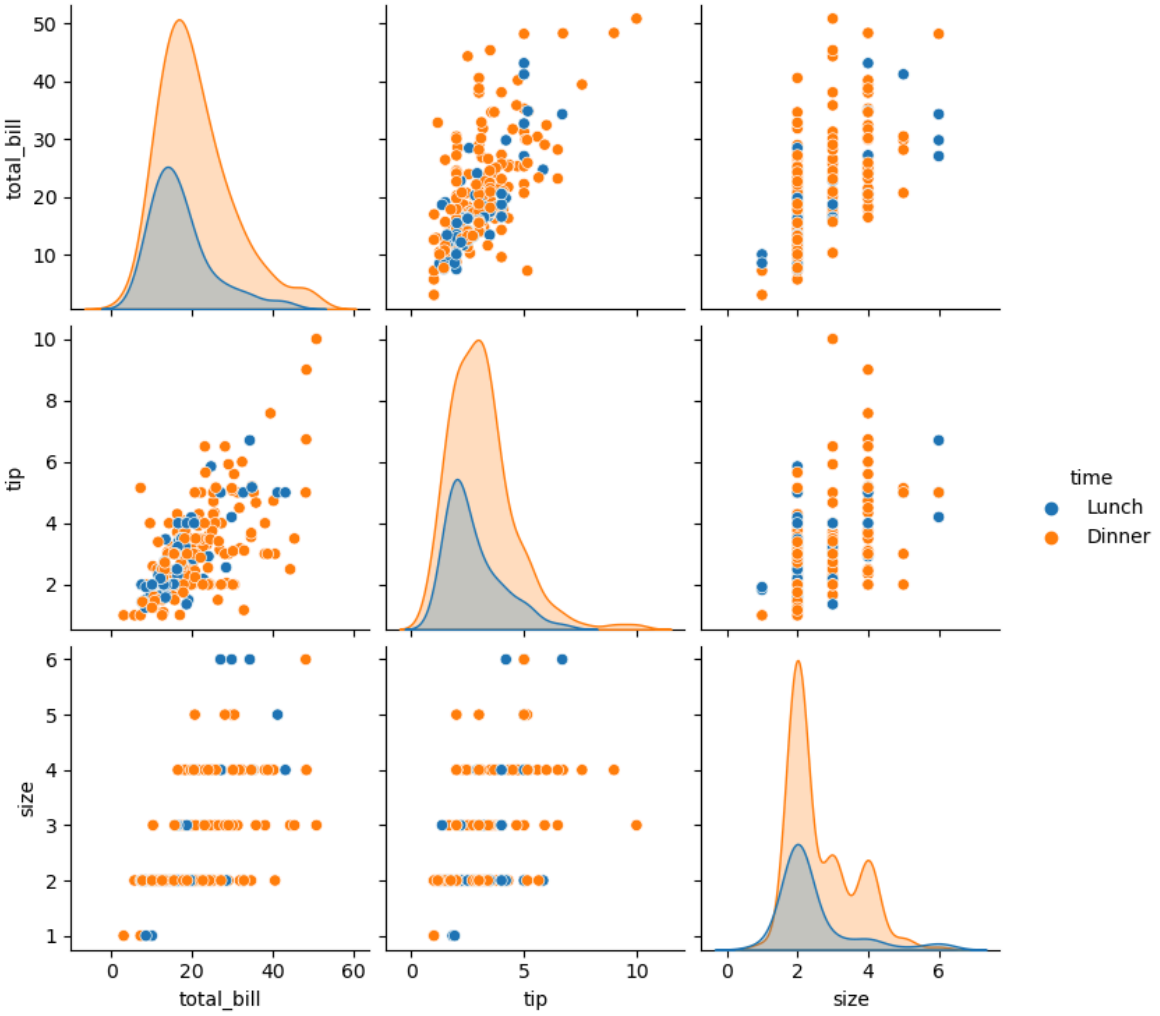


In [12]:
```
tips.time.value_counts()
```

Out[12]:
```
Dinner    176
Lunch      68
Name: time, dtype: int64
```

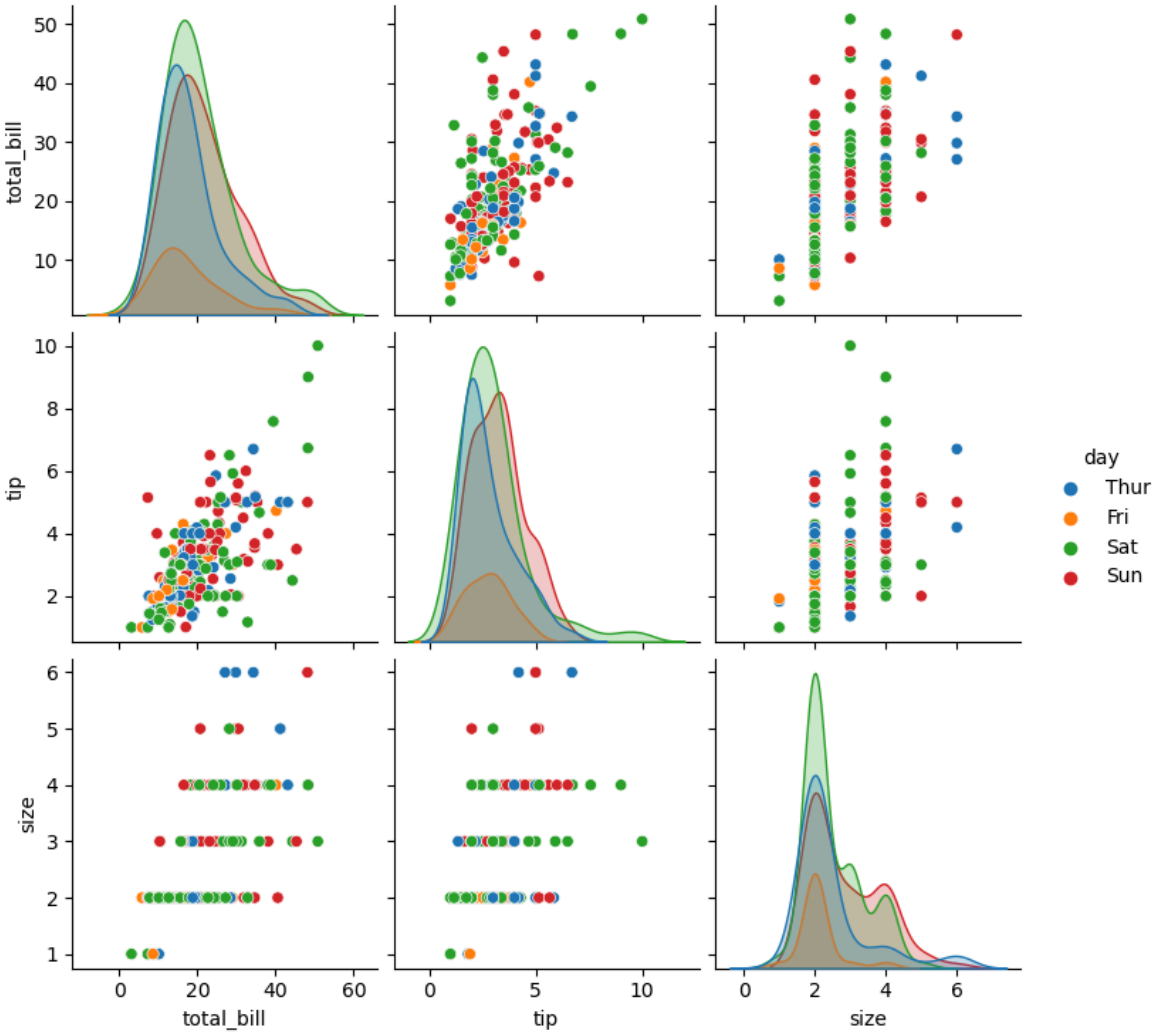
In [13]:
```
sns.pairplot(tips,hue='time')
```

Out[13]:
```
<seaborn.axisgrid.PairGrid at 0x2b03856d330>
```



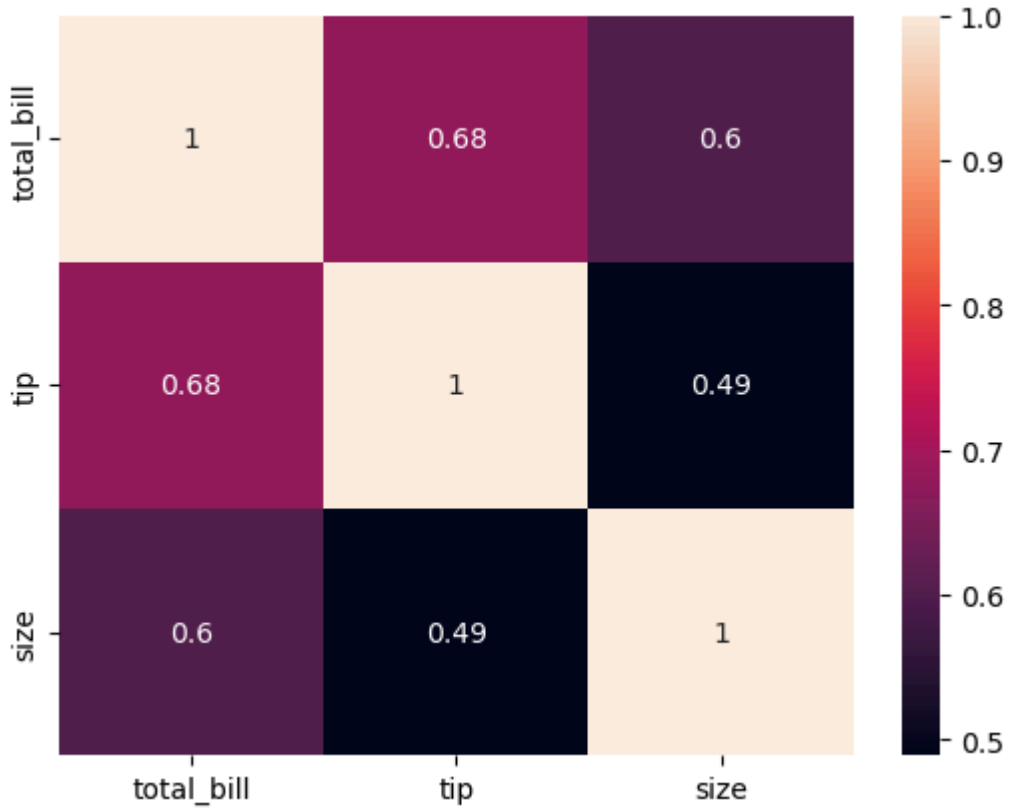
In [14]:
```
sns.pairplot(tips,hue='day')
```

Out[14]:
```
<seaborn.axisgrid.PairGrid at 0x2b03a6f74f0>
```



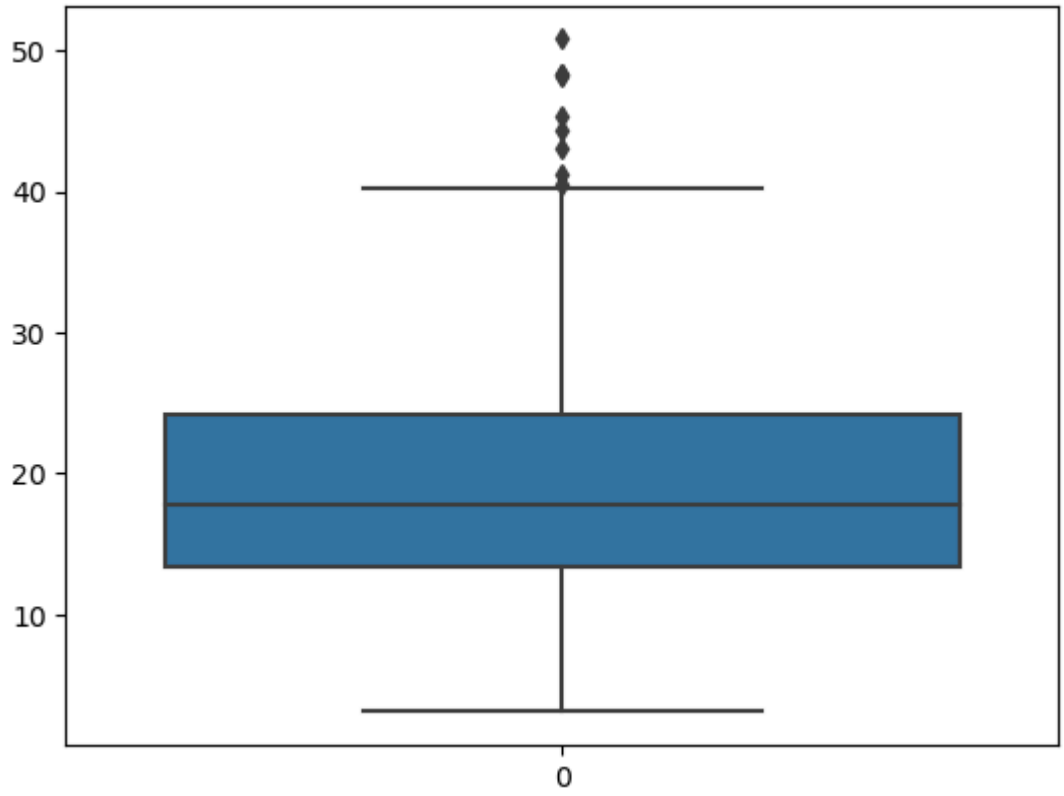
In [15]:
```
sns.heatmap(tips.corr(numeric_only=True),annot=True)#only for numerical data
```

Out[15]: <Axes: >



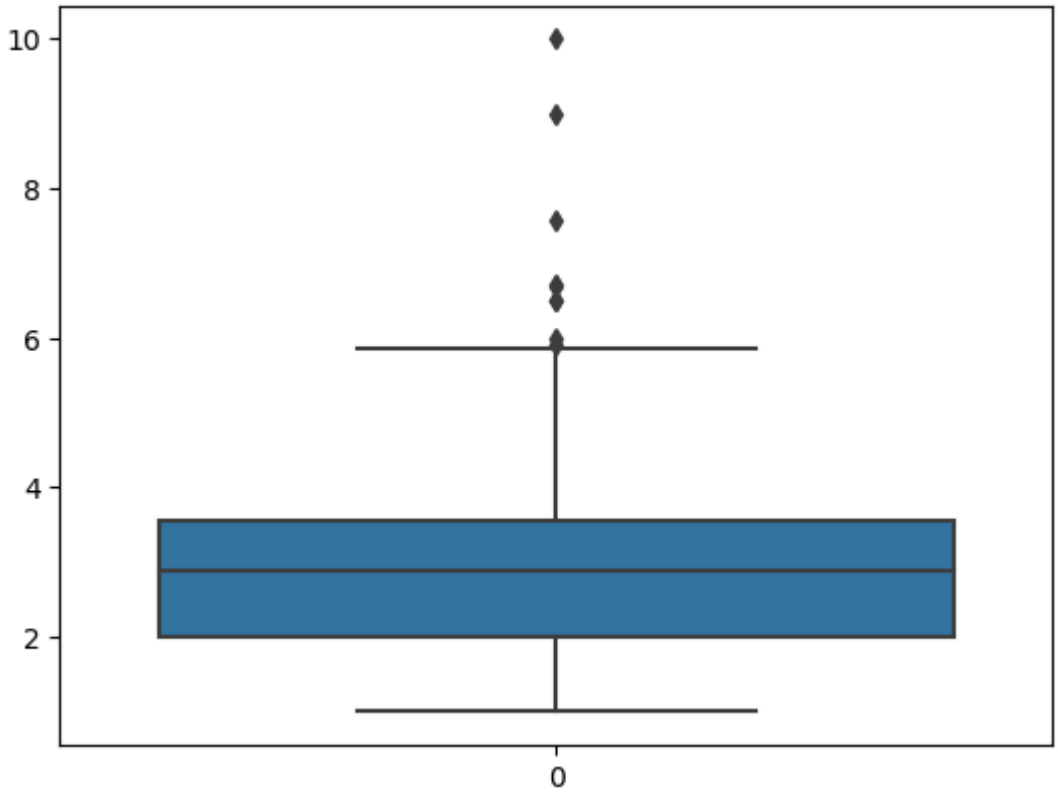
In [16]:
```
sns.boxplot(tips.total_bill)
```

Out[16]: <Axes: >



In [17]:
```
sns.boxplot(tips.tip)
```

Out[17]: <Axes: >



In [18]:
```
sns.countplot(tips.day)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[18], line 1
----> 1 sns.countplot(tips.day)

File ~\anaconda3\lib\site-packages\seaborn\categorical.py:2943, in countplot(dat
a, x, y, hue, order, hue_order, orient, color, palette, saturation, width, dodge,
ax, **kwargs)
   2940 elif x is not None and y is not None:
   2941     raise ValueError("Cannot pass values for both `x` and `y`")
-> 2943 plotter = _CountPlotter(
   2944     x, y, hue, data, order, hue_order,
   2945     estimator, errorbar, n_boot, units, seed,
   2946     orient, color, palette, saturation,
   2947     width, errcolor, errwidth, capsize, dodge
   2948 )
   2950 plotter.value_label = "count"
   2952 if ax is None:

File ~\anaconda3\lib\site-packages\seaborn\categorical.py:1530, in _BarPlotter.__
init__(self, x, y, hue, data, order, hue_order, estimator, errorbar, n_boot, unit
s, seed, orient, color, palette, saturation, width, errcolor, errwidth, capsize,
dodge)
   1525 def __init__(self, x, y, hue, data, order, hue_order,
   1526              estimator, errorbar, n_boot, units, seed,
   1527              orient, color, palette, saturation, width,
   1528              errcolor, errwidth, capsize, dodge):
   1529     """Initialize the plotter."""
-> 1530     self.establish_variables(x, y, hue, data, orient,
   1531                              order, hue_order, units)
   1532     self.establish_colors(color, palette, saturation)
   1533     self.estimate_statistic(estimator, errorbar, n_boot, seed)

File ~\anaconda3\lib\site-packages\seaborn\categorical.py:516, in _CategoricalPlo
tter.establish_variables(self, x, y, hue, data, orient, order, hue_order, units)
    513     plot_data = data
    515 # Convert to a list of arrays, the common representation
--> 516 plot_data = [np.asarray(d, float) for d in plot_data]
    518 # The group names will just be numeric indices
    519 group_names = list(range(len(plot_data)))

File ~\anaconda3\lib\site-packages\seaborn\categorical.py:516, in <listcomp>(.0)
    513     plot_data = data
    515 # Convert to a list of arrays, the common representation
--> 516 plot_data = [np.asarray(d, float) for d in plot_data]
    518 # The group names will just be numeric indices
    519 group_names = list(range(len(plot_data)))

File ~\anaconda3\lib\site-packages\pandas\core\series.py:893, in Series.__array__
(self, dtype)
    846 def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
    847     """
    848     Return the values as a NumPy array.
    849
   (...)
    891           dtype='datetime64[ns]')
    892     """
--> 893     return np.asarray(self._values, dtype)

File ~\anaconda3\lib\site-packages\pandas\core\arrays\_mixins.py:85, in ravel_com
```

```
pat.<locals>.method(self, *args, **kwargs)
     82 @wraps(meth)
     83 def method(self, *args, **kwargs):
     84     if self.ndim == 1:
---> 85         return meth(self, *args, **kwargs)
     87     flags = self._ndarray.flags
     88     flat = self.ravel("K")

File ~\anaconda3\lib\site-packages\pandas\core\arrays\categorical.py:1609, in Categorical.__array__(self, dtype)
   1607 ret = take_nd(self.categories._values, self._codes)
   1608 if dtype and not is_dtype_equal(dtype, self.categories.dtype):
-> 1609     return np.asarray(ret, dtype)
   1610 # When we're a Categorical[ExtensionArray], like Interval,
   1611 # we need to ensure __array__ gets all the way to an
   1612 # ndarray.
   1613 return np.asarray(ret)

ValueError: could not convert string to float: 'Sun'
```

In [ ]:

230701036

ARUN MC

II CSE A

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:
```
```

In [3]:
```
data=pd.read_csv('Iris.csv')
data
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Id           150 non-null    int64  
 1   SepalLength  150 non-null    float64
 2   SepalWidth   150 non-null    float64
 3   PetalLength  150 non-null    float64
 4   PetalWidth   150 non-null    float64
 5   Species      150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [4]:
```
data.describe()
```

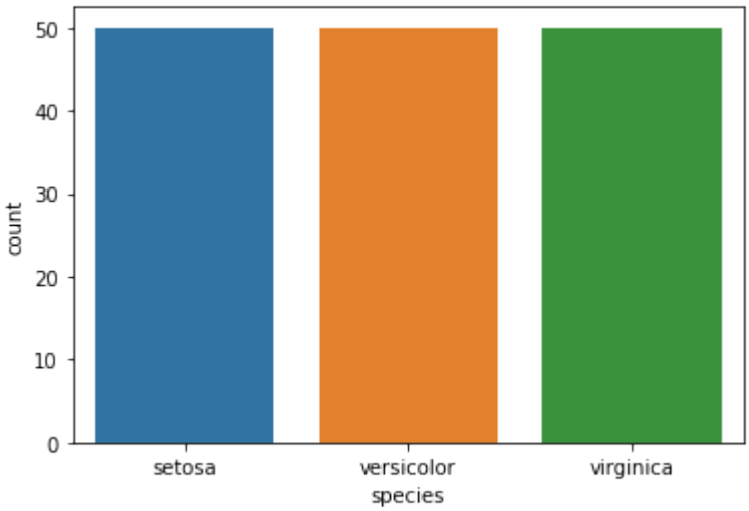
Out[4]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [5]:
```
data.value_counts('species')
```

Out[5]:
```
species
setosa        50
versicolor    50
virginica     50
dtype: int64
```

In [6]:
```
sns.countplot(x='species',data=data,)
plt.show()
```



In [7]:
```
dummies=pd.get_dummies(data.species)
```

In [8]:
```
FinalDataset=pd.concat([pd.get_dummies(data.species),data.iloc[:,[0,1,2,3]]],axi
```

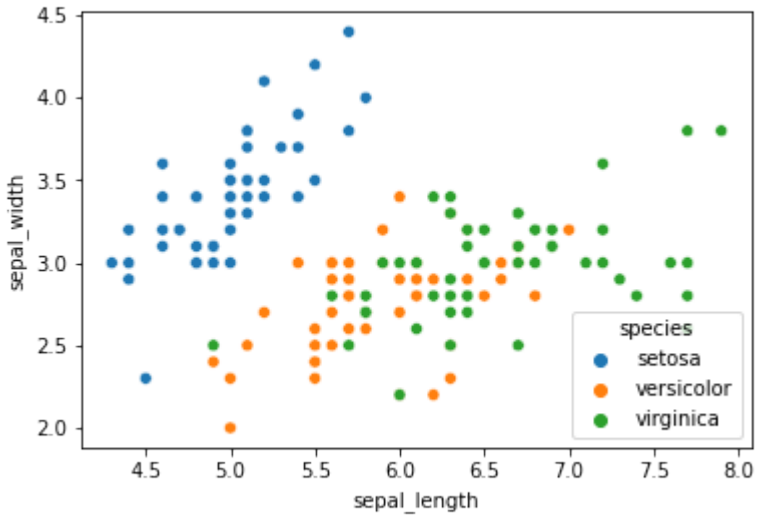
In [9]:
```
FinalDataset.head()
```

Out[9]:

	setosa	versicolor	virginica	sepal_length	sepal_width	petal_length	petal_width
0	1	0	0	5.1	3.5	1.4	0.2
1	1	0	0	4.9	3.0	1.4	0.2
2	1	0	0	4.7	3.2	1.3	0.2
3	1	0	0	4.6	3.1	1.5	0.2
4	1	0	0	5.0	3.6	1.4	0.2

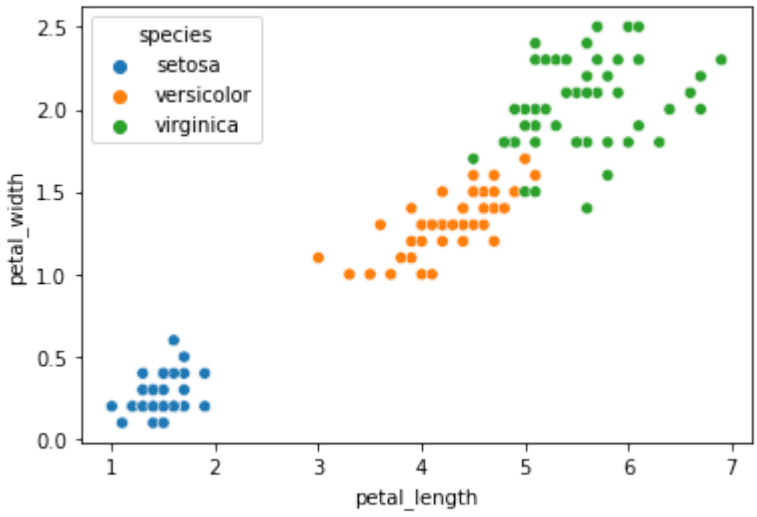
In [11]:
```
sns.scatterplot(x='sepal_length',y='sepal_width',hue='species',data=data,)
```

Out[11]:
```
<AxesSubplot:xlabel='sepal_length', ylabel='sepal_width'>
```



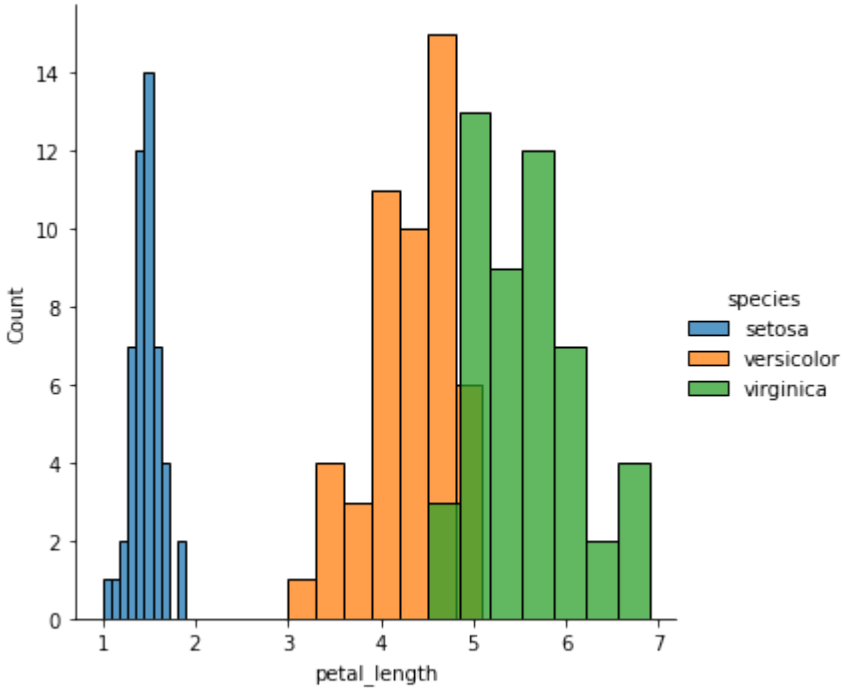
In [14]:
```
sns.scatterplot(x='petal_length',y='petal_width',hue='species',data=data,)
```

Out[14]:
```
<AxesSubplot:xlabel='petal_length', ylabel='petal_width'>
```

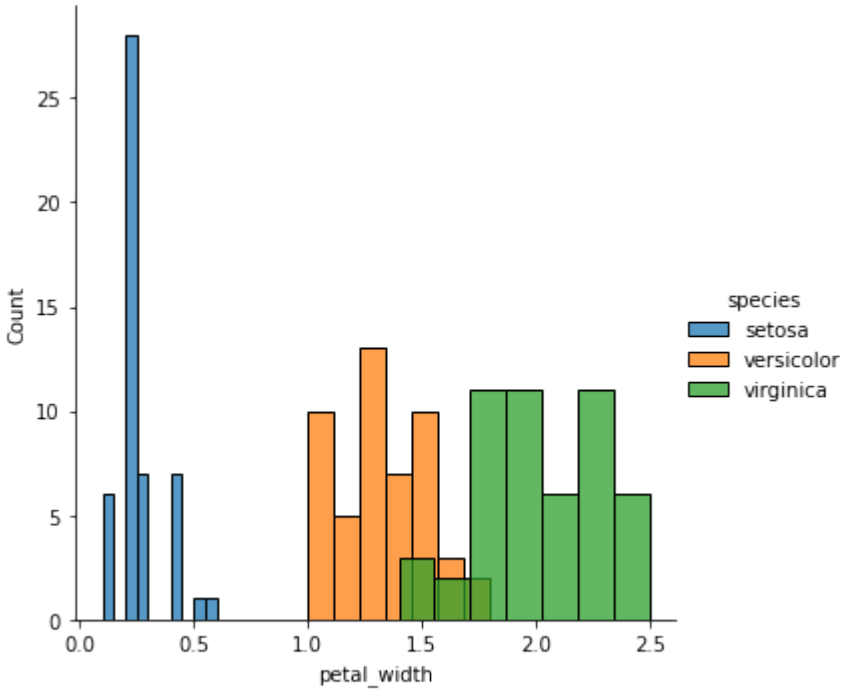


In [ ]:
```
sns.pairplot(data,hue='species',height=3);
```

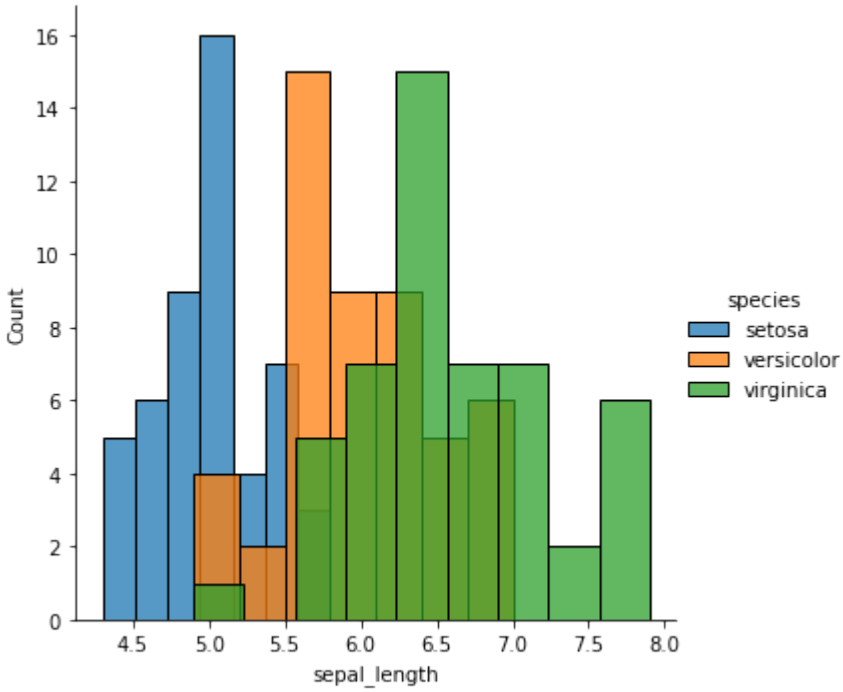
In [15]:
```
sns.FacetGrid(data,hue='species',height=5).map(sns.histplot,'petal_length').add_
plt.show();
```



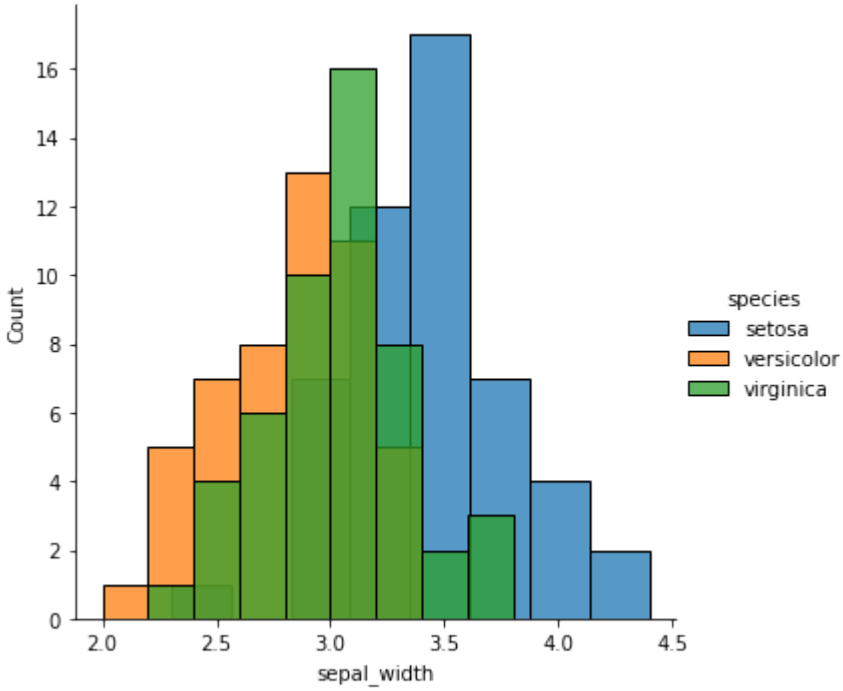
In [17]:
```
sns.FacetGrid(data,hue='species',height=5).map(sns.histplot,'petal_width').add_l
plt.show();
```



In [18]:
```
sns.FacetGrid(data,hue='species',height=5).map(sns.histplot,'sepal_length').add_
plt.show();
```



In [16]:
```
sns.FacetGrid(data,hue='species',height=5).map(sns.histplot,'sepal_width').add_l
plt.show();
```



In [ ]:

230701036

ARUN MC

II CSE A

In [1]:
```
import numpy as np
array=np.random.randint(1,100,9)
array
```

Out[1]:
```
array([69, 41, 25, 26, 17, 92, 20, 87, 81])
```

In [2]:
```
np.sqrt(array)
```

Out[2]:
```
array([8.30662386, 6.40312424, 5.        , 5.09901951, 4.12310563,
       9.59166305, 4.47213595, 9.32737905, 9.        ])
```

In [3]:
```
array.ndim
```

Out[3]:
```
1
```

In [4]:
```
new_array=array.reshape(3,3)
new_array
```

Out[4]:
```
array([[69, 41, 25],
       [26, 17, 92],
       [20, 87, 81]])
```

In [5]:
```
new_array.ndim
```

Out[5]:
```
2
```

In [6]:
```
new_array.ravel()
```

Out[6]:
```
array([69, 41, 25, 26, 17, 92, 20, 87, 81])
```

In [7]:
```
newm=new_array.reshape(3,3)
newm
```

Out[7]:
```
array([[69, 41, 25],
       [26, 17, 92],
       [20, 87, 81]])
```

In [8]:
```
newm[2,1:3]
```

Out[8]:
```
array([87, 81])
```

In [9]:
```
newm[1:2,1:3]
```

Out[9]:
```
array([[17, 92]])
```

In [10]:
```
new_array[0:3,0:0]
```

Out[10]:
```
array([], shape=(3, 0), dtype=int32)
```

In [11]:
```
new_array[0:2,0:1]
```

Out[11]:
```
array([[69],
       [26]])
```

In [12]:
```
new_array[0:3,0:1]
```

Out[12]:
```
array([[69],
       [26],
       [20]])
```

In [ ]:
```

```

230701036

ARUN MC

II CSE A

In [1]:
```
import numpy as np
import pandas as pd
list=[[1,'Smith',50000],[2,'Jones',60000]]
```

In [2]:
```
df=pd.DataFrame(list)
df
```

Out[2]:

	0	1	2
0	1	Smith	50000
1	2	Jones	60000

In [3]:
```
df.columns=['Empd','Name','Salary']
df
```

Out[3]:

	Empd	Name	Salary
0	1	Smith	50000
1	2	Jones	60000

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Empd    2 non-null      int64 
 1   Name    2 non-null      object
 2   Salary  2 non-null      int64 
dtypes: int64(2), object(1)
memory usage: 176.0+ bytes
```

In [7]:
```
df=pd.read_csv("50_Startups.csv")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   R&D Spend        50 non-null     float64
 1   Administration   50 non-null     float64
 2   Marketing Spend  50 non-null     float64
 3   State            50 non-null     object 
 4   Profit           50 non-null     float64
dtypes: float64(4), object(1)
memory usage: 2.1+ KB
```

In [8]:
```
df.head()
```

Out[8]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

In [9]:
```
df.tail()
```

Out[9]:

	R&D Spend	Administration	Marketing Spend	State	Profit
45	1000.23	124153.04	1903.93	New York	64926.08
46	1315.46	115816.21	297114.46	Florida	49490.75
47	0.00	135426.92	0.00	California	42559.73
48	542.05	51743.15	0.00	New York	35673.41
49	0.00	116983.80	45173.06	California	14681.40

In [1]:
```
import numpy as np
import pandas as pd
df=pd.read_csv("Employee.csv")
```

In [2]:
```
df.head()
```

Out[2]:

	Education	JoiningYear	City	PaymentTier	Age	Gender	EverBenched	Experie
0	Bachelors	2017	Bangalore	3	34	Male	No	
1	Bachelors	2013	Pune	1	28	Female	No	
2	Bachelors	2014	New Delhi	3	38	Female	No	
3	Masters	2016	Bangalore	3	27	Male	No	
4	Masters	2017	Pune	3	24	Male	Yes	

In [3]:
```
df.tail()
```

Out[3]:

	Education	JoiningYear	City	PaymentTier	Age	Gender	EverBenched	Exp
4648	Bachelors	2013	Bangalore	3	26	Female	No	
4649	Masters	2013	Pune	2	37	Male	No	
4650	Masters	2018	New Delhi	3	27	Male	No	
4651	Bachelors	2012	Bangalore	3	30	Male	Yes	
4652	Bachelors	2015	Bangalore	3	33	Male	Yes	

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

230701036

ARUN MC

II CSE A

In [2]:
```
import numpy as np
array=np.random.randint(1,100,16)
array
```

Out[2]:
```
array([19, 14,  5, 49, 21, 61, 61, 83, 20, 71, 59,  4,  9, 25, 18, 48])
```

In [3]:
```
array.mean()
```

Out[3]:
```
35.4375
```

In [4]:
```
np.percentile(array,25)
```

Out[4]:
```
17.0
```

In [5]:
```
np.percentile(array,50)
```

Out[5]:
```
23.0
```

In [6]:
```
np.percentile(array,75)
```

Out[6]:
```
59.5
```

In [7]:
```
np.percentile(array,100)
```

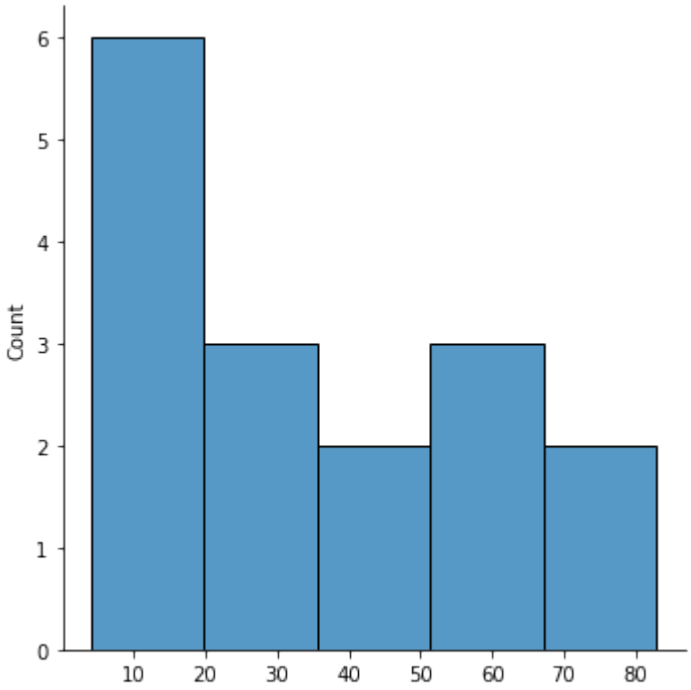
Out[7]:
```
83.0
```

In [8]:
```
def outDetection(array):
    sorted(array)
    Q1,Q3=np.percentile(array,[25,75])
    IQR=Q3-Q1
    lr=Q1-(1.5*IQR)
    ur=Q3+(1.5*IQR)
    return lr,ur
lr,ur=outDetection(array)
lr,ur
```

Out[8]:
```
(-46.75, 123.25)
```

In [9]:
```
import seaborn as sns
%matplotlib inline
sns.displot(array)
```

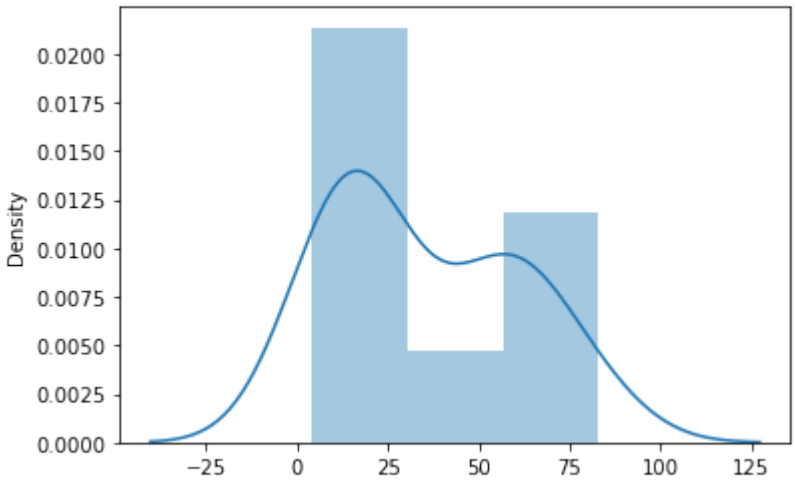
Out[9]:
```
<seaborn.axisgrid.FacetGrid at 0x1c187e16250>
```



In [10]:
```
sns.distplot(array)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future versi
on. Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

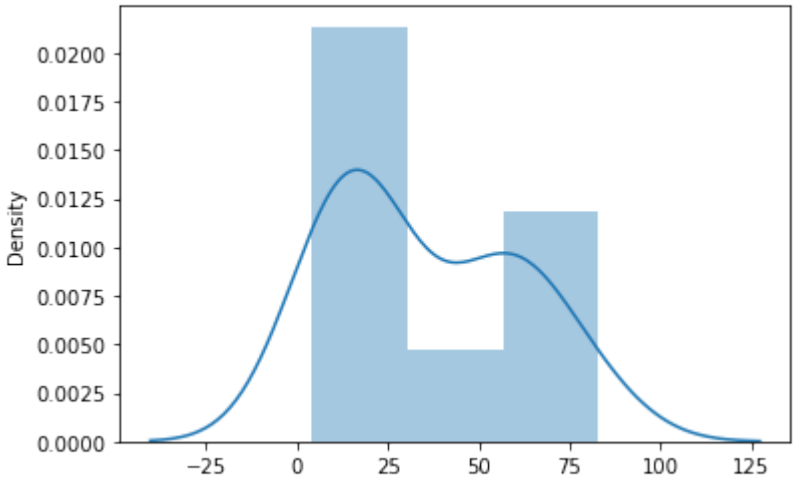
Out[10]: <AxesSubplot:ylabel='Density'>



In [11]:
```
sns.distplot(array)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future versi
on. Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[11]: <AxesSubplot:ylabel='Density'>

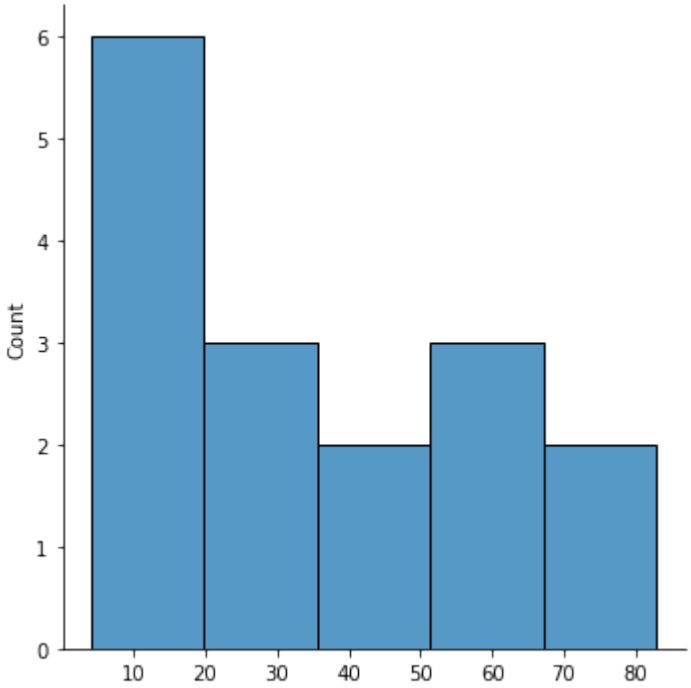


In [12]:
```
new_array=array[(array>lr) & (array<ur)]
new_array
```

Out[12]:
```
array([19, 14,  5, 49, 21, 61, 61, 83, 20, 71, 59,  4,  9, 25, 18, 48])
```

In [13]:
```
sns.displot(new_array)
```

Out[13]:
```
<seaborn.axisgrid.FacetGrid at 0x1c18c35ecd0>
```



In [14]:
```
lr1,ur1=outDetection(new_array)
lr1,ur1
```

Out[14]:
```
(-46.75, 123.25)
```

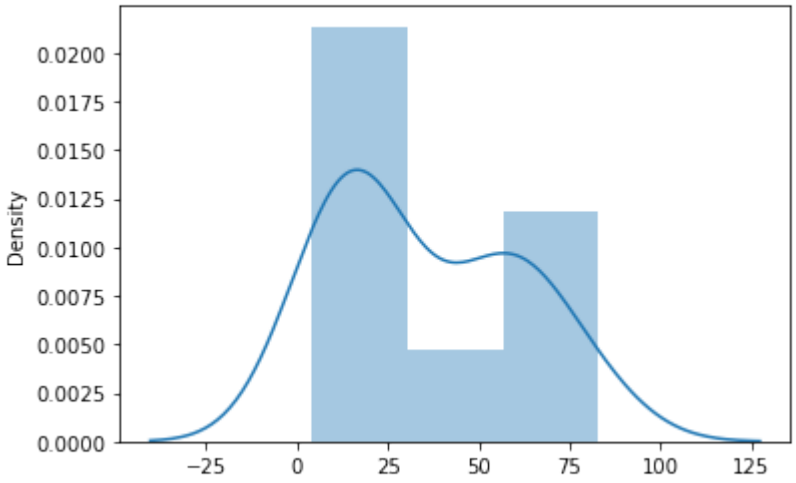
In [15]:
```
final_array=new_array[(new_array>lr1) & (new_array<ur1)]
final_array
```

Out[15]:
```
array([19, 14,  5, 49, 21, 61, 61, 83, 20, 71, 59,  4,  9, 25, 18, 48])
```

In [16]:
```
sns.distplot(final_array)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

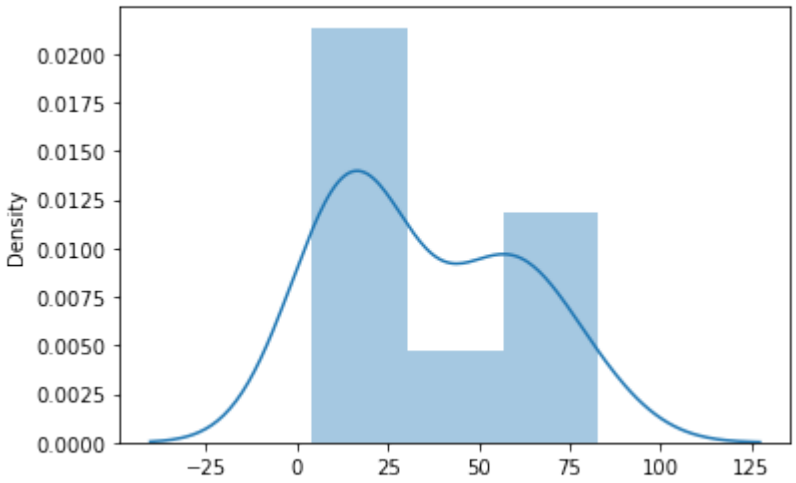
Out[16]: <AxesSubplot:ylabel='Density'>



In [17]:
```
sns.distplot(final_array)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[17]: <AxesSubplot:ylabel='Density'>



In [ ]:

230701036

ARUN MC

II CSE A

In [1]:
```
import numpy as np
import pandas as pd
df=pd.read_csv("Hotel_Dataset.csv")
df
```

Out[1]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	E
0	1	20-25	4	Ibis	veg	1300	2	
1	2	30-35	5	LemonTree	Non-Veg	2000	3	
2	3	25-30	6	RedFox	Veg	1322	2	
3	4	20-25	-1	LemonTree	Veg	1234	2	
4	5	35+	3	Ibis	Vegetarian	989	2	
5	6	35+	3	Ibys	Non-Veg	1909	2	
6	7	35+	4	RedFox	Vegetarian	1000	-1	
7	8	20-25	7	LemonTree	Veg	2999	-10	
8	9	25-30	2	Ibis	Non-Veg	3456	3	
9	9	25-30	2	Ibis	Non-Veg	3456	3	
10	10	30-35	5	RedFox	non-Veg	-6755	4	

In [2]:
```
df.duplicated()
```

Out[2]:
```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9      True
10    False
dtype: bool
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11 entries, 0 to 10
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   CustomerID       11 non-null     int64 
 1   Age_Group        11 non-null     object
 2   Rating(1-5)      11 non-null     int64 
 3   Hotel            11 non-null     object
 4   FoodPreference   11 non-null     object
 5   Bill             11 non-null     int64 
 6   NoOfPax          11 non-null     int64 
 7   EstimatedSalary  11 non-null     int64 
 8   Age_Group.1      11 non-null     object
dtypes: int64(5), object(4)
memory usage: 920.0+ bytes
```

In [4]:
```
df.drop_duplicates(inplace=True)
df
```

Out[4]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	E
0	1	20-25	4	Ibis	veg	1300	2	
1	2	30-35	5	LemonTree	Non-Veg	2000	3	
2	3	25-30	6	RedFox	Veg	1322	2	
3	4	20-25	-1	LemonTree	Veg	1234	2	
4	5	35+	3	Ibis	Vegetarian	989	2	
5	6	35+	3	Ibys	Non-Veg	1909	2	
6	7	35+	4	RedFox	Vegetarian	1000	-1	
7	8	20-25	7	LemonTree	Veg	2999	-10	
8	9	25-30	2	Ibis	Non-Veg	3456	3	
10	10	30-35	5	RedFox	non-Veg	-6755	4	

In [5]:
```
len(df)
```

Out[5]: 10

In [6]:
```
index=np.array(list(range(0,len(df))))
df.set_index(index,inplace=True)
index
```

Out[6]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [7]:
```
df
```

Out[7]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	Es
0	1	20-25	4	Ibis	veg	1300	2	
1	2	30-35	5	LemonTree	Non-Veg	2000	3	
2	3	25-30	6	RedFox	Veg	1322	2	
3	4	20-25	-1	LemonTree	Veg	1234	2	
4	5	35+	3	Ibis	Vegetarian	989	2	
5	6	35+	3	Ibys	Non-Veg	1909	2	
6	7	35+	4	RedFox	Vegetarian	1000	-1	
7	8	20-25	7	LemonTree	Veg	2999	-10	
8	9	25-30	2	Ibis	Non-Veg	3456	3	
9	10	30-35	5	RedFox	non-Veg	-6755	4	

In [8]:
```
df.drop(['Age_Group.1'],axis=1,inplace=True)
df
```

Out[8]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	Es
0	1	20-25	4	Ibis	veg	1300	2	
1	2	30-35	5	LemonTree	Non-Veg	2000	3	
2	3	25-30	6	RedFox	Veg	1322	2	
3	4	20-25	-1	LemonTree	Veg	1234	2	
4	5	35+	3	Ibis	Vegetarian	989	2	
5	6	35+	3	Ibys	Non-Veg	1909	2	
6	7	35+	4	RedFox	Vegetarian	1000	-1	
7	8	20-25	7	LemonTree	Veg	2999	-10	
8	9	25-30	2	Ibis	Non-Veg	3456	3	
9	10	30-35	5	RedFox	non-Veg	-6755	4	

In [9]:
```
df.CustomerID.loc[df.CustomerID<0]=np.nan
df.Bill.loc[df.Bill<0]=np.nan
df.EstimatedSalary.loc[df.EstimatedSalary<0]=np.nan
df
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:1732: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

Out[9]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	Es
0	1.0	20-25	4	Ibis	veg	1300.0	2	
1	2.0	30-35	5	LemonTree	Non-Veg	2000.0	3	
2	3.0	25-30	6	RedFox	Veg	1322.0	2	
3	4.0	20-25	-1	LemonTree	Veg	1234.0	2	
4	5.0	35+	3	Ibis	Vegetarian	989.0	2	
5	6.0	35+	3	Ibys	Non-Veg	1909.0	2	
6	7.0	35+	4	RedFox	Vegetarian	1000.0	-1	
7	8.0	20-25	7	LemonTree	Veg	2999.0	-10	
8	9.0	25-30	2	Ibis	Non-Veg	3456.0	3	
9	10.0	30-35	5	RedFox	non-Veg	NaN	4	

In [10]:
```
df['NoOfPax'].loc[(df['NoOfPax']<1) | (df['NoOfPax']>20)]=np.nan
df
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:1732: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

Out[10]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	Es
0	1.0	20-25	4	Ibis	veg	1300.0	2.0	
1	2.0	30-35	5	LemonTree	Non-Veg	2000.0	3.0	
2	3.0	25-30	6	RedFox	Veg	1322.0	2.0	
3	4.0	20-25	-1	LemonTree	Veg	1234.0	2.0	
4	5.0	35+	3	Ibis	Vegetarian	989.0	2.0	
5	6.0	35+	3	Ibys	Non-Veg	1909.0	2.0	
6	7.0	35+	4	RedFox	Vegetarian	1000.0	NaN	
7	8.0	20-25	7	LemonTree	Veg	2999.0	NaN	
8	9.0	25-30	2	Ibis	Non-Veg	3456.0	3.0	
9	10.0	30-35	5	RedFox	non-Veg	NaN	4.0	

In [11]:
```
df.Age_Group.unique()
```

Out[11]:
```
array(['20-25', '30-35', '25-30', '35+'], dtype=object)
```

In [12]:
```
df.Hotel.unique()
```

Out[12]:
```
array(['Ibis', 'LemonTree', 'RedFox', 'Ibys'], dtype=object)
```

In [13]:
```
df.Hotel.replace(['Ibys'],'Ibis',inplace=True)
df.FoodPreference.unique
```

Out[13]:
```
<bound method Series.unique of 0           veg
1       Non-Veg
2           Veg
3           Veg
4    Vegetarian
5       Non-Veg
6    Vegetarian
7           Veg
8       Non-Veg
9       non-Veg
Name: FoodPreference, dtype: object>
```

In [14]:
```
df.FoodPreference.replace(['Vegetarian','veg'],'Veg',inplace=True)
df.FoodPreference.replace(['non-Veg'],'Non-Veg',inplace=True)
```

In [15]:
```
df.EstimatedSalary.fillna(round(df.EstimatedSalary.mean()),inplace=True)
df.NoOfPax.fillna(round(df.NoOfPax.median()),inplace=True)
df['Rating(1-5)'].fillna(round(df['Rating(1-5)'].median()), inplace=True)
df.Bill.fillna(round(df.Bill.mean()),inplace=True)
df
```

Out[15]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	Es
0	1.0	20-25	4	Ibis	Veg	1300.0	2.0	
1	2.0	30-35	5	LemonTree	Non-Veg	2000.0	3.0	
2	3.0	25-30	6	RedFox	Veg	1322.0	2.0	
3	4.0	20-25	-1	LemonTree	Veg	1234.0	2.0	
4	5.0	35+	3	Ibis	Veg	989.0	2.0	
5	6.0	35+	3	Ibis	Non-Veg	1909.0	2.0	
6	7.0	35+	4	RedFox	Veg	1000.0	2.0	
7	8.0	20-25	7	LemonTree	Veg	2999.0	2.0	
8	9.0	25-30	2	Ibis	Non-Veg	3456.0	3.0	
9	10.0	30-35	5	RedFox	Non-Veg	1801.0	4.0	

In [ ]:

In [ ]:

230701036

ARUN MC

II CSE A

In [3]:
```
import numpy as np
import pandas as pd
df=pd.read_csv('pre_process_datasample.csv')
df
```

Out[3]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

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

Out[4]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes

In [5]:
```
df.Country.fillna(df.Country.mode()[0],inplace=True)
features=df.iloc[:,:-1].values
```

In [6]:
```
label=df.iloc[:,-1].values
```

In [7]:
```
from sklearn.impute import SimpleImputer
age=SimpleImputer(strategy="mean",missing_values=np.nan)
Salary=SimpleImputer(strategy="mean",missing_values=np.nan)
age.fit(features[:,[1]])
```

Out[7]:
```
SimpleImputer()
```

In [8]:
```
Salary.fit(features[:,[2]])
```

Out[8]:
```
SimpleImputer()
```

In [9]:
```
SimpleImputer()
```

Out[9]:
```
SimpleImputer()
```

In [10]:
```
features[:,[1]]=age.transform(features[:,[1]])
features[:,[2]]=Salary.transform(features[:,[2]])
features
```

Out[10]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 63777.77777777778],
       ['France', 35.0, 58000.0],
       ['Spain', 38.77777777777778, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

In [28]:
```
from sklearn.preprocessing import OneHotEncoder
oh = OneHotEncoder()
Country=oh.fit_transform(features[:,[0]]).toarray()
```

In [29]:
```
final_set=np.concatenate((Country,features[:,[1]],features[:,[2]]),axis=1)
final_set
```

Out[29]:
```
array([[1.0, 0.0, 0.0, 44.0, 72000.0],
       [0.0, 0.0, 1.0, 27.0, 48000.0],
       [0.0, 1.0, 0.0, 30.0, 54000.0],
       [0.0, 0.0, 1.0, 38.0, 61000.0],
       [0.0, 1.0, 0.0, 40.0, 63777.77777777778],
       [1.0, 0.0, 0.0, 35.0, 58000.0],
       [0.0, 0.0, 1.0, 38.77777777777778, 52000.0],
       [1.0, 0.0, 0.0, 48.0, 79000.0],
       [0.0, 1.0, 0.0, 50.0, 83000.0],
       [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

In [30]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit(final_set)
feat_standard_scaler=sc.transform(final_set)
feat_standard_scaler
```

Out[30]:
```
array([[ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
         7.58874362e-01,  7.49473254e-01],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
        -1.71150388e+00, -1.43817841e+00],
       [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
        -1.27555478e+00, -8.91265492e-01],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
        -1.13023841e-01, -2.53200424e-01],
       [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
         1.77608893e-01,  6.63219199e-16],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
        -5.48972942e-01, -5.26656882e-01],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
         0.00000000e+00, -1.07356980e+00],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
         1.34013983e+00,  1.38753832e+00],
       [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
         1.63077256e+00,  1.75214693e+00],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
        -2.58340208e-01,  2.93712492e-01]])
```

In [31]:
```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler(feature_range=(0,1))
mms.fit(final_set)
feat_minmax_scaler=mms.transform(final_set)
feat_minmax_scaler
```

Out[31]:
```
array([[1.        , 0.        , 0.        , 0.73913043, 0.68571429],
       [0.        , 0.        , 1.        , 0.        , 0.        ],
       [0.        , 1.        , 0.        , 0.13043478, 0.17142857],
       [0.        , 0.        , 1.        , 0.47826087, 0.37142857],
       [0.        , 1.        , 0.        , 0.56521739, 0.45079365],
       [1.        , 0.        , 0.        , 0.34782609, 0.28571429],
       [0.        , 0.        , 1.        , 0.51207729, 0.11428571],
       [1.        , 0.        , 0.        , 0.91304348, 0.88571429],
       [0.        , 1.        , 0.        , 1.        , 1.        ],
       [1.        , 0.        , 0.        , 0.43478261, 0.54285714]])
```

In [ ]:

In [ ]:

230701036

ARUN MC

II CSE A

In [1]:
```
import numpy as np
import pandas as pd
df=pd.read_csv("pre_process_datasample.csv")
df
```

Out[1]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

In [2]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Country    10 non-null     object 
 1   Age        9 non-null      float64
 2   Salary     9 non-null      float64
 3   Purchased  10 non-null     object 
dtypes: float64(2), object(2)
memory usage: 448.0+ bytes
```

In [3]:
```
df.Country.mode()
```

Out[3]:
```
0    France
dtype: object
```

In [4]:
```
df.Country.mode()[0]
```

Out[4]: 'France'

In [5]:
```
type(df.Country.mode())
```

Out[5]: pandas.core.series.Series

In [6]:
```
df.Country.fillna(df.Country.mode()[0],inplace=True)
```

In [7]:
```
df.Age.fillna(df.Age.median(),inplace=True)
```

In [8]:
```
df.Salary.fillna(round(df.Salary.mean()),inplace=True)
```

In [9]:
```
df
```

Out[9]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	63778.0	Yes
5	France	35.0	58000.0	Yes
6	Spain	38.0	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

In [10]:
```
pd.get_dummies(df.Country)
```

Out[10]:

	France	Germany	Spain
0	1	0	0
1	0	0	1
2	0	1	0
3	0	0	1
4	0	1	0
5	1	0	0
6	0	0	1
7	1	0	0
8	0	1	0
9	1	0	0

In [11]:
```
updated_dataset=pd.concat([pd.get_dummies(df.Country),df.iloc[:,[1,2,3]]],axis=1
updated_dataset
```

Out[11]:

	France	Germany	Spain	Age	Salary	Purchased
0	1	0	0	44.0	72000.0	No
1	0	0	1	27.0	48000.0	Yes
2	0	1	0	30.0	54000.0	No
3	0	0	1	38.0	61000.0	No
4	0	1	0	40.0	63778.0	Yes
5	1	0	0	35.0	58000.0	Yes
6	0	0	1	38.0	52000.0	No
7	1	0	0	48.0	79000.0	Yes
8	0	1	0	50.0	83000.0	No
9	1	0	0	37.0	67000.0	Yes

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Country    10 non-null     object 
 1   Age        10 non-null     float64
 2   Salary     10 non-null     float64
 3   Purchased  10 non-null     object 
dtypes: float64(2), object(2)
memory usage: 448.0+ bytes
```

In [13]:
```
updated_dataset
```

Out[13]:

	France	Germany	Spain	Age	Salary	Purchased
0	1	0	0	44.0	72000.0	No
1	0	0	1	27.0	48000.0	Yes
2	0	1	0	30.0	54000.0	No
3	0	0	1	38.0	61000.0	No
4	0	1	0	40.0	63778.0	Yes
5	1	0	0	35.0	58000.0	Yes
6	0	0	1	38.0	52000.0	No
7	1	0	0	48.0	79000.0	Yes
8	0	1	0	50.0	83000.0	No
9	1	0	0	37.0	67000.0	Yes

In [ ]:

In [ ]:

230701036

ARUN MC

II CSE A

In [39]:
```
import numpy as np
import pandas as pd
```

In [40]:
```
salary_data=pd.read_csv("Salary_data.csv")
salary_data
```

Out[40]:

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891
5	2.9	56642
6	3.0	60150
7	3.2	54445
8	3.2	64445
9	3.7	57189
10	3.9	63218
11	4.0	55794
12	4.0	56957
13	4.1	57081
14	4.5	61111
15	4.9	67938
16	5.1	66029
17	5.3	83088
18	5.9	81363
19	6.0	93940
20	6.8	91738
21	7.1	98273
22	7.9	101302
23	8.2	113812
24	8.7	109431
25	9.0	105582
26	9.5	116969
27	9.6	112635
28	10.3	122391
29	10.5	121872

In [41]:
```
salary_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 608.0 bytes
```

In [42]:
```
#machine learing introduction
salary_data.dropna(inplace=True)
```

In [43]:
```
salary_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 608.0 bytes
```

In [44]:
```
salary_data.describe()# to give the statistics decription of the data set
```

Out[44]:

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

In [45]:
```
#from the dataset years of experience is feature i.e input
# "    "    "     salary is label i.e output
# here we use superised model
```

In [46]:
```
features=salary_data.iloc[:,[0]].values
# it select all the rows of the 0th column
label=salary_data.iloc[:,[1]].values
# it select all the rows of the 1th column
```

In [47]:
```
from sklearn.model_selection import train_test_split
#split the data into train with 80% of the entire data, test with 20% of the ent
X_train,X_test,Y_train,Y_test=train_test_split(features,label,test_size=0.2,rand
#test_size=0.2 refer to the 20 % of data are taken for testing remaining all for
```

In [48]:
```
#scikit is the expansion for sk; here we use linear regression algorithm
from sklearn.linear_model import LinearRegression
model=LinearRegression()
#the below is for traning the model
model.fit(X_train,Y_train)
```

Out[48]:
```
▾ LinearRegression

LinearRegression()
```

In [49]:
```
#accuracy of the train set
model.score(X_train,Y_train)
```

Out[49]:
```
0.9603182547438908
```

In [50]:
```
#accuracy of the test set
model.score(X_test,Y_test)
```

Out[50]:
```
0.9184170849214232
```

In [51]:
```
#here we use linear regression so, wkt y=mx+c is the straight line ;where m=slop
```

In [52]:
```
model.coef_
```

Out[52]:
```
array([[9281.30847068]])
```

In [53]:
```
model.intercept_
```

Out[53]:
```
array([27166.73682891])
```

In [54]:
```
#pickle is for converting memory object into file
import pickle
pickle.dump(model,open('SalaryPred.model','wb'))
```

In [55]:
```
model=pickle.load(open('SalaryPred.model','rb'))
```

In [64]:
```
years=float(input('Enter the years of experience : '))
years_np=np.array([[years]])
salary=model.predict(years_np)
print("The estimated salary for %.1f years is %.2f"%(years,salary))
```

```
Enter the years of experience : 1.1
The estimated salary for 1.1 years is 37376.18
```

In [ ]:

230701036

ARUN MC

II CSE A

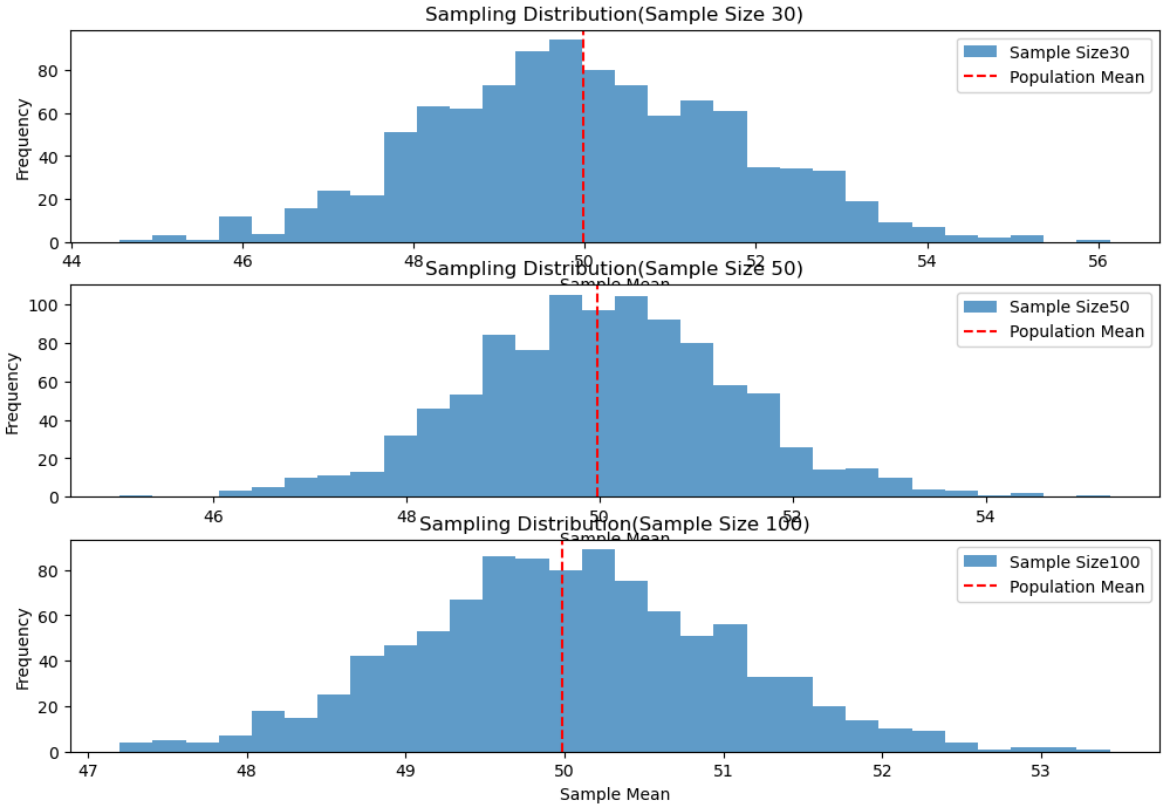
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [6]:
```
p_mean=50
p_standard_deviation=10
p_size=100000
population=np.random.normal(p_mean,p_standard_deviation,p_size)
```

In [7]:
```
sample_size=[30,50,100]
num_sample=1000#no. of sample for each sample size
sample_mean={}
```

In [9]:
```
for i in sample_size:
    sample_mean[i]=[]
    for j in range(num_sample):
        sample=np.random.choice(population,size=i,replace=False)
        sample_mean[i].append(np.mean(sample))
```

In [11]:
```
plt.figure(figsize=(12,8))
for i, size in enumerate(sample_size):
    plt.subplot(len(sample_size),1,i+1)
    plt.hist(sample_mean[size],bins=30,alpha=0.7,label=f'Sample Size{size}')
    plt.axvline(np.mean(population),color='red',linestyle='dashed',linewidth=1.5
    plt.title(f'Sampling Distribution(Sample Size {size})')
    plt.xlabel('Sample Mean')
    plt.ylabel('Frequency')
    plt.legend()
```



In [12]:
```
plt.tight_layout()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```

In [20]:
```
population
```

Out[20]:
```
array([38.46712741, 71.69909629, 31.14996005, ..., 41.3178591 ,
       29.0701355 , 51.65231964])
```

In [14]:
```
len(population)
```

Out[14]:
```
100000
```

In [ ]:

In [ ]:

230701036

ARUN MC

II CSE A

In [1]:
```
import pandas as pd
import numpy as np
```

In [3]:
```
data=pd.read_csv('Social_Network_Ads.csv')
data.head(20)
```

Out[3]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0
16	15733883	Male	47	25000	1
17	15617482	Male	45	26000	1
18	15704583	Male	46	28000	1
19	15621083	Female	48	29000	1

In [71]:
```
features=data.iloc[0:,2:4].values#input values
features
```

Out[71]:
```
array([[    19,  19000],
       [    35,  20000],
       [    26,  43000],
       [    27,  57000],
       [    19,  76000],
       [    27,  58000],
       [    27,  84000],
       [    32, 150000],
       [    25,  33000],
       [    35,  65000],
       [    26,  80000],
       [    26,  52000],
       [    20,  86000],
       [    32,  18000],
       [    18,  82000],
       [    29,  80000],
       [    47,  25000],
       [    45,  26000],
       [    46,  28000],
       [    48,  29000],
       [    45,  22000],
       [    47,  49000],
       [    48,  41000],
       [    45,  22000],
       [    46,  23000],
       [    47,  20000],
       [    49,  28000],
       [    47,  30000],
       [    29,  43000],
       [    31,  18000],
       [    31,  74000],
       [    27, 137000],
       [    21,  16000],
       [    28,  44000],
       [    27,  90000],
       [    35,  27000],
       [    33,  28000],
       [    30,  49000],
       [    26,  72000],
       [    27,  31000],
       [    27,  17000],
       [    33,  51000],
       [    35, 108000],
       [    30,  15000],
       [    28,  84000],
       [    23,  20000],
       [    25,  79000],
       [    27,  54000],
       [    30, 135000],
       [    31,  89000],
       [    24,  32000],
       [    18,  44000],
       [    29,  83000],
       [    35,  23000],
       [    27,  58000],
       [    24,  55000],
       [    23,  48000],
       [    28,  79000],
       [    22,  18000],
       [    32, 117000],
```

```
[    27,  20000],
[    25,  87000],
[    23,  66000],
[    32, 120000],
[    59,  83000],
[    24,  58000],
[    24,  19000],
[    23,  82000],
[    22,  63000],
[    31,  68000],
[    25,  80000],
[    24,  27000],
[    20,  23000],
[    33, 113000],
[    32,  18000],
[    34, 112000],
[    18,  52000],
[    22,  27000],
[    28,  87000],
[    26,  17000],
[    30,  80000],
[    39,  42000],
[    20,  49000],
[    35,  88000],
[    30,  62000],
[    31, 118000],
[    24,  55000],
[    28,  85000],
[    26,  81000],
[    35,  50000],
[    22,  81000],
[    30, 116000],
[    26,  15000],
[    29,  28000],
[    29,  83000],
[    35,  44000],
[    35,  25000],
[    28, 123000],
[    35,  73000],
[    28,  37000],
[    27,  88000],
[    28,  59000],
[    32,  86000],
[    33, 149000],
[    19,  21000],
[    21,  72000],
[    26,  35000],
[    27,  89000],
[    26,  86000],
[    38,  80000],
[    39,  71000],
[    37,  71000],
[    38,  61000],
[    37,  55000],
[    42,  80000],
[    40,  57000],
[    35,  75000],
[    36,  52000],
[    40,  59000],
[    41,  59000],
```

```
[    36,  75000],
[    37,  72000],
[    40,  75000],
[    35,  53000],
[    41,  51000],
[    39,  61000],
[    42,  65000],
[    26,  32000],
[    30,  17000],
[    26,  84000],
[    31,  58000],
[    33,  31000],
[    30,  87000],
[    21,  68000],
[    28,  55000],
[    23,  63000],
[    20,  82000],
[    30, 107000],
[    28,  59000],
[    19,  25000],
[    19,  85000],
[    18,  68000],
[    35,  59000],
[    30,  89000],
[    34,  25000],
[    24,  89000],
[    27,  96000],
[    41,  30000],
[    29,  61000],
[    20,  74000],
[    26,  15000],
[    41,  45000],
[    31,  76000],
[    36,  50000],
[    40,  47000],
[    31,  15000],
[    46,  59000],
[    29,  75000],
[    26,  30000],
[    32, 135000],
[    32, 100000],
[    25,  90000],
[    37,  33000],
[    35,  38000],
[    33,  69000],
[    18,  86000],
[    22,  55000],
[    35,  71000],
[    29, 148000],
[    29,  47000],
[    21,  88000],
[    34, 115000],
[    26, 118000],
[    34,  43000],
[    34,  72000],
[    23,  28000],
[    35,  47000],
[    25,  22000],
[    24,  23000],
[    31,  34000],
```

```
[    26,  16000],
[    31,  71000],
[    32, 117000],
[    33,  43000],
[    33,  60000],
[    31,  66000],
[    20,  82000],
[    33,  41000],
[    35,  72000],
[    28,  32000],
[    24,  84000],
[    19,  26000],
[    29,  43000],
[    19,  70000],
[    28,  89000],
[    34,  43000],
[    30,  79000],
[    20,  36000],
[    26,  80000],
[    35,  22000],
[    35,  39000],
[    49,  74000],
[    39, 134000],
[    41,  71000],
[    58, 101000],
[    47,  47000],
[    55, 130000],
[    52, 114000],
[    40, 142000],
[    46,  22000],
[    48,  96000],
[    52, 150000],
[    59,  42000],
[    35,  58000],
[    47,  43000],
[    60, 108000],
[    49,  65000],
[    40,  78000],
[    46,  96000],
[    59, 143000],
[    41,  80000],
[    35,  91000],
[    37, 144000],
[    60, 102000],
[    35,  60000],
[    37,  53000],
[    36, 126000],
[    56, 133000],
[    40,  72000],
[    42,  80000],
[    35, 147000],
[    39,  42000],
[    40, 107000],
[    49,  86000],
[    38, 112000],
[    46,  79000],
[    40,  57000],
[    37,  80000],
[    46,  82000],
[    53, 143000],
```

```
[    42, 149000],
[    38,  59000],
[    50,  88000],
[    56, 104000],
[    41,  72000],
[    51, 146000],
[    35,  50000],
[    57, 122000],
[    41,  52000],
[    35,  97000],
[    44,  39000],
[    37,  52000],
[    48, 134000],
[    37, 146000],
[    50,  44000],
[    52,  90000],
[    41,  72000],
[    40,  57000],
[    58,  95000],
[    45, 131000],
[    35,  77000],
[    36, 144000],
[    55, 125000],
[    35,  72000],
[    48,  90000],
[    42, 108000],
[    40,  75000],
[    37,  74000],
[    47, 144000],
[    40,  61000],
[    43, 133000],
[    59,  76000],
[    60,  42000],
[    39, 106000],
[    57,  26000],
[    57,  74000],
[    38,  71000],
[    49,  88000],
[    52,  38000],
[    50,  36000],
[    59,  88000],
[    35,  61000],
[    37,  70000],
[    52,  21000],
[    48, 141000],
[    37,  93000],
[    37,  62000],
[    48, 138000],
[    41,  79000],
[    37,  78000],
[    39, 134000],
[    49,  89000],
[    55,  39000],
[    37,  77000],
[    35,  57000],
[    36,  63000],
[    42,  73000],
[    43, 112000],
[    45,  79000],
[    46, 117000],
```

```
[    58,  38000],
[    48,  74000],
[    37, 137000],
[    37,  79000],
[    40,  60000],
[    42,  54000],
[    51, 134000],
[    47, 113000],
[    36, 125000],
[    38,  50000],
[    42,  70000],
[    39,  96000],
[    38,  50000],
[    49, 141000],
[    39,  79000],
[    39,  75000],
[    54, 104000],
[    35,  55000],
[    45,  32000],
[    36,  60000],
[    52, 138000],
[    53,  82000],
[    41,  52000],
[    48,  30000],
[    48, 131000],
[    41,  60000],
[    41,  72000],
[    42,  75000],
[    36, 118000],
[    47, 107000],
[    38,  51000],
[    48, 119000],
[    42,  65000],
[    40,  65000],
[    57,  60000],
[    36,  54000],
[    58, 144000],
[    35,  79000],
[    38,  55000],
[    39, 122000],
[    53, 104000],
[    35,  75000],
[    38,  65000],
[    47,  51000],
[    47, 105000],
[    41,  63000],
[    53,  72000],
[    54, 108000],
[    39,  77000],
[    38,  61000],
[    38, 113000],
[    37,  75000],
[    42,  90000],
[    37,  57000],
[    36,  99000],
[    60,  34000],
[    54,  70000],
[    41,  72000],
[    40,  71000],
[    42,  54000],
```

```
[    43, 129000],
[    53,  34000],
[    47,  50000],
[    42,  79000],
[    42, 104000],
[    59,  29000],
[    58,  47000],
[    46,  88000],
[    38,  71000],
[    54,  26000],
[    60,  46000],
[    60,  83000],
[    39,  73000],
[    59, 130000],
[    37,  80000],
[    46,  32000],
[    46,  74000],
[    42,  53000],
[    41,  87000],
[    58,  23000],
[    42,  64000],
[    48,  33000],
[    44, 139000],
[    49,  28000],
[    57,  33000],
[    56,  60000],
[    49,  39000],
[    39,  71000],
[    47,  34000],
[    48,  35000],
[    48,  33000],
[    47,  23000],
[    45,  45000],
[    60,  42000],
[    39,  59000],
[    46,  41000],
[    51,  23000],
[    50,  20000],
[    36,  33000],
[    49,  36000]], dtype=int64)
```

In [77]:
```
labels=data.iloc[:,4].values#output values
labels
```

Out[77]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1], dtype=int64)
```

In [78]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [97]:
```
#to find the which random state will give maximum accuracy
acc=[]
tr_acc=[]
iteration=[]
for i in range(len(data)):
    x_train,x_test,y_train,y_test=train_test_split(features,labels,test_size=0.2
    model=LogisticRegression()
    model.fit(x_train,y_train)
    train_acc=model.score(x_train,y_train)
    test_acc=model.score(x_test,y_test)
    if test_acc>train_acc:
        acc.append(test_acc)
        tr_acc.append(train_acc)
        iteration.append(i)
       # print(test_acc,' ',train_acc,' ',i)
ran_state=iteration[acc.index(max(acc)) ]
print(ran_state)
print(max(acc))
print(max(tr_acc))
```

```
314
0.9125
0.840625
```

In [98]:
```
x_train,x_test,y_train,y_test=train_test_split(features,labels,test_size=0.2,ran
model1=LogisticRegression()
model1.fit(x_train,y_train)
```

Out[98]:
```
▾ LogisticRegression

LogisticRegression()
```

In [99]:
```
model1.score(x_test,y_test)
```

Out[99]:
```
0.9125
```

In [ ]:

230701036

ARUN MC

II CSE A

Z test

In [2]:
```
import numpy as np
import scipy.stats as stats
```

In [3]:
```
sample_data = np.array([152, 148, 151, 149, 147, 153, 150, 148, 152,
149,151, 150, 149, 152, 151, 148, 150, 152, 149, 150,148, 153, 151,
150, 149, 152, 148, 151, 150, 153])
```

In [4]:
```
# Population mean under the null hypothesis
population_mean = 150
# Calculate sample statistics
sample_mean = np.mean(sample_data)
sample_std = np.std(sample_data, ddof=1)
# Using sample standarddeviation
```

In [5]:
```
# Number of observations
n = len(sample_data)
```

In [6]:
```
# Calculate the Z-statistic
z_statistic = (sample_mean - population_mean) / (sample_std /np.sqrt(n))

# Calculate the p-value
p_value = 2 * (1 - stats.norm.cdf(np.abs(z_statistic)))
# Two-tailed test


# Print results
print(f"Sample Mean: {sample_mean:.2f}")
print(f"Z-Statistic: {z_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")
```

```
Sample Mean: 150.20
Z-Statistic: 0.6406
P-Value: 0.5218
```

In [7]:
```
# Decision based on the significance level
alpha = 0.05
if p_value <alpha:
    print("Reject the null hypothesis: The average weight is significantly diffe
else:
    print("Fail to reject the null hypothesis: There is no significant differenc
```

```
Fail to reject the null hypothesis: There is no significant difference in average
weight from 150 grams.
```

In [ ]:

230701036

ARUN MC

II CSE A

In [10]:
```
import numpy as np
import scipy.stats as stats
```

In [5]:
```
np.random.seed(42)
sample_size = 25
sample_data = np.random.normal(loc=102, scale=15, size=sample_size)
```

In [6]:
```
population_mean = 100

sample_mean = np.mean(sample_data)
sample_std = np.std(sample_data, ddof=1)
```

In [7]:
```
n = len(sample_data)

t_statistic, p_value = stats.ttest_1samp(sample_data, population_mean)
```

In [8]:
```
print(f'Sample Mean: {sample_mean:.2f}')
print(f'T-Statistic: {t_statistic:.4f}')
print(f'P-Value: {p_value:.4f}')
```

```
Sample Mean: 99.55
T-Statistic: -0.1577
P-Value: 0.8760
```

In [9]:
```
alpha = 0.05
if p_value < alpha:
    print('Reject the null hypothesis: The average IQ score is significantly dif
else:
    print('Fail to reject the null hypothesis: There is no significant differenc
```

```
Fail to reject the null hypothesis: There is no significant difference in average
IQ score from 100.
```

In [ ]:
```

```

230701036

ARUN MC

II CSE A

In [11]:
```
import numpy as np
import scipy.stats as stats
```

In [12]:
```
np.random.seed(42)

n_plants = 25


growth_A = np.random.normal(loc=10, scale=2, size=n_plants)
growth_B = np.random.normal(loc=12, scale=3, size=n_plants)
growth_C = np.random.normal(loc=15, scale=2.5, size=n_plants)
```

In [13]:
```
all_data = np.concatenate([growth_A, growth_B, growth_C])
treatment_labels = ['A'] * n_plants + ['B'] * n_plants + ['C'] * n_plants
f_statistic, p_value = stats.f_oneway(growth_A, growth_B, growth_C)
```

In [14]:
```
print('Treatment A Mean Growth:', np.mean(growth_A))
print('Treatment B Mean Growth:', np.mean(growth_B))
print('Treatment C Mean Growth:', np.mean(growth_C))
```

```
Treatment A Mean Growth: 9.672983882683818
Treatment B Mean Growth: 11.137680744437432
Treatment C Mean Growth: 15.265234904828972
```

In [15]:
```
print(f'F-Statistic: {f_statistic:.4f}')
print(f'P-Value: {p_value:.4f}')
```

```
F-Statistic: 36.1214
P-Value: 0.0000
```

In [16]:
```
alpha = 0.05
if p_value < alpha:
    print('Reject the null hypothesis: There is a significant difference in mean
else:
    print('Fail to reject the null hypothesis: There is no significant differenc
```

```
Reject the null hypothesis: There is a significant difference in mean growth rate
s among the three treatments.
```

In [17]:
```
if p_value < alpha:
    from statsmodels.stats.multicomp import pairwise_tukeyhsd
    tukey_results = pairwise_tukeyhsd(all_data, treatment_labels,alpha=0.05)
    print("\nTukey's HSD Post-hoc Test:")
    print(tukey_results)
```

```
Tukey's HSD Post-hoc Test:
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
group1 group2 meandiff p-adj   lower  upper  reject
-------------------------------------------------
     A      B   1.4647 0.0877 -0.1683 3.0977  False
     A      C   5.5923  0.001  3.9593 7.2252   True
     B      C   4.1276  0.001  2.4946 5.7605   True
-------------------------------------------------
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: