In [ ]:
```
# Name  : Aravinthaa S
# Year : II
# Branch : B.E.Computer Science and Engineering
# Section : A
# Register No : 230701032
# Semester : III
# Subject Code : CS23334
# Subject Name : Fundamentals of Data Science
```

In [4]:
```
# 1.a) Basic Practice Experiments
# 230701032
# Aravinthaa S
# Date : 30.07.2024

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

data=pd.read_csv('Iris_Dataset.csv')
data
```

Out[4]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	variety
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [2]:
```
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   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   variety        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

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

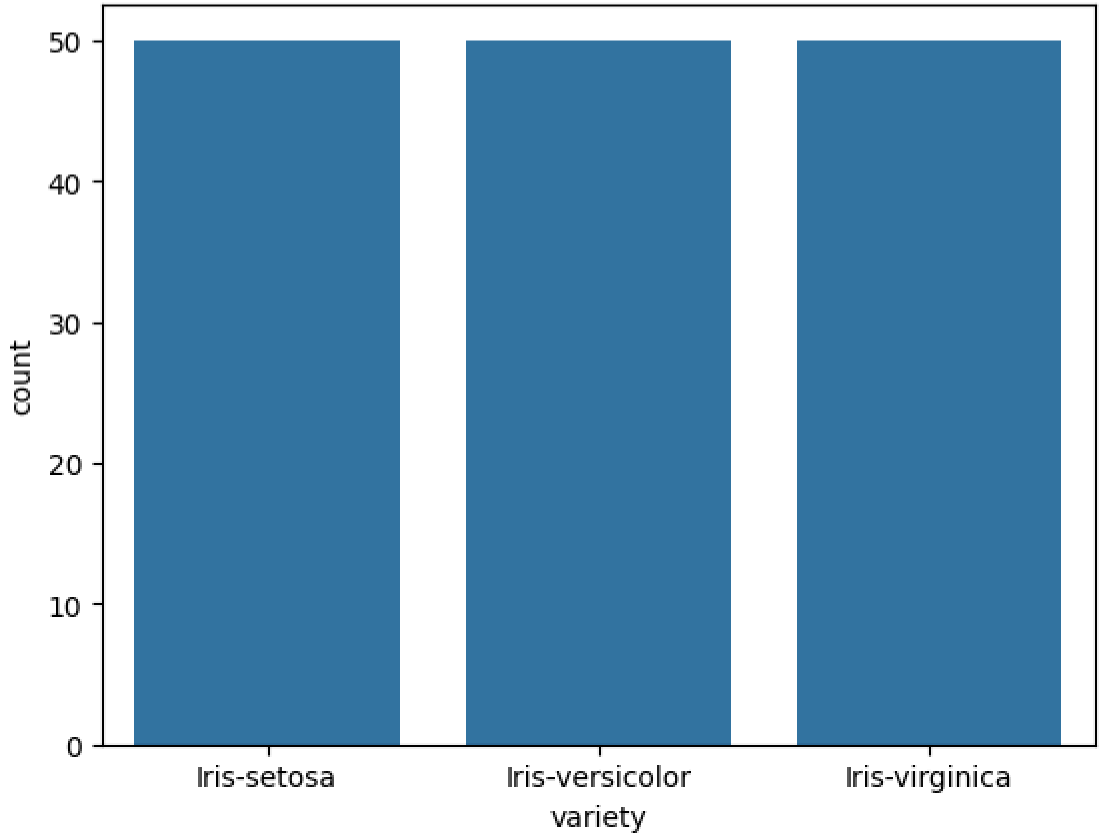
Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [6]:
```
data.value_counts('variety')
```

Out[6]:
```
variety
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
```

In [7]:
```
sns.countplot(x='variety',data=data,)
plt.show()
```



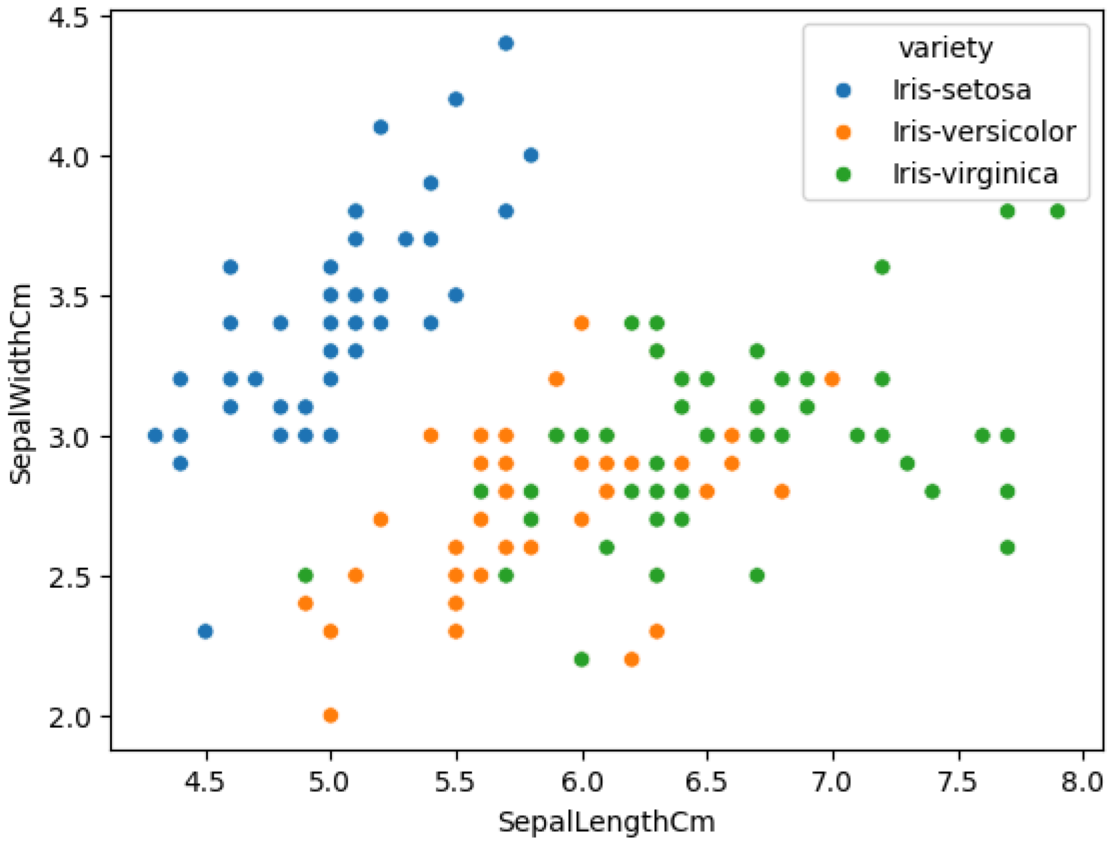
In [9]:
```
dummies=pd.get_dummies(data.variety)
FinalDataset=pd.concat([pd.get_dummies(data.variety),data.iloc[:,[0,1,2,3]]],axis=1)
FinalDataset.head()
```

Out[9]:

	Iris-setosa	Iris-versicolor	Iris-virginica	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm
0	True	False	False	1	5.1	3.5	1.4
1	True	False	False	2	4.9	3.0	1.4
2	True	False	False	3	4.7	3.2	1.3
3	True	False	False	4	4.6	3.1	1.5
4	True	False	False	5	5.0	3.6	1.4

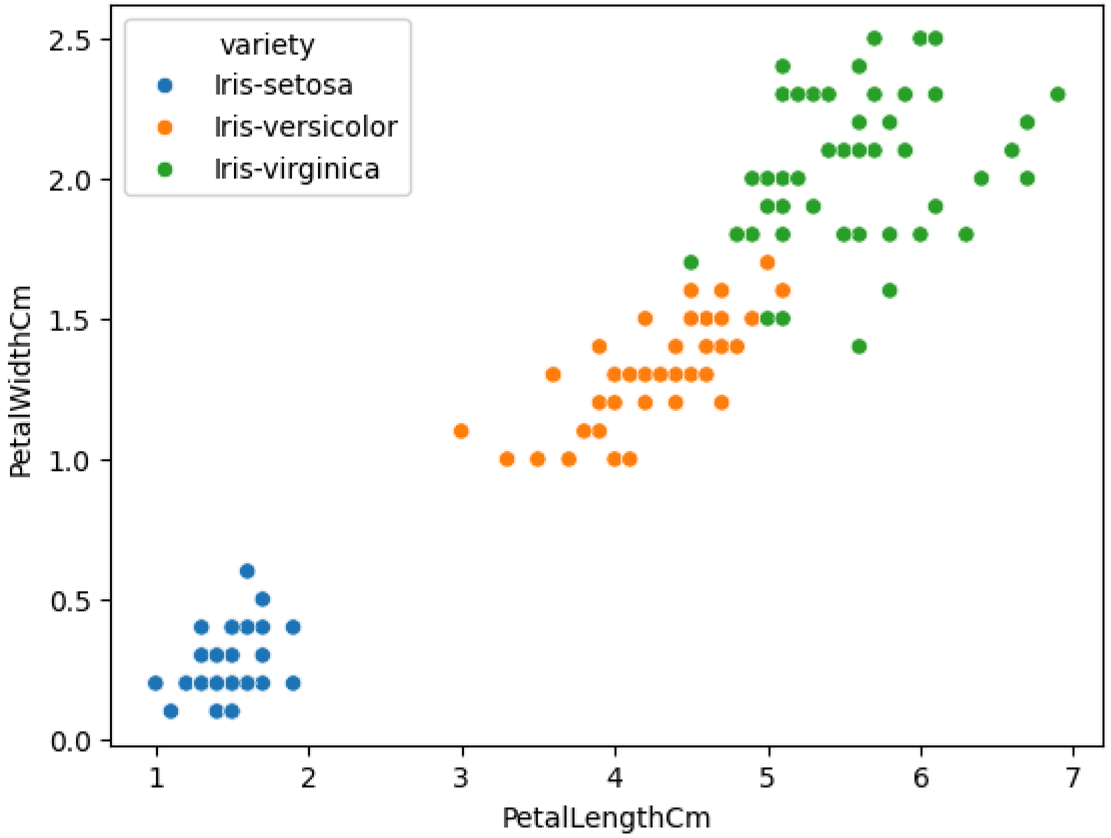
In [10]:
```
sns.scatterplot(x='SepalLengthCm',y='SepalWidthCm',hue='variety',data=data,)
```

Out[10]:
```
<Axes: xlabel='SepalLengthCm', ylabel='SepalWidthCm'>
```

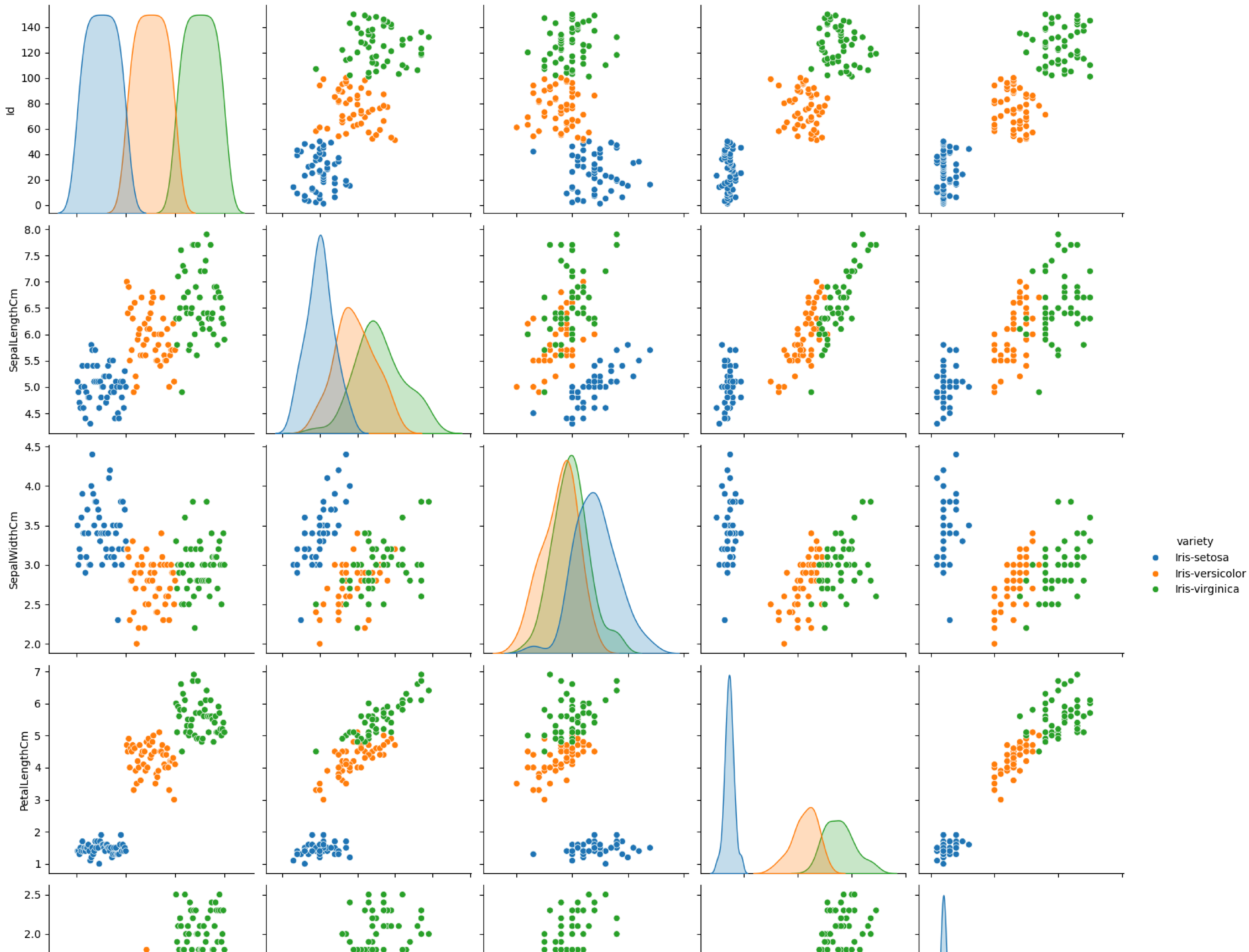


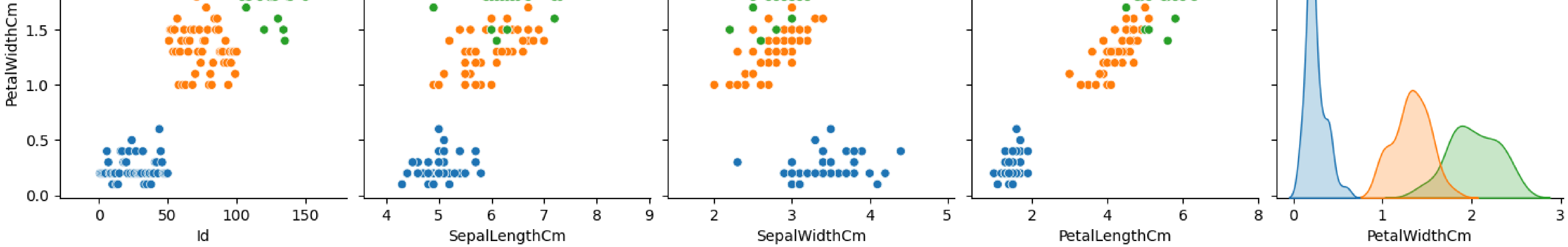
In [11]:
```
sns.scatterplot(x='PetalLengthCm',y='PetalWidthCm',hue='variety',data=data,)
```

Out[11]:
```
<Axes: xlabel='PetalLengthCm', ylabel='PetalWidthCm'>
```

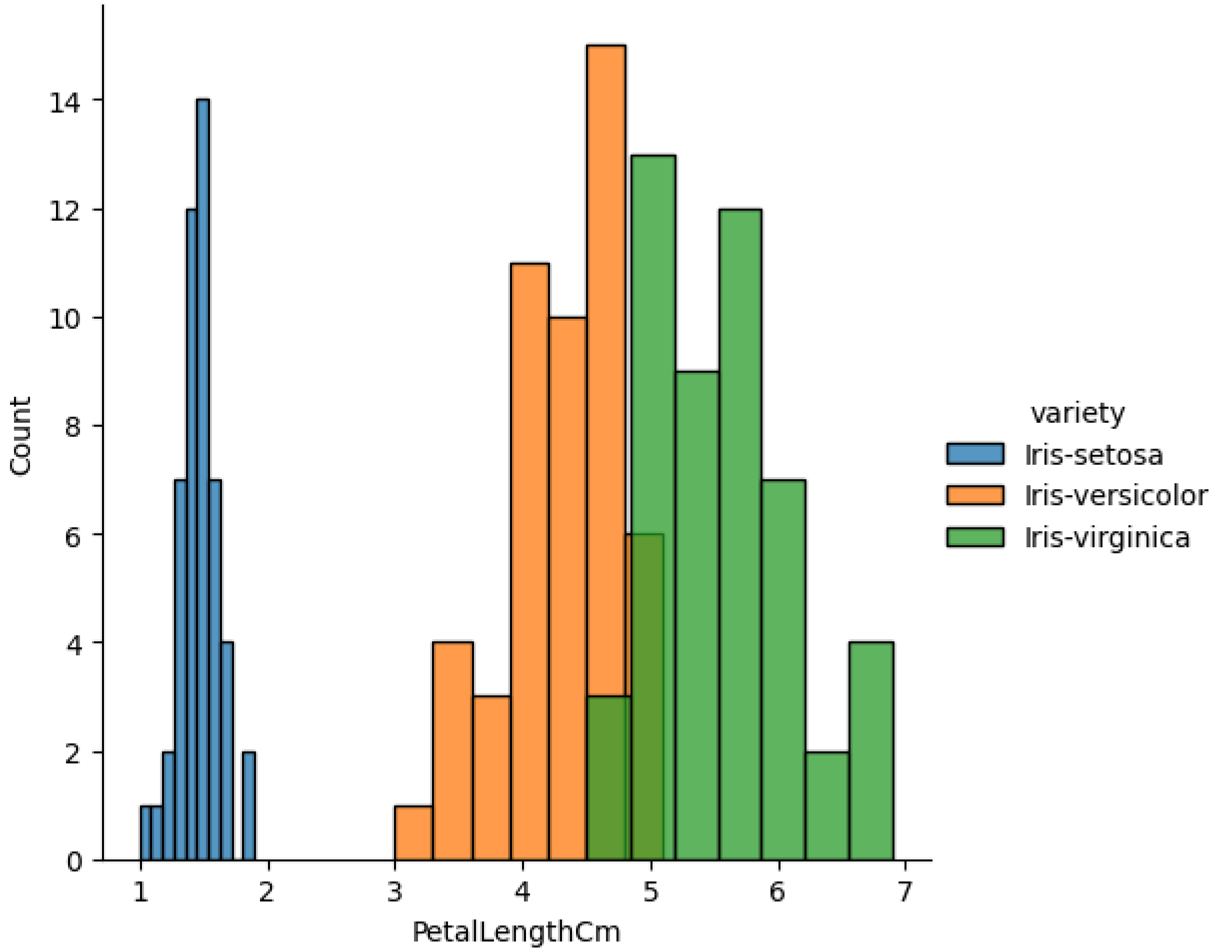


In [12]:
```
sns.pairplot(data,hue='variety',height=3);
```

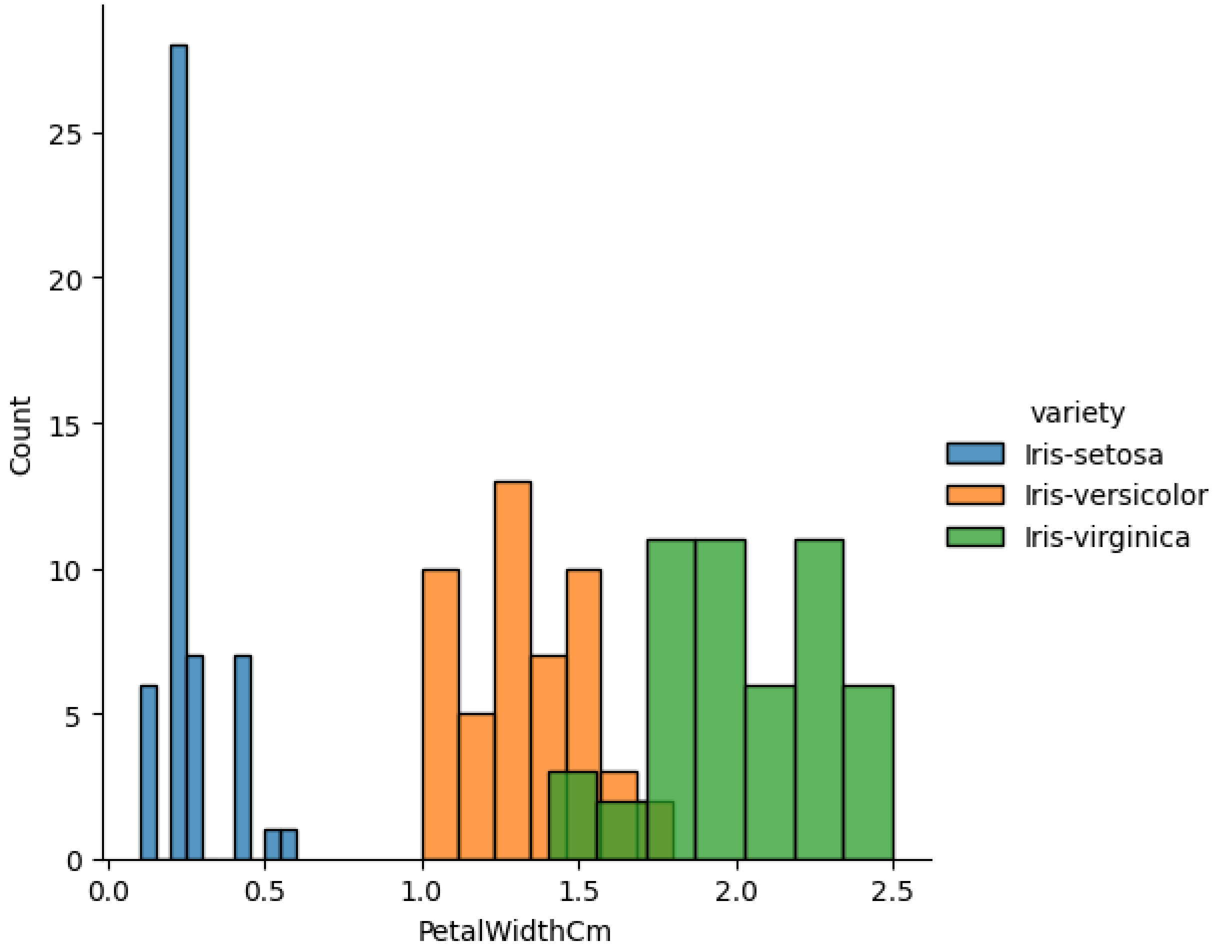

Id
SepalLengthCm
SepalWidthCm
PetalLengthCm
variety
Iris-setosa
Iris-versicolor
Iris-virginica



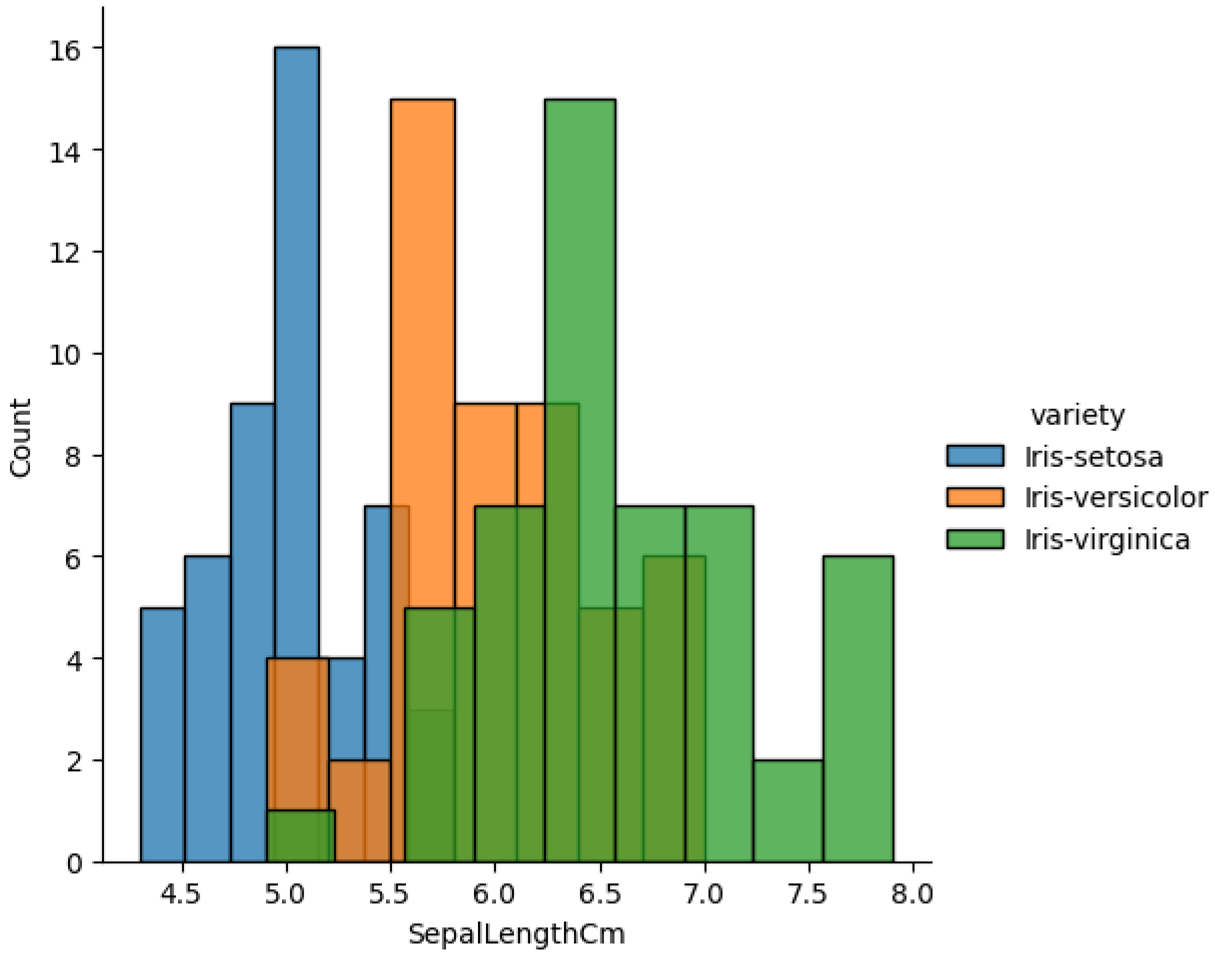
In [13]:
```
plt.show()
sns.FacetGrid(data,hue='variety',height=5).map(sns.histplot,'PetalLengthCm').add_legend();
plt.show();
```



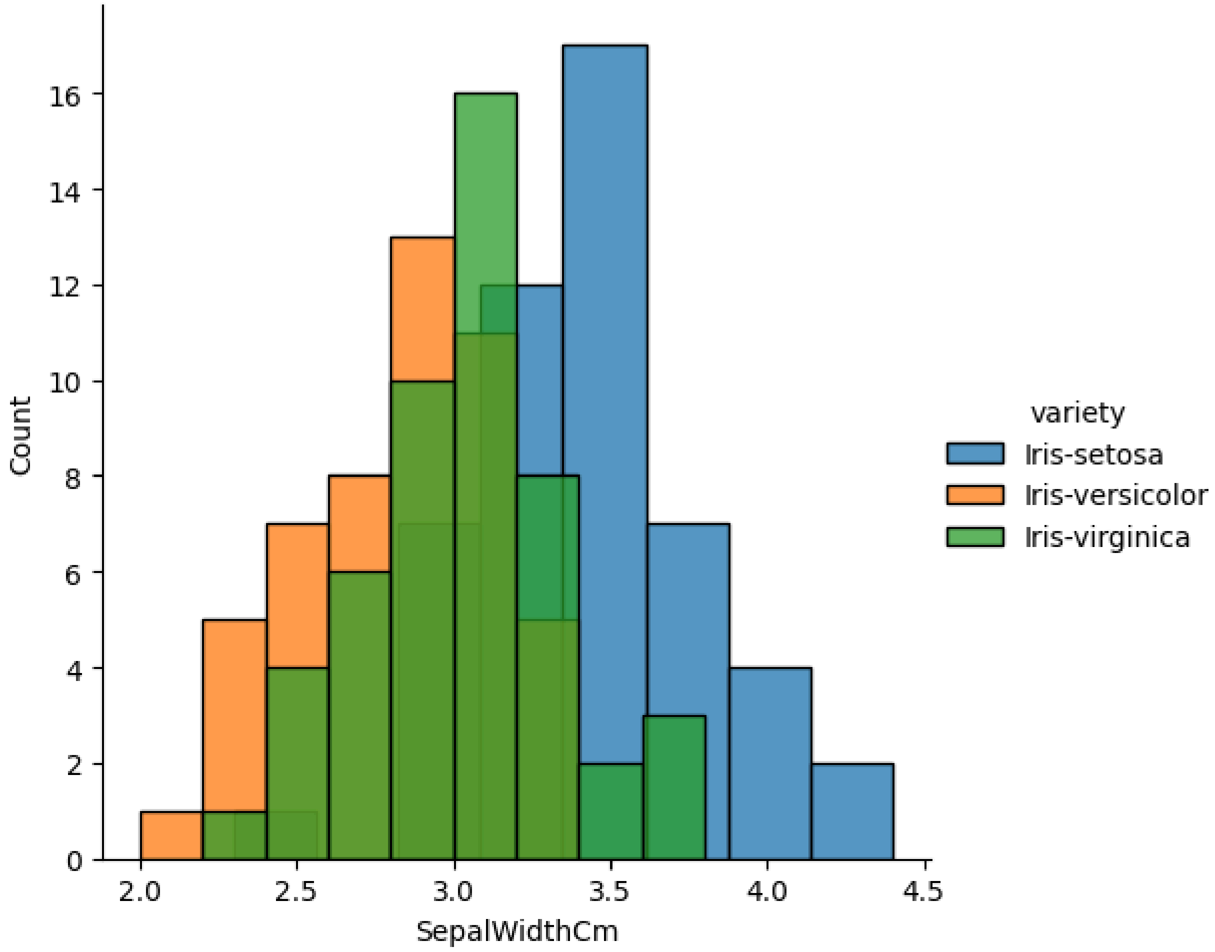
In [14]:
```
sns.FacetGrid(data,hue='variety',height=5).map(sns.histplot,'PetalWidthCm').add_legend();
plt.show();
```



In [15]:
```
sns.FacetGrid(data,hue='variety',height=5).map(sns.histplot,'SepalLengthCm').add_legend();
plt.show();
```



In [16]:
```
sns.FacetGrid(data,hue='variety',height=5).map(sns.histplot,'SepalWidthCm').add_legend();
plt.show();
```



In [ ]:

In [ ]:
```
# 1.b) Pandas Buit in function; Numpy Buit in fuction- Array slicing, Ravel,Reshape,ndim
# 230701032
# Aravinthaa S
# Date : 06.08.2024
```

In [64]:
```
# Pandas Buit in function
import numpy as np
import pandas as pd
list=[[1,'Smith',50000],[2,'Jones',60000]]

df=pd.DataFrame(list)
df
```

Out[64]:

	0	1	2
0	1	Smith	50000
1	2	Jones	60000

In [65]:
```
df.columns=['Empd','Name','Salary']
df
```

Out[65]:

	Empd	Name	Salary
0	1	Smith	50000
1	2	Jones	60000

In [66]:
```
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 [67]:
```
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 [68]:
```
df.head()
```

Out[68]:

	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 [69]:
```
df.tail()
```

Out[69]:

	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 [71]:
```
import numpy as np
import pandas as pd
df=pd.read_csv("employee.csv")

df.head()
```

Out[71]:

	emp id	name	salary
0	1	John Doe	28000
1	2	Jane Smith	24000
2	3	Michael Johnson	12000
3	4	Emily Davis	8000
4	5	Christopher Brown	25000

In [72]:

```
df.head()
```

Out[72]:

	emp id	name	salary
0	1	John Doe	28000
1	2	Jane Smith	24000
2	3	Michael Johnson	12000
3	4	Emily Davis	8000
4	5	Christopher Brown	25000

In [73]:

```
df.tail()
```

Out[73]:

	emp id	name	salary
25	26	Madison Campbell	15000
26	27	Lucas Nelson	31000
27	28	Ella Carter	26000
28	29	Mason Mitchell	13000
29	30	Grace Perez	9000

In [74]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   emp id  30 non-null     int64 
 1   name    30 non-null     object
 2   salary  30 non-null     int64 
dtypes: int64(2), object(1)
memory usage: 848.0+ bytes
```

In [75]:

```
df.salary
```

```
Out[75]: 0     28000
1     24000
2     12000
3      8000
4     25000
5     15000
6      9000
7     29000
8     13000
9      8500
10    30000
11    27000
12     7500
13    26000
14    23000
15    11000
16    28000
17    25000
18     8000
19    14000
20    24000
21     9500
22    16000
23    27000
24     7000
25    15000
26    31000
27    26000
28    13000
29     9000
Name: salary, dtype: int64
```

```
In [76]: type(df.salary)
```

```
Out[76]: pandas.core.series.Series
```

```
In [77]: df.salary.mean()
```

```
Out[77]: 18283.333333333332
```

```
In [78]: df.salary.median()
```

```
Out[78]: 15500.0
```

```
In [79]: df.salary.mode()
```

```
Out[79]: 0     8000
1     9000
2    13000
3    15000
4    24000
5    25000
6    26000
7    27000
8    28000
Name: salary, dtype: int64
```

```
In [80]: df.salary.var()
```

```
Out[80]: 70529022.98850574
```

```
In [81]: df.salary.std()
```

```
Out[81]: 8398.155927851407
```

In [82]:
```
df.describe()
```

Out[82]:

	emp id	salary
count	30.000000	30.000000
mean	15.500000	18283.333333
std	8.803408	8398.155928
min	1.000000	7000.000000
25%	8.250000	9875.000000
50%	15.500000	15500.000000
75%	22.750000	26000.000000
max	30.000000	31000.000000

In [83]:
```
df.describe(include='all')
```

Out[83]:

	emp id	name	salary
count	30.000000	30	30.000000
unique	NaN	30	NaN
top	NaN	John Doe	NaN
freq	NaN	1	NaN
mean	15.500000	NaN	18283.333333
std	8.803408	NaN	8398.155928
min	1.000000	NaN	7000.000000
25%	8.250000	NaN	9875.000000
50%	15.500000	NaN	15500.000000
75%	22.750000	NaN	26000.000000
max	30.000000	NaN	31000.000000

In [84]:
```
empCol=df.columns

empCol
```

Out[84]:
```
Index(['emp id', 'name', 'salary'], dtype='object')
```

In [87]:
```
emparray=df.values
emparray
```

Out[87]:
```
array([[1, 'John Doe', 28000],
       [2, 'Jane Smith', 24000],
       [3, 'Michael Johnson', 12000],
       [4, 'Emily Davis', 8000],
       [5, 'Christopher Brown', 25000],
       [6, 'Jessica Wilson', 15000],
       [7, 'Daniel Garcia', 9000],
       [8, 'Sophia Martinez', 29000],
       [9, 'David Anderson', 13000],
       [10, 'Olivia Lee', 8500],
       [11, 'James Thomas', 30000],
       [12, 'Isabella Taylor', 27000],
       [13, 'Matthew Harris', 7500],
       [14, 'Emma Clark', 26000],
       [15, 'Joshua Lewis', 23000],
       [16, 'Ava Walker', 11000],
       [17, 'Andrew Robinson', 28000],
       [18, 'Mia Hall', 25000],
       [19, 'Ethan Young', 8000],
       [20, 'Amelia King', 14000],
       [21, 'Alexander Wright', 24000],
       [22, 'Charlotte Scott', 9500],
       [23, 'William Green', 16000],
       [24, 'Abigail Adams', 27000],
       [25, 'Benjamin Baker', 7000],
       [26, 'Madison Campbell', 15000],
       [27, 'Lucas Nelson', 31000],
       [28, 'Ella Carter', 26000],
       [29, 'Mason Mitchell', 13000],
       [30, 'Grace Perez', 9000]], dtype=object)
```

In [88]:
```
employee_DF=pd.DataFrame(emparray,columns=empCol)

employee_DF
```

Out[88]:

	emp id	name	salary
0	1	John Doe	28000
1	2	Jane Smith	24000
2	3	Michael Johnson	12000
3	4	Emily Davis	8000
4	5	Christopher Brown	25000
5	6	Jessica Wilson	15000
6	7	Daniel Garcia	9000
7	8	Sophia Martinez	29000
8	9	David Anderson	13000
9	10	Olivia Lee	8500
10	11	James Thomas	30000
11	12	Isabella Taylor	27000
12	13	Matthew Harris	7500
13	14	Emma Clark	26000
14	15	Joshua Lewis	23000
15	16	Ava Walker	11000
16	17	Andrew Robinson	28000
17	18	Mia Hall	25000
18	19	Ethan Young	8000
19	20	Amelia King	14000
20	21	Alexander Wright	24000
21	22	Charlotte Scott	9500
22	23	William Green	16000
23	24	Abigail Adams	27000
24	25	Benjamin Baker	7000
25	26	Madison Campbell	15000
26	27	Lucas Nelson	31000
27	28	Ella Carter	26000
28	29	Mason Mitchell	13000
29	30	Grace Perez	9000

In [ ]:

In [50]:

```
# Numpy Buit in fuction- Array slicing, Ravel,Reshape,ndim
import numpy as np
array=np.random.randint(1,100,9)
array
```

Out[50]:

```
array([84, 40, 35, 49, 33, 81, 48, 98, 19])
```

In [51]:
```
np.sqrt(array)
```
Out[51]:
```
array([9.16515139, 6.32455532, 5.91607978, 7.        , 5.74456265,
       9.        , 6.92820323, 9.89949494, 4.35889894])
```
In [52]:
```
array.ndim
```
Out[52]:
```
1
```
In [53]:
```
new_array=array.reshape(3,3)
new_array
```
Out[53]:
```
array([[84, 40, 35],
       [49, 33, 81],
       [48, 98, 19]])
```
In [54]:
```
new_array.ndim
```
Out[54]:
```
2
```
In [55]:
```
new_array.ravel()
```
Out[55]:
```
array([84, 40, 35, 49, 33, 81, 48, 98, 19])
```
In [56]:
```
newm=new_array.reshape(3,3)
newm
```
Out[56]:
```
array([[84, 40, 35],
       [49, 33, 81],
       [48, 98, 19]])
```
In [57]:
```
newm[2,1:3]
```
Out[57]:
```
array([98, 19])
```
In [58]:
```
newm[1:2,1:3]
```
Out[58]:
```
array([[33, 81]])
```
In [59]:
```
new_array[0:3,0:0]
```
Out[59]:
```
array([], shape=(3, 0), dtype=int32)
```
In [60]:
```
new_array[0:2,0:1]
```
Out[60]:
```
array([[84],
       [49]])
```
In [61]:
```
new_array[0:3,0:1]
```
Out[61]:
```
array([[84],
       [49],
       [48]])
```
In [62]:
```
new_array[1:3]
```
Out[62]:
```
array([[49, 33, 81],
       [48, 98, 19]])
```
In [ ]:
```

```
In [31]:
```
# 2) Outlier detection
# 230701032
```

```
# Aravinthaa S
# Date : 13.08.2024

#sample calculation for low range(lr) , upper range (ur),percentile
import numpy as np
array=np.random.randint(1,100,16) # randomly generate 16 numbers between 1 to 100
array
```

Out[31]:
```
array([70, 97, 21, 33, 42,  9, 73,  4, 78,  7, 65, 28, 29, 99, 61, 68])
```

In [32]:
```
array.mean()
```

Out[32]:
```
49.0
```

In [33]:
```
np.percentile(array,25)
```

Out[33]:
```
26.25
```

In [34]:
```
np.percentile(array,50)
```

Out[34]:
```
51.5
```

In [35]:
```
np.percentile(array,75)
```

Out[35]:
```
70.75
```

In [36]:
```
np.percentile(array,100)
```

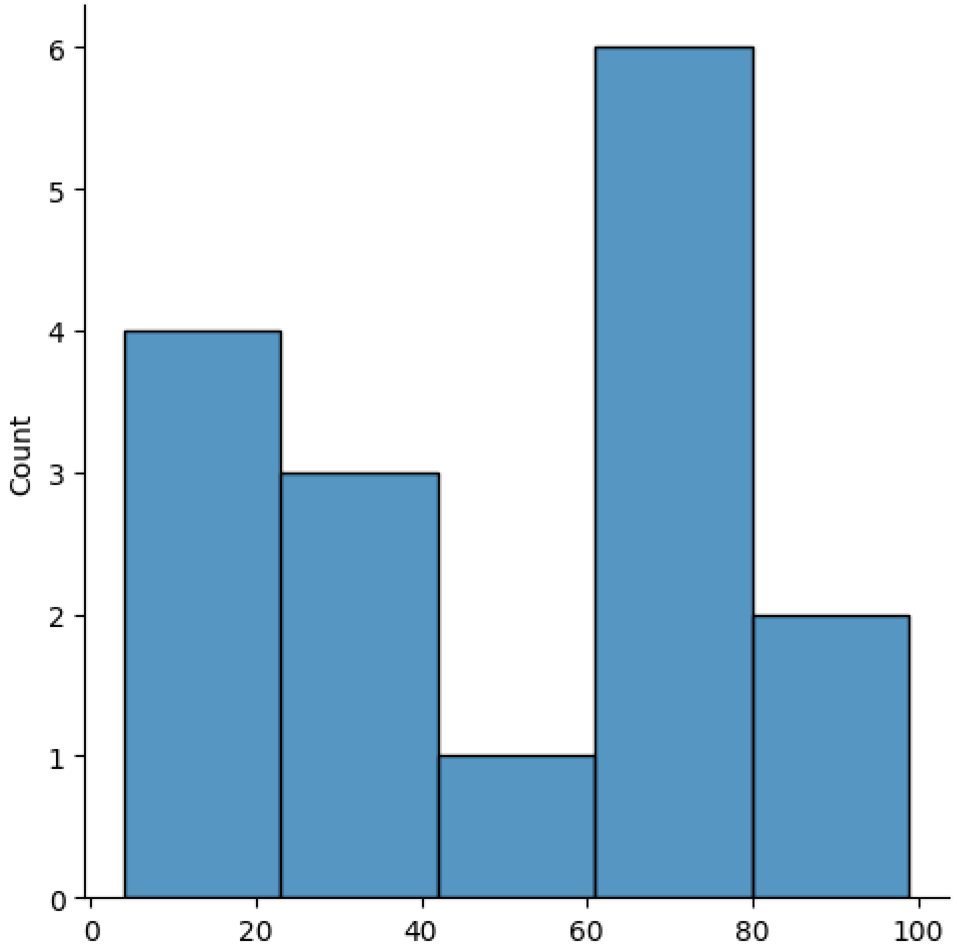
Out[36]:
```
99.0
```

In [37]:
```
#outliers detection
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[37]:
```
(-40.5, 137.5)
```

In [38]:
```
import seaborn as sns
%matplotlib inline
sns.displot(array)
```

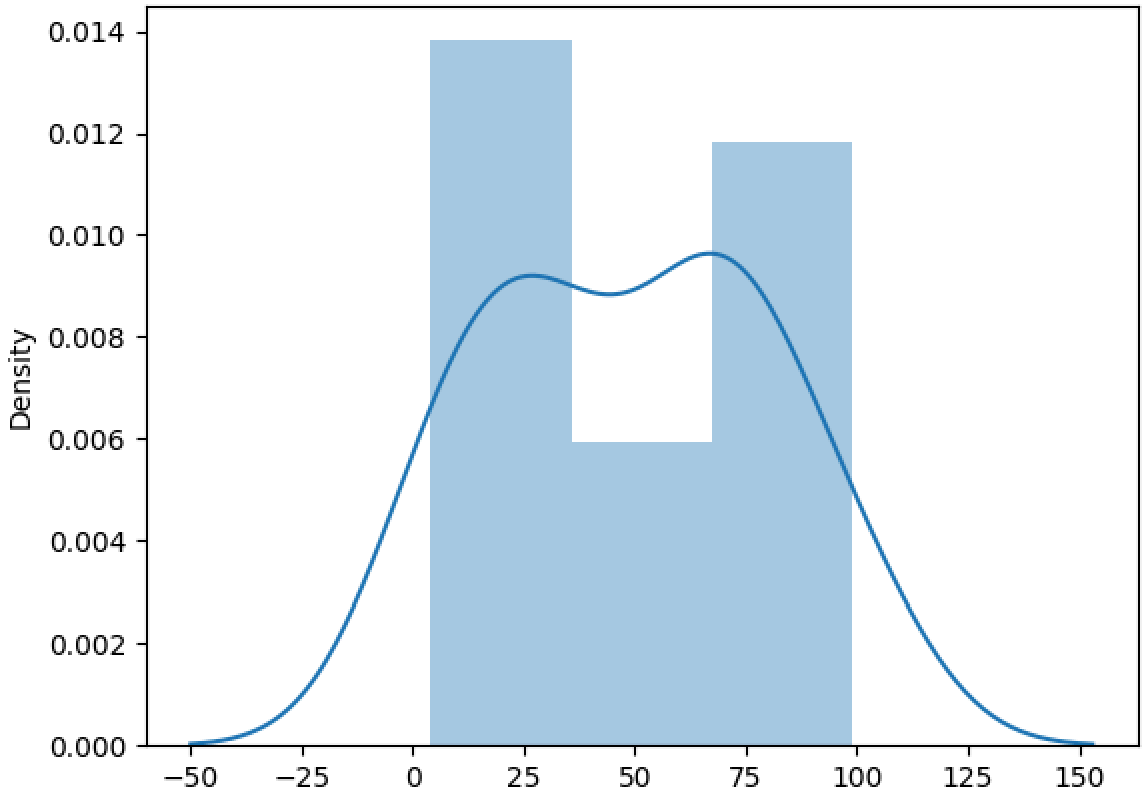
Out[38]:
```
<seaborn.axisgrid.FacetGrid at 0x1bb5fca3f40>
```



In [39]:
```
sns.distplot(array)
```

```
C:\Users\91950\AppData\Local\Temp\ipykernel_16168\1133588802.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(array)
```

Out[39]:
```
<Axes: ylabel='Density'>
```

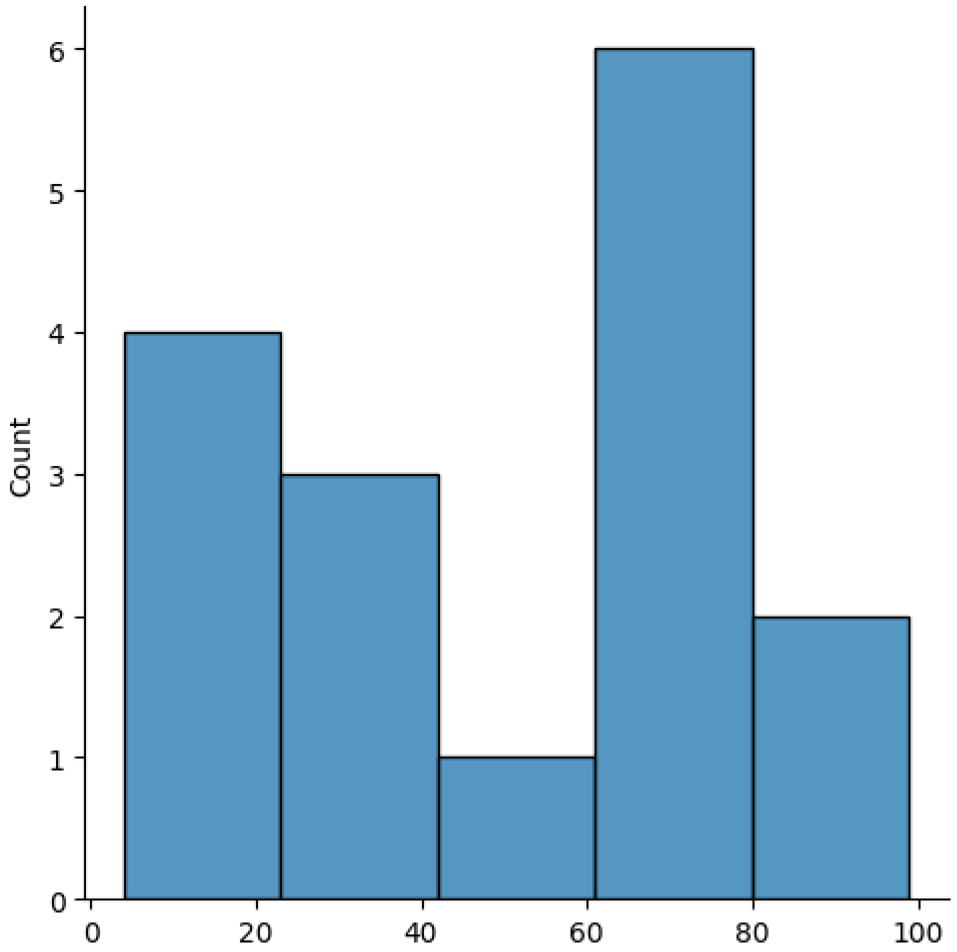


In [40]:
```
new_array=array[(array>lr) & (array<ur)]
new_array
```

Out[40]:
```
array([70, 97, 21, 33, 42,  9, 73,  4, 78,  7, 65, 28, 29, 99, 61, 68])
```

In [41]:
```
sns.displot(new_array)
```

Out[41]:
```
<seaborn.axisgrid.FacetGrid at 0x1bb5fca1420>
```



In [42]:
```
lr1,ur1=outDetection(new_array)
```

```
lr1,ur1
```

Out[42]: (-40.5, 137.5)

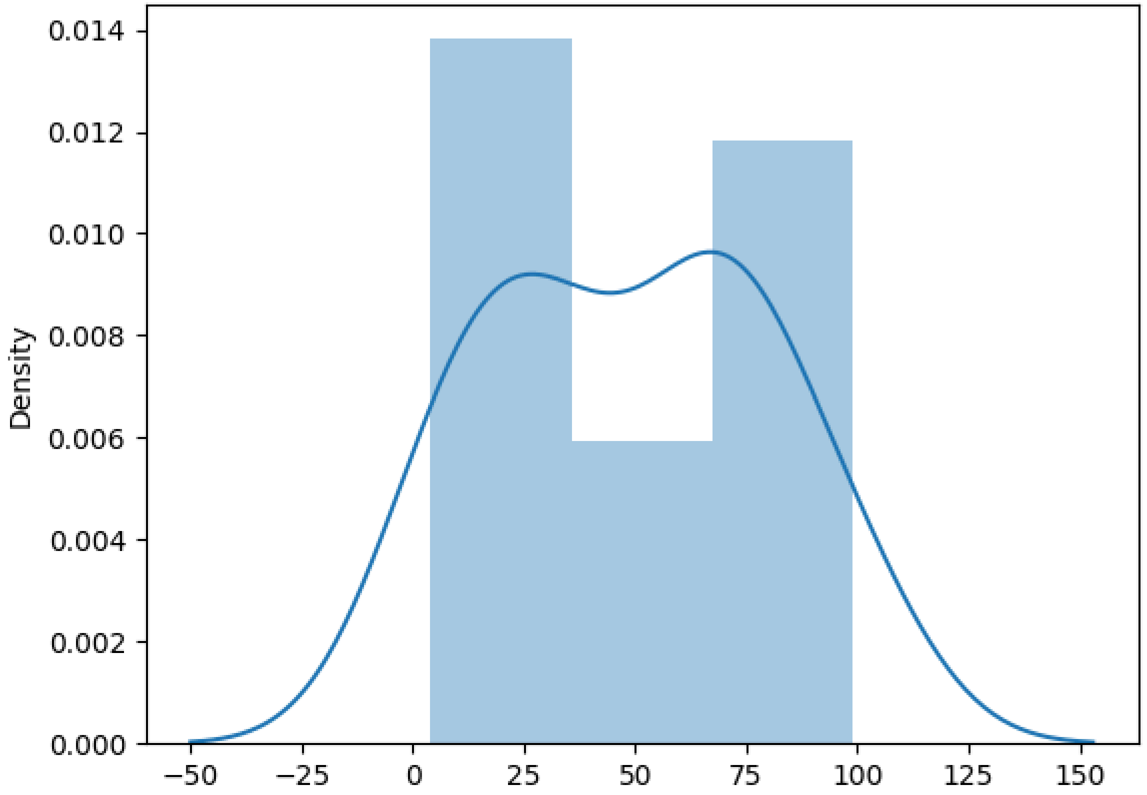
In [43]:
```
final_array=new_array[(new_array>lr1) & (new_array<ur1)]
final_array
```

Out[43]: array([70, 97, 21, 33, 42, 9, 73, 4, 78, 7, 65, 28, 29, 99, 61, 68])

In [44]:
```
sns.distplot(final_array)
```

```
C:\Users\91950\AppData\Local\Temp\ipykernel_16168\209491988.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(final_array)
```

Out[44]: <Axes: ylabel='Density'>



In [ ]:

In [16]:
```
# 3) Missing and inappropriate data
# 230701032
# Aravinthaa S
# Date : 20.08.2024

import numpy as np
import pandas as pd
df=pd.read_csv("Hotel_Dataset.csv")
df
```

Out[16]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary	Age_Group.1
0	1	20-25	4	Ibis	veg	1300	2	40000	20-25
1	2	30-35	5	LemonTree	Non-Veg	2000	3	59000	30-35
2	3	25-30	6	RedFox	Veg	1322	2	30000	25-30
3	4	20-25	-1	LemonTree	Veg	1234	2	120000	20-25
4	5	35+	3	Ibis	Vegetarian	989	2	45000	35+
5	6	35+	3	Ibys	Non-Veg	1909	2	122220	35+
6	7	35+	4	RedFox	Vegetarian	1000	-1	21122	35+
7	8	20-25	7	LemonTree	Veg	2999	-10	345673	20-25
8	9	25-30	2	Ibis	Non-Veg	3456	3	-99999	25-30
9	9	25-30	2	Ibis	Non-Veg	3456	3	-99999	25-30
10	10	30-35	5	RedFox	non-Veg	-6755	4	87777	30-35

In [17]:
```
df.duplicated()
```

Out[17]:
```
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 [18]:
```
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 [19]:
```
df.drop_duplicates(inplace=True)
df
```

Out[19]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary	Age_Group.1
0	1	20-25	4	Ibis	veg	1300	2	40000	20-25
1	2	30-35	5	LemonTree	Non-Veg	2000	3	59000	30-35
2	3	25-30	6	RedFox	Veg	1322	2	30000	25-30
3	4	20-25	-1	LemonTree	Veg	1234	2	120000	20-25
4	5	35+	3	Ibis	Vegetarian	989	2	45000	35+
5	6	35+	3	Ibys	Non-Veg	1909	2	122220	35+
6	7	35+	4	RedFox	Vegetarian	1000	-1	21122	35+
7	8	20-25	7	LemonTree	Veg	2999	-10	345673	20-25
8	9	25-30	2	Ibis	Non-Veg	3456	3	-99999	25-30
10	10	30-35	5	RedFox	non-Veg	-6755	4	87777	30-35

In [20]:

```
len(df)
```

Out[20]: 10

In [21]:

```
index = np.arange(len(df))  # Creates a NumPy array of indices directly
df.set_index(index, inplace=True)  # Sets the DataFrame index
index
df
```

Out[21]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary	Age_Group.1
0	1	20-25	4	Ibis	veg	1300	2	40000	20-25
1	2	30-35	5	LemonTree	Non-Veg	2000	3	59000	30-35
2	3	25-30	6	RedFox	Veg	1322	2	30000	25-30
3	4	20-25	-1	LemonTree	Veg	1234	2	120000	20-25
4	5	35+	3	Ibis	Vegetarian	989	2	45000	35+
5	6	35+	3	Ibys	Non-Veg	1909	2	122220	35+
6	7	35+	4	RedFox	Vegetarian	1000	-1	21122	35+
7	8	20-25	7	LemonTree	Veg	2999	-10	345673	20-25
8	9	25-30	2	Ibis	Non-Veg	3456	3	-99999	25-30
9	10	30-35	5	RedFox	non-Veg	-6755	4	87777	30-35

In [22]:

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

Out[22]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary
0	1	20-25	4	Ibis	veg	1300	2	40000
1	2	30-35	5	LemonTree	Non-Veg	2000	3	59000
2	3	25-30	6	RedFox	Veg	1322	2	30000
3	4	20-25	-1	LemonTree	Veg	1234	2	120000
4	5	35+	3	Ibis	Vegetarian	989	2	45000
5	6	35+	3	Ibys	Non-Veg	1909	2	122220
6	7	35+	4	RedFox	Vegetarian	1000	-1	21122
7	8	20-25	7	LemonTree	Veg	2999	-10	345673
8	9	25-30	2	Ibis	Non-Veg	3456	3	-99999
9	10	30-35	5	RedFox	non-Veg	-6755	4	87777

In [23]:

```
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:\Users\91950\AppData\Local\Temp\ipykernel_16168\2080958306.py:1: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  df.CustomerID.loc[df.CustomerID<0]=np.nan
C:\Users\91950\AppData\Local\Temp\ipykernel_16168\2080958306.py:1: SettingWithCopyWarning:
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/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.CustomerID.loc[df.CustomerID<0]=np.nan
C:\Users\91950\AppData\Local\Temp\ipykernel_16168\2080958306.py:2: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  df.Bill.loc[df.Bill<0]=np.nan
C:\Users\91950\AppData\Local\Temp\ipykernel_16168\2080958306.py:2: SettingWithCopyWarning:
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/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.Bill.loc[df.Bill<0]=np.nan
C:\Users\91950\AppData\Local\Temp\ipykernel_16168\2080958306.py:3: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  df.EstimatedSalary.loc[df.EstimatedSalary<0]=np.nan
C:\Users\91950\AppData\Local\Temp\ipykernel_16168\2080958306.py:3: SettingWithCopyWarning:
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/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.EstimatedSalary.loc[df.EstimatedSalary<0]=np.nan
```

Out[23]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary
0	1.0	20-25	4	Ibis	veg	1300.0	2	40000.0
1	2.0	30-35	5	LemonTree	Non-Veg	2000.0	3	59000.0
2	3.0	25-30	6	RedFox	Veg	1322.0	2	30000.0
3	4.0	20-25	-1	LemonTree	Veg	1234.0	2	120000.0
4	5.0	35+	3	Ibis	Vegetarian	989.0	2	45000.0
5	6.0	35+	3	Ibys	Non-Veg	1909.0	2	122220.0
6	7.0	35+	4	RedFox	Vegetarian	1000.0	-1	21122.0
7	8.0	20-25	7	LemonTree	Veg	2999.0	-10	345673.0
8	9.0	25-30	2	Ibis	Non-Veg	3456.0	3	NaN
9	10.0	30-35	5	RedFox	non-Veg	NaN	4	87777.0

In [24]:

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

```
C:\Users\91950\AppData\Local\Temp\ipykernel_16168\2129877948.py:1: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will become the default behaviour in pandas 3.0) this will never work to update
the original DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  df['NoOfPax'].loc[(df['NoOfPax']<1) | (df['NoOfPax']>20)]=np.nan
C:\Users\91950\AppData\Local\Temp\ipykernel_16168\2129877948.py:1: SettingWithCopyWarning:
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/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['NoOfPax'].loc[(df['NoOfPax']<1) | (df['NoOfPax']>20)]=np.nan
```

Out[24]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary
0	1.0	20-25	4	Ibis	veg	1300.0	2.0	40000.0
1	2.0	30-35	5	LemonTree	Non-Veg	2000.0	3.0	59000.0
2	3.0	25-30	6	RedFox	Veg	1322.0	2.0	30000.0
3	4.0	20-25	-1	LemonTree	Veg	1234.0	2.0	120000.0
4	5.0	35+	3	Ibis	Vegetarian	989.0	2.0	45000.0
5	6.0	35+	3	Ibys	Non-Veg	1909.0	2.0	122220.0
6	7.0	35+	4	RedFox	Vegetarian	1000.0	NaN	21122.0
7	8.0	20-25	7	LemonTree	Veg	2999.0	NaN	345673.0
8	9.0	25-30	2	Ibis	Non-Veg	3456.0	3.0	NaN
9	10.0	30-35	5	RedFox	non-Veg	NaN	4.0	87777.0

In [25]:

```
df.Age_Group.unique()
```

Out[25]:

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

In [26]:
```
df.Hotel.unique()
```

Out[26]:
```
array(['Ibis', 'LemonTree', 'RedFox', 'Ibys'], dtype=object)
```

In [27]:
```
df.Hotel.replace(['Ibys'],'Ibis',inplace=True)
df.FoodPreference.unique
```

```
C:\Users\91950\AppData\Local\Temp\ipykernel_16168\456600217.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df.Hotel.replace(['Ibys'],'Ibis',inplace=True)
```

Out[27]:
```
<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 [122…
```
df.FoodPreference.replace(['Vegetarian','veg'],'Veg',inplace=True)
df.FoodPreference.replace(['non-Veg'],'Non-Veg',inplace=True)
```

In [29]:
```
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
```

```
C:\Users\91950\AppData\Local\Temp\ipykernel_16168\3711388855.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['Rating(1-5)'].fillna(round(df['Rating(1-5)'].median()), inplace=True)
```

Out[29]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary
0	1.0	20-25	4	Ibis	veg	1300.0	2.0	40000.0
1	2.0	30-35	5	LemonTree	Non-Veg	2000.0	3.0	59000.0
2	3.0	25-30	6	RedFox	Veg	1322.0	2.0	30000.0
3	4.0	20-25	-1	LemonTree	Veg	1234.0	2.0	120000.0
4	5.0	35+	3	Ibis	Vegetarian	989.0	2.0	45000.0
5	6.0	35+	3	Ibis	Non-Veg	1909.0	2.0	122220.0
6	7.0	35+	4	RedFox	Vegetarian	1000.0	2.0	21122.0
7	8.0	20-25	7	LemonTree	Veg	2999.0	2.0	345673.0
8	9.0	25-30	2	Ibis	Non-Veg	3456.0	3.0	96755.0
9	10.0	30-35	5	RedFox	non-Veg	1801.0	4.0	87777.0

In [ ]:

```

```

In [1]:

```
# 4) Data Preprocessing
# 230701032
# Aravinthaa S
# Date : 27.08.2024

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	NaN	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    9 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
Name: Country, dtype: object
```

In [4]:

```
df.Country.mode()[0]
```

Out[4]:

```
'France'
```

In [5]:

```
type(df.Country.mode())
```

Out[5]:

```
pandas.core.series.Series
```

In [16]:

```
df.Country.fillna(df.Country.mode()[0],inplace=True)
df.Age.fillna(df.Age.median(),inplace=True)
df.Salary.fillna(round(df.Salary.mean()),inplace=True)
df
```

```
C:\Users\91950\AppData\Local\Temp\ipykernel_13364\1020198583.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df.Age.fillna(df.Age.median(),inplace=True)
C:\Users\91950\AppData\Local\Temp\ipykernel_13364\1020198583.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df.Salary.fillna(round(df.Salary.mean()),inplace=True)
```

Out[16]:

	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	France	50.0	83000.0	No
9	France	37.0	67000.0	Yes

In [7]:

```
pd.get_dummies(df.Country)
```

Out[7]:

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

In [8]:

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

updated_dataset
```

Out[8]:

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

In [9]:
```
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 [11]:
```
updated_dataset.Purchased.replace(['No','Yes'],[0,1],inplace=True)

updated_dataset
```

Out[11]:

	France	Germany	Spain	Age	Salary	Purchased
0	True	False	False	44.0	72000.0	0
1	False	False	True	27.0	48000.0	1
2	False	True	False	30.0	54000.0	0
3	False	False	True	38.0	61000.0	0
4	False	True	False	40.0	63778.0	1
5	True	False	False	35.0	58000.0	1
6	False	False	True	38.0	52000.0	0
7	True	False	False	48.0	79000.0	1
8	True	False	False	50.0	83000.0	0
9	True	False	False	37.0	67000.0	1

In [ ]:

In [18]:
```
# 5) EDA-Quantitative and Qualitative plots
# 230701032
# Aravinthaa S
# Date : 03.09.2024
```

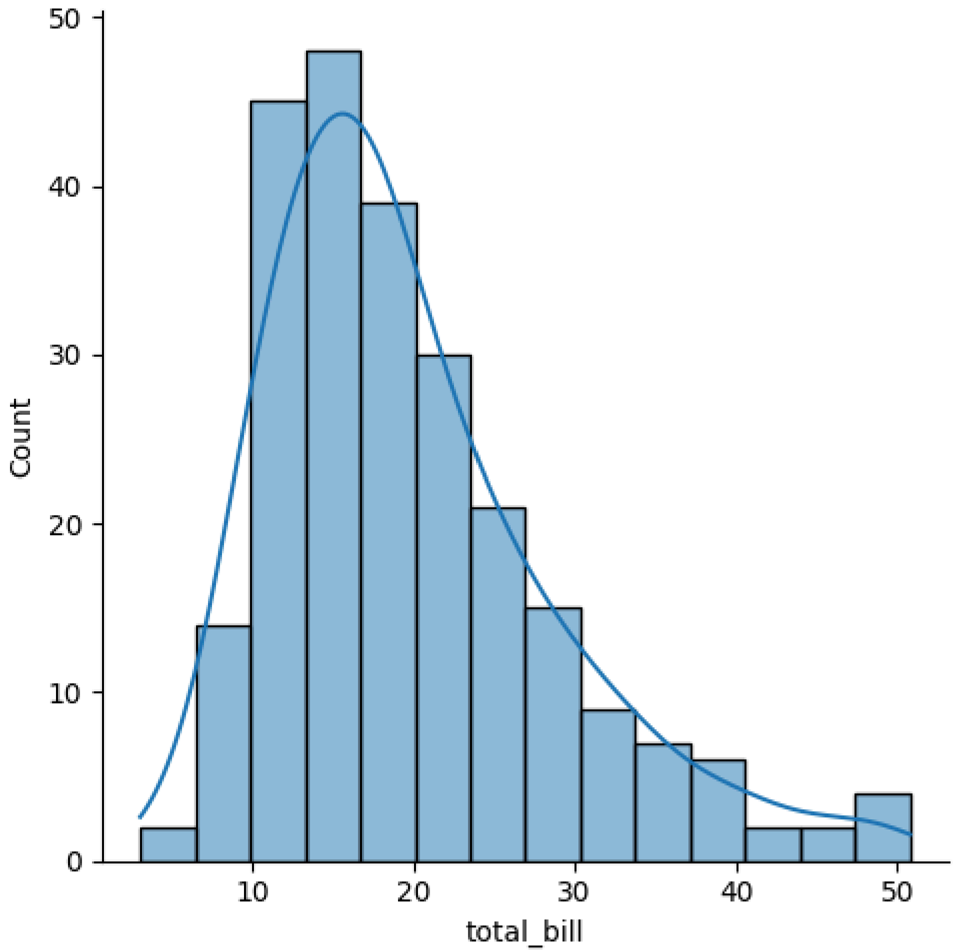
```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
tips=sns.load_dataset('tips')

tips.head()
```

Out[18]:

	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

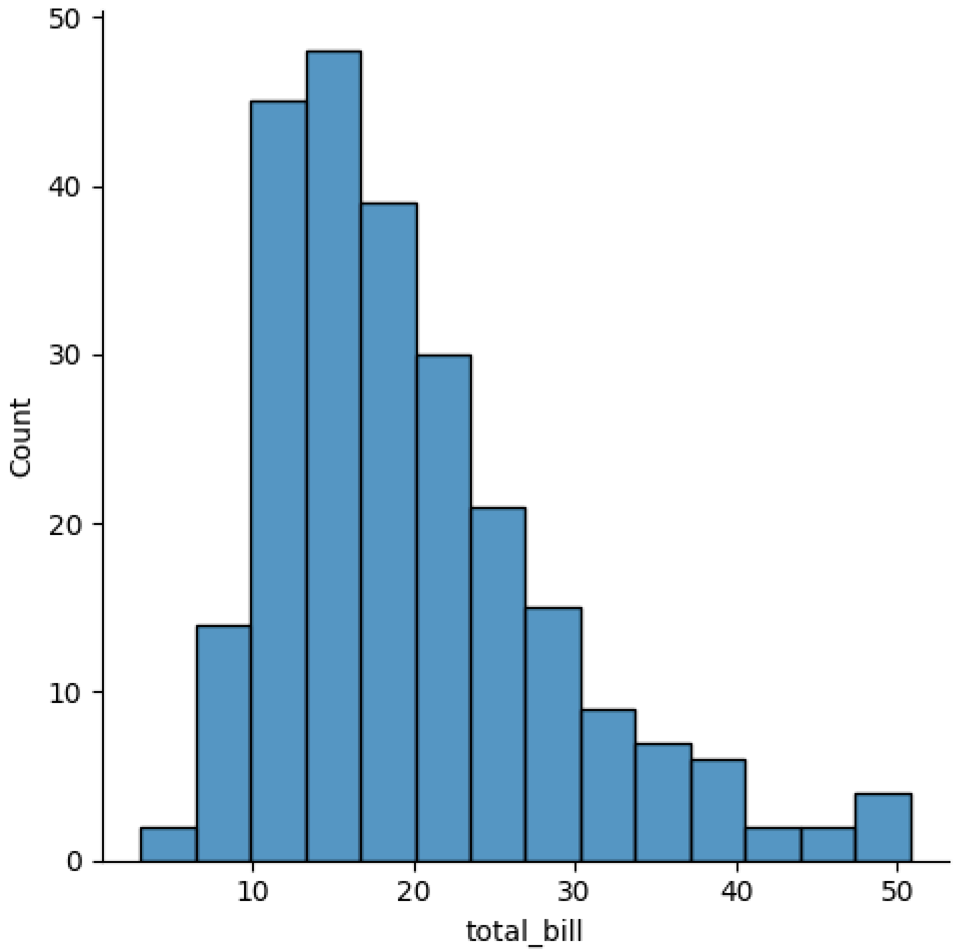
In [19]:
```
sns.displot(tips.total_bill,kde=True)
```

Out[19]: <seaborn.axisgrid.FacetGrid at 0x2d189642950>



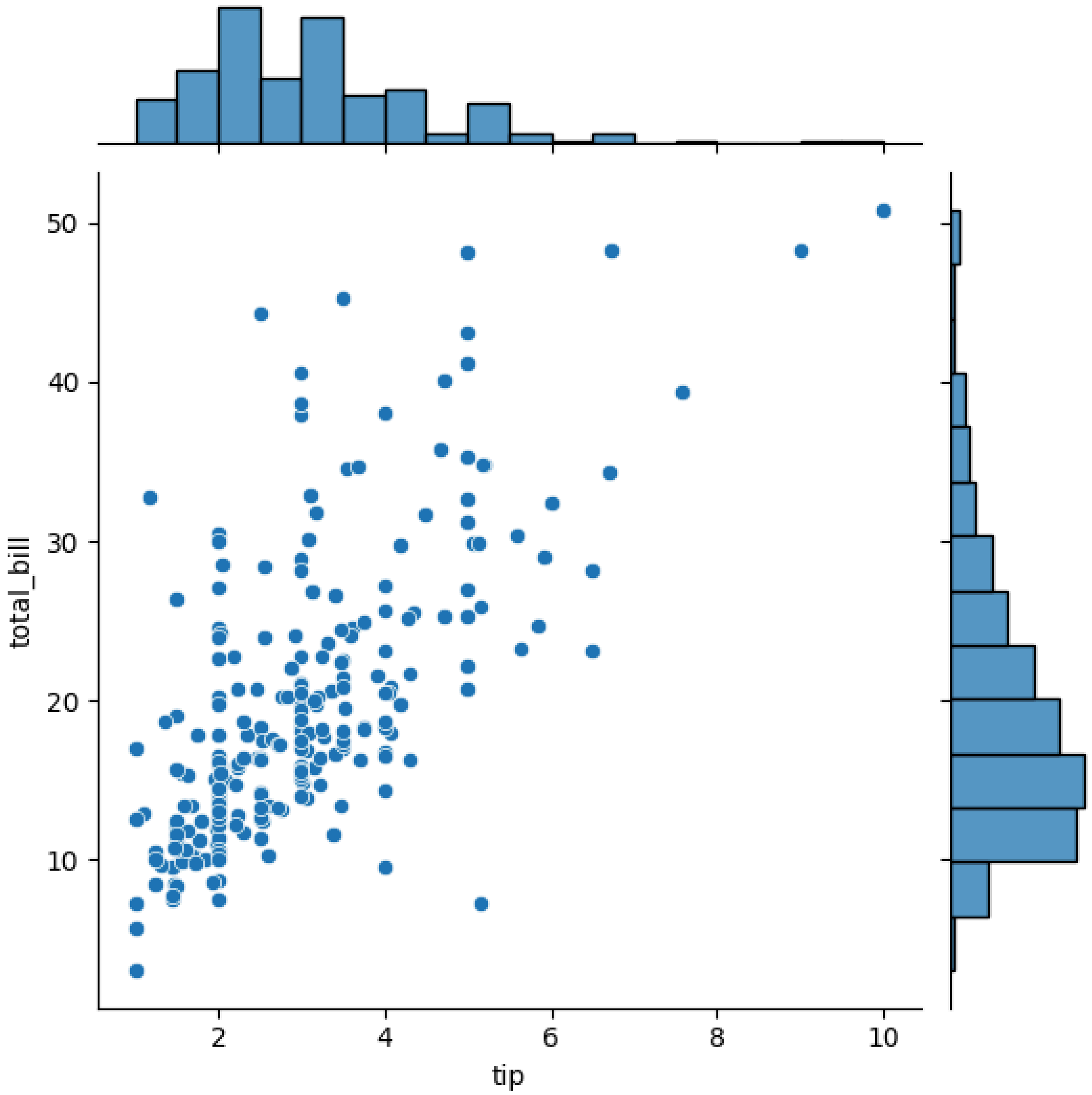
In [20]:
```
sns.displot(tips.total_bill,kde=False)
```

Out[20]: <seaborn.axisgrid.FacetGrid at 0x2d1998f7ac0>



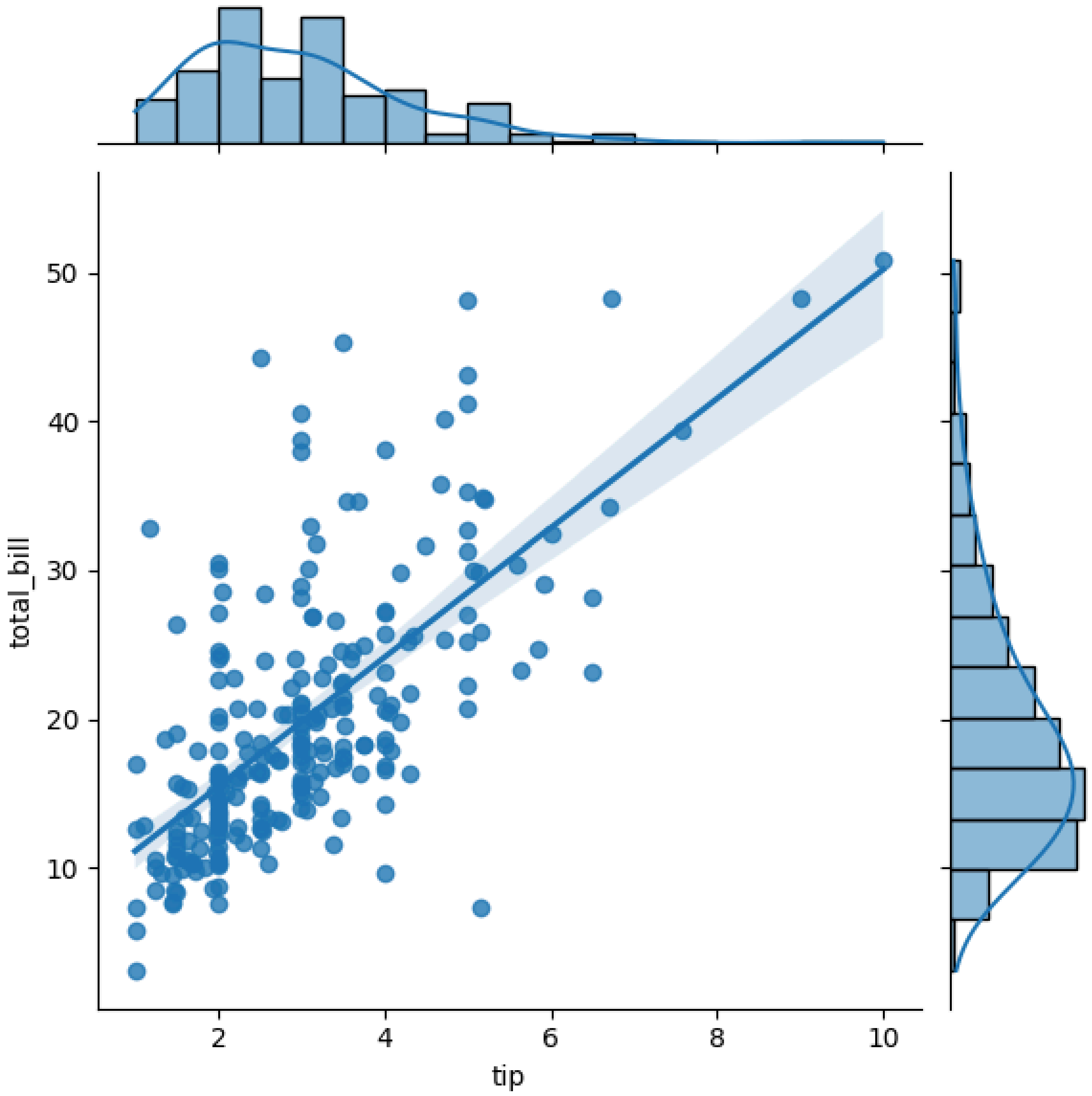
In [21]:
```
sns.jointplot(x=tips.tip,y=tips.total_bill)
```

Out[21]: <seaborn.axisgrid.JointGrid at 0x2d1b1d14a90>



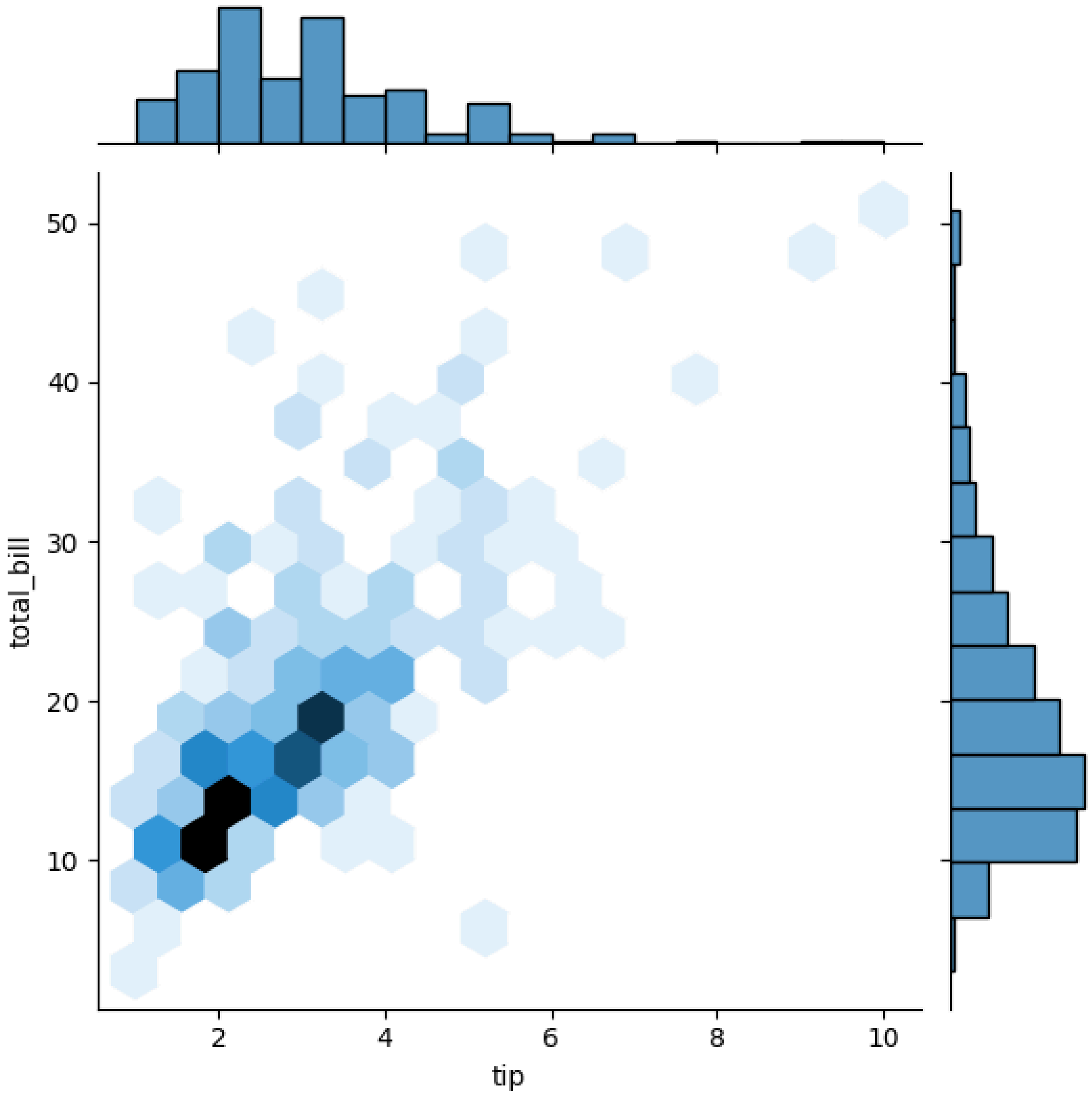
In [22]:
```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="reg")
```

Out[22]:
```
<seaborn.axisgrid.JointGrid at 0x2d1b1e34820>
```



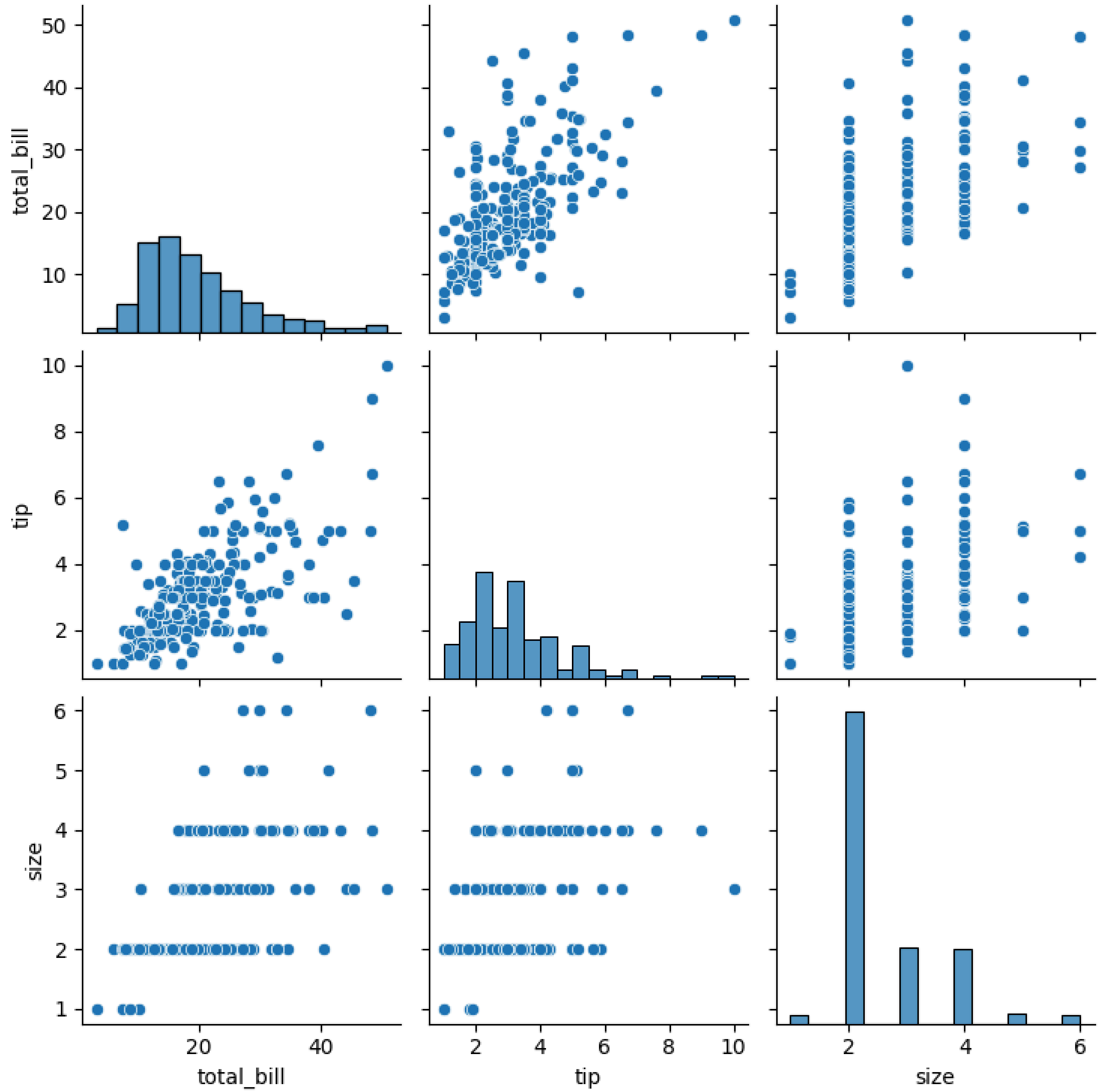
In [23]:
```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="hex")
```

Out[23]: <seaborn.axisgrid.JointGrid at 0x2d1b2446080>



In [24]:
```
sns.pairplot(tips)
```

Out[24]: <seaborn.axisgrid.PairGrid at 0x2d1b26a3790>

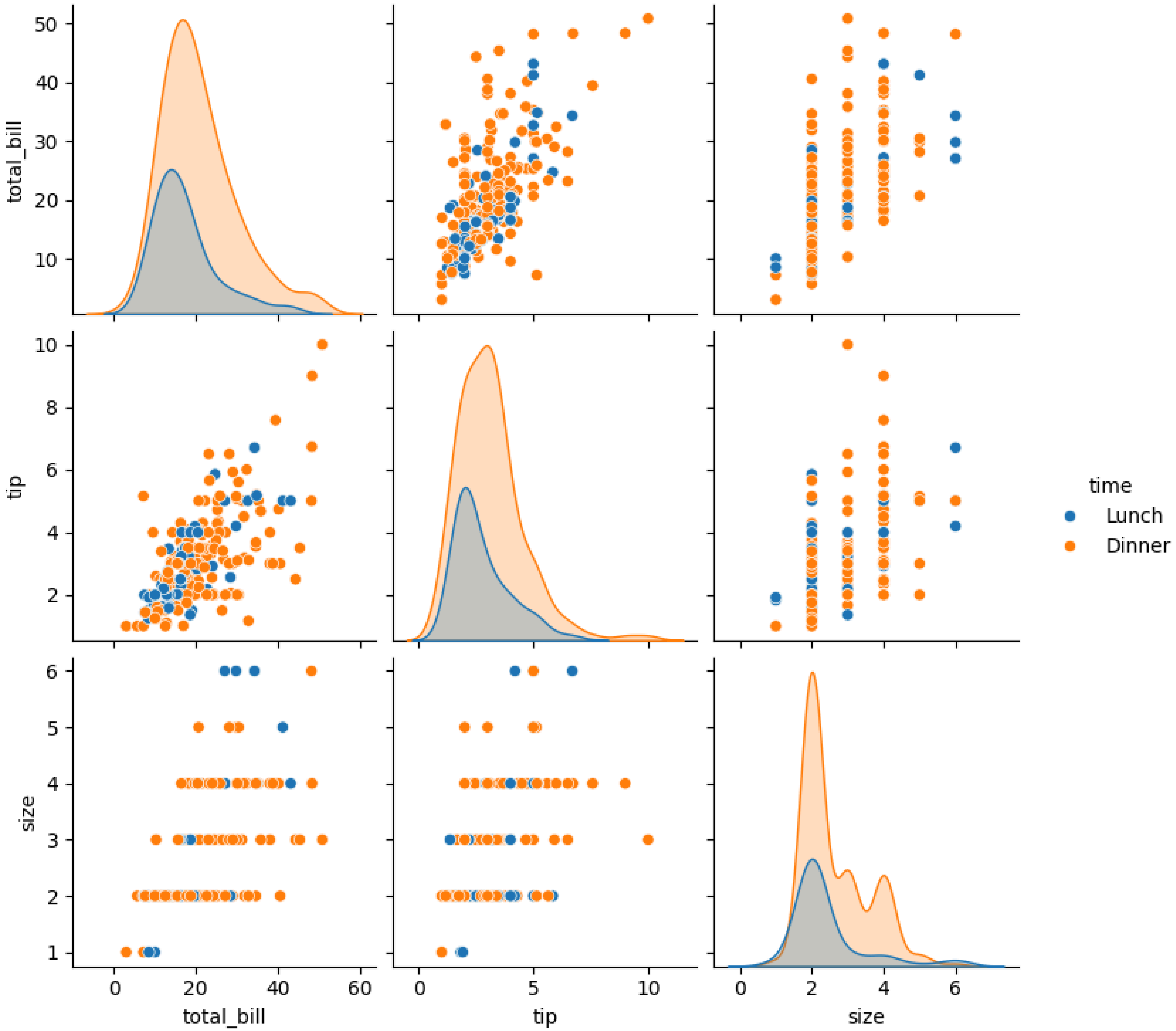


In [25]:
```
tips.time.value_counts()
```

Out[25]:
```
time
Dinner    176
Lunch      68
Name: count, dtype: int64
```

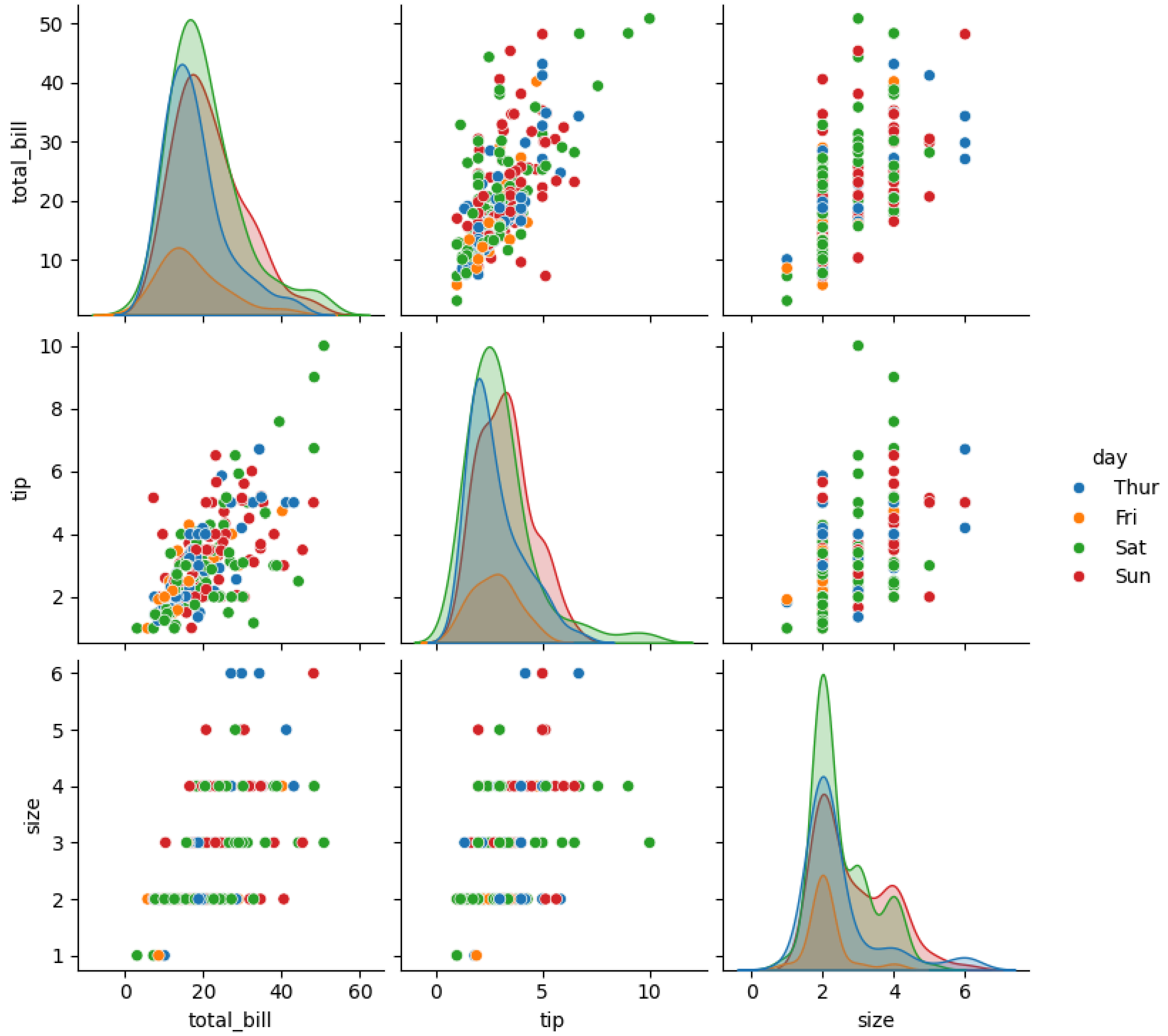
In [26]:
```
sns.pairplot(tips,hue='time')
```

Out[26]:
```
<seaborn.axisgrid.PairGrid at 0x2d1998f6800>
```



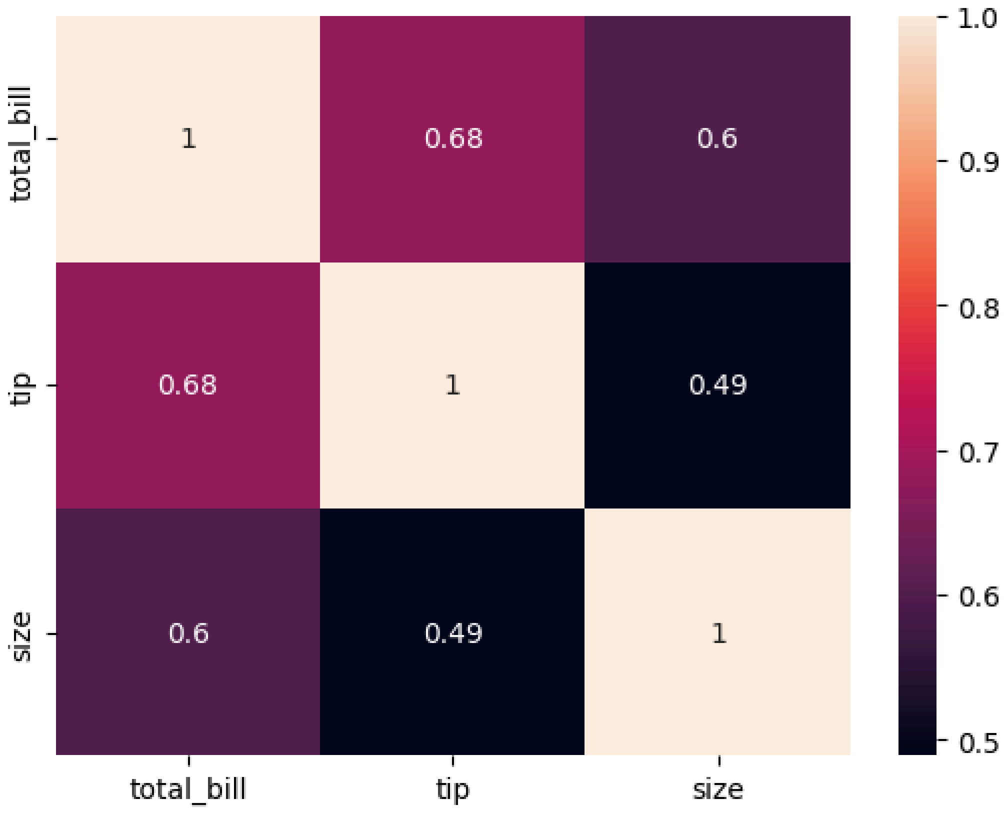

```
In [27]: sns.pairplot(tips,hue='day')
```

```
Out[27]: <seaborn.axisgrid.PairGrid at 0x2d1b398cd60>
```



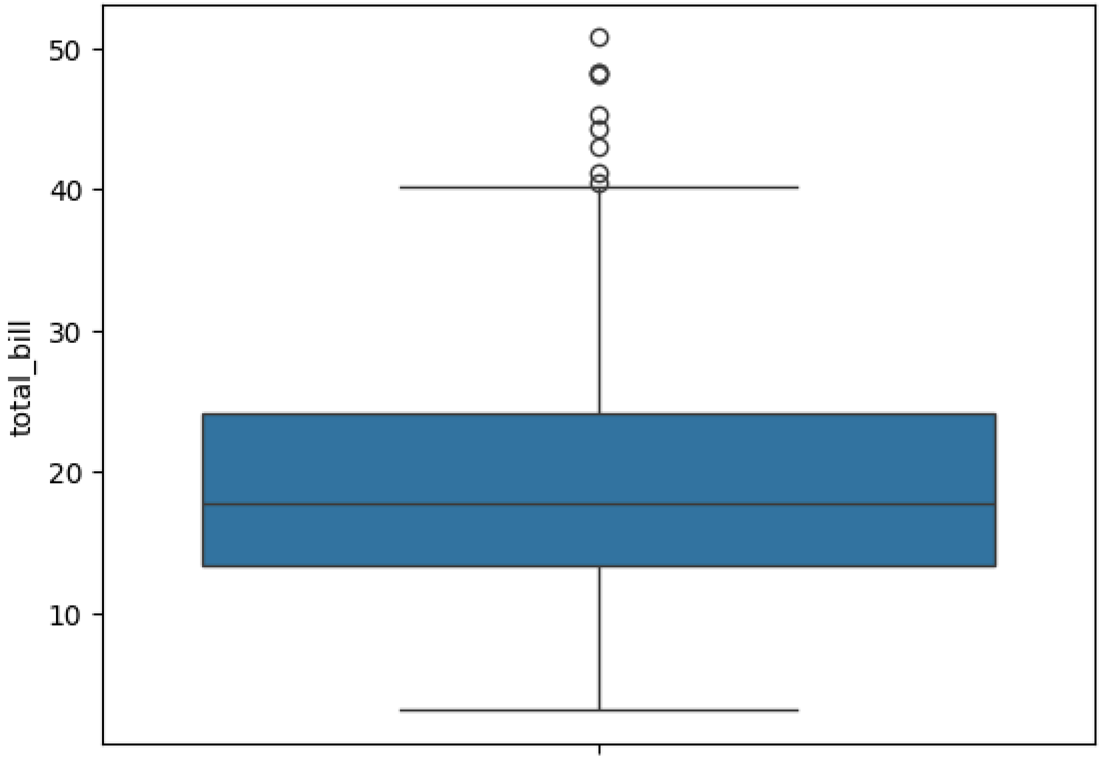
In [28]:
```
sns.heatmap(tips.corr(numeric_only=True),annot=True)
```

Out[28]: <Axes: >



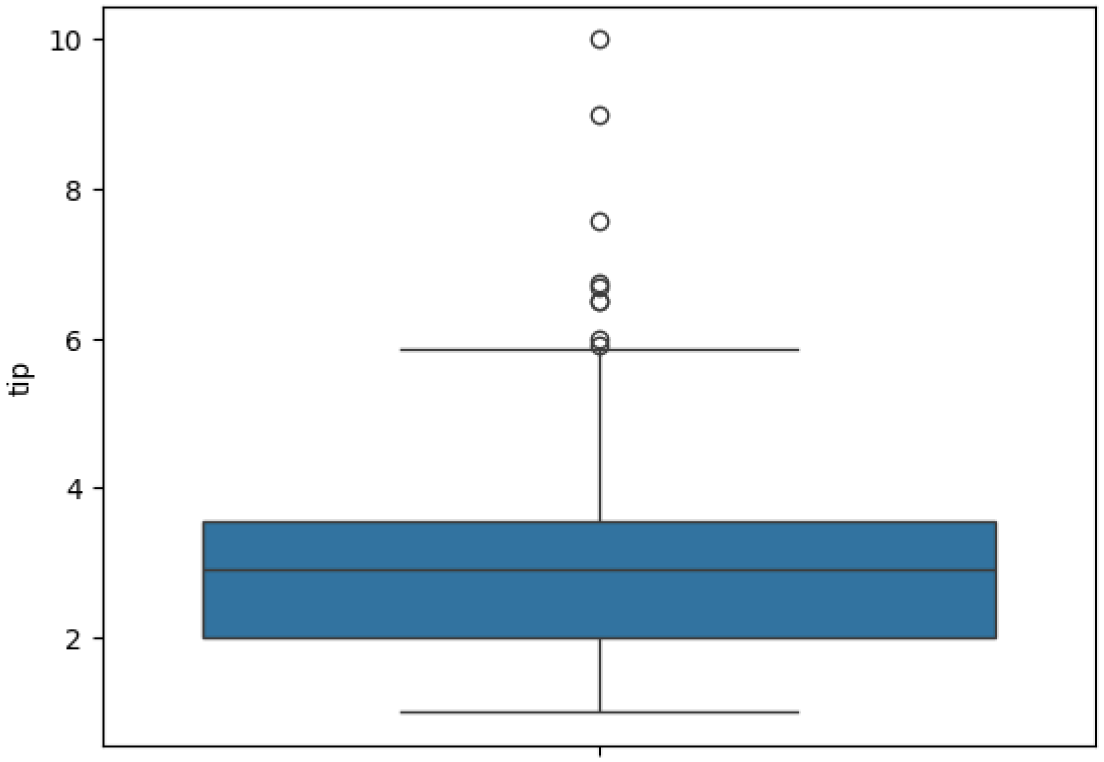
In [29]:
```
sns.boxplot(tips.total_bill)
```

Out[29]: <Axes: ylabel='total_bill'>



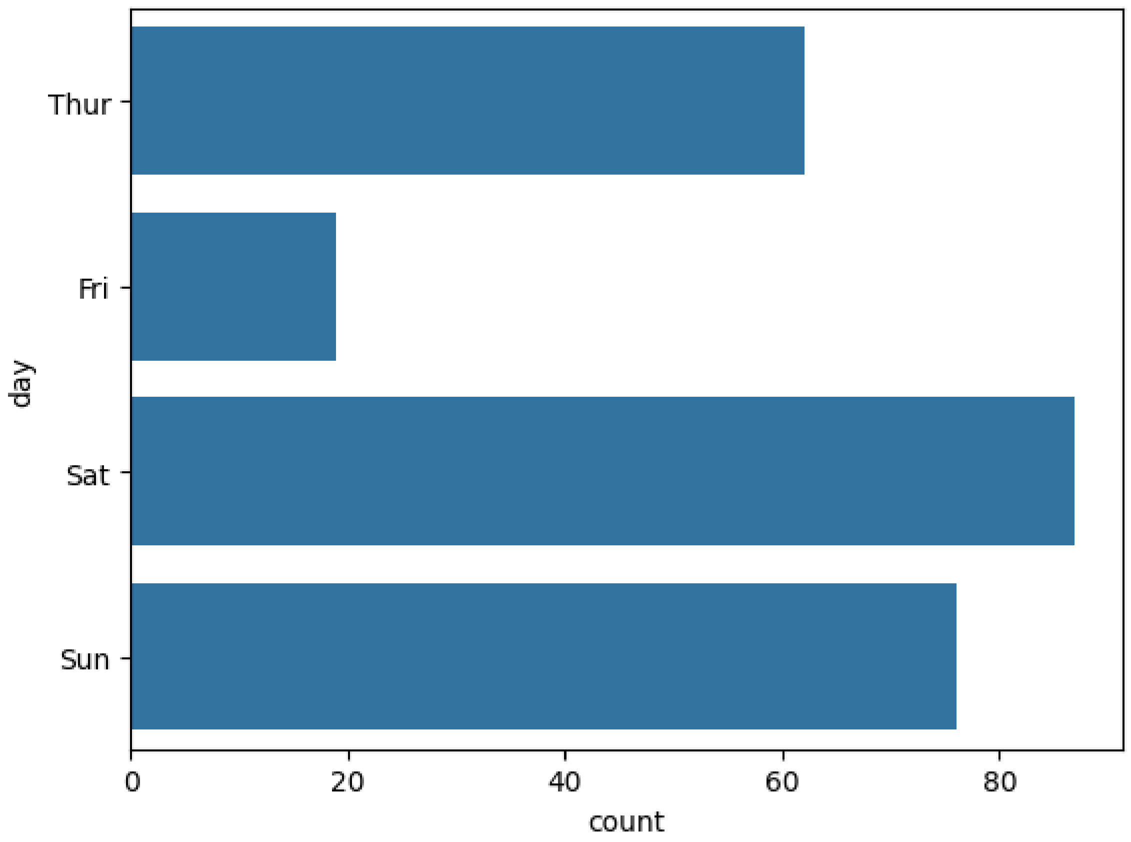
In [30]:
```
sns.boxplot(tips.tip)
```

Out[30]: <Axes: ylabel='tip'>



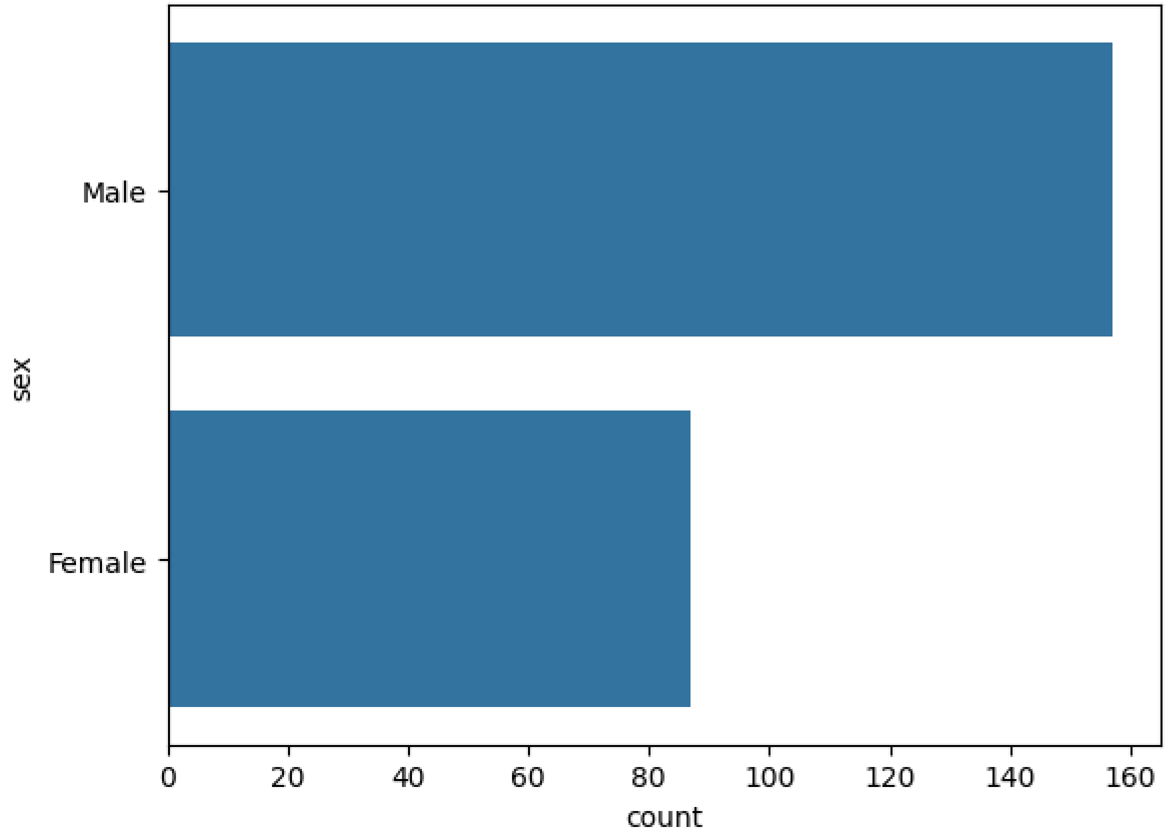

```
In [31]: sns.countplot(tips.day)
```

Out[31]: <Axes: xlabel='count', ylabel='day'>



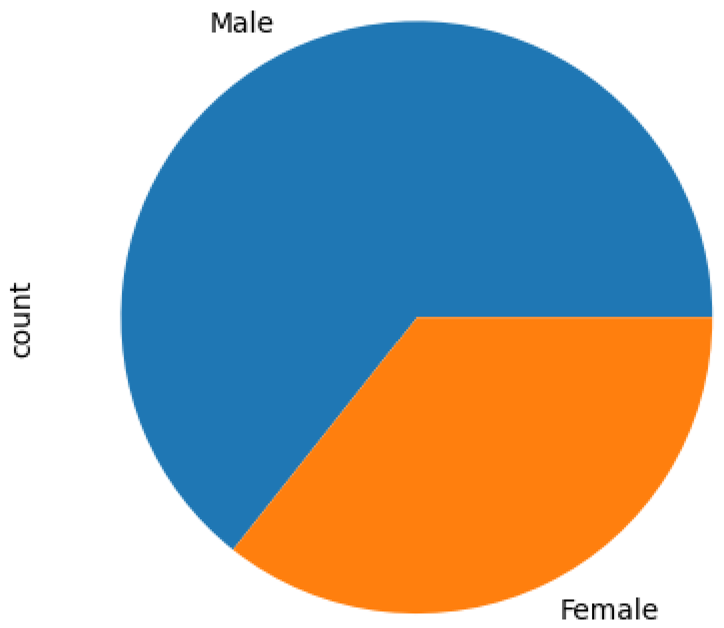

```
In [32]: sns.countplot(tips.sex)
```

Out[32]: <Axes: xlabel='count', ylabel='sex'>



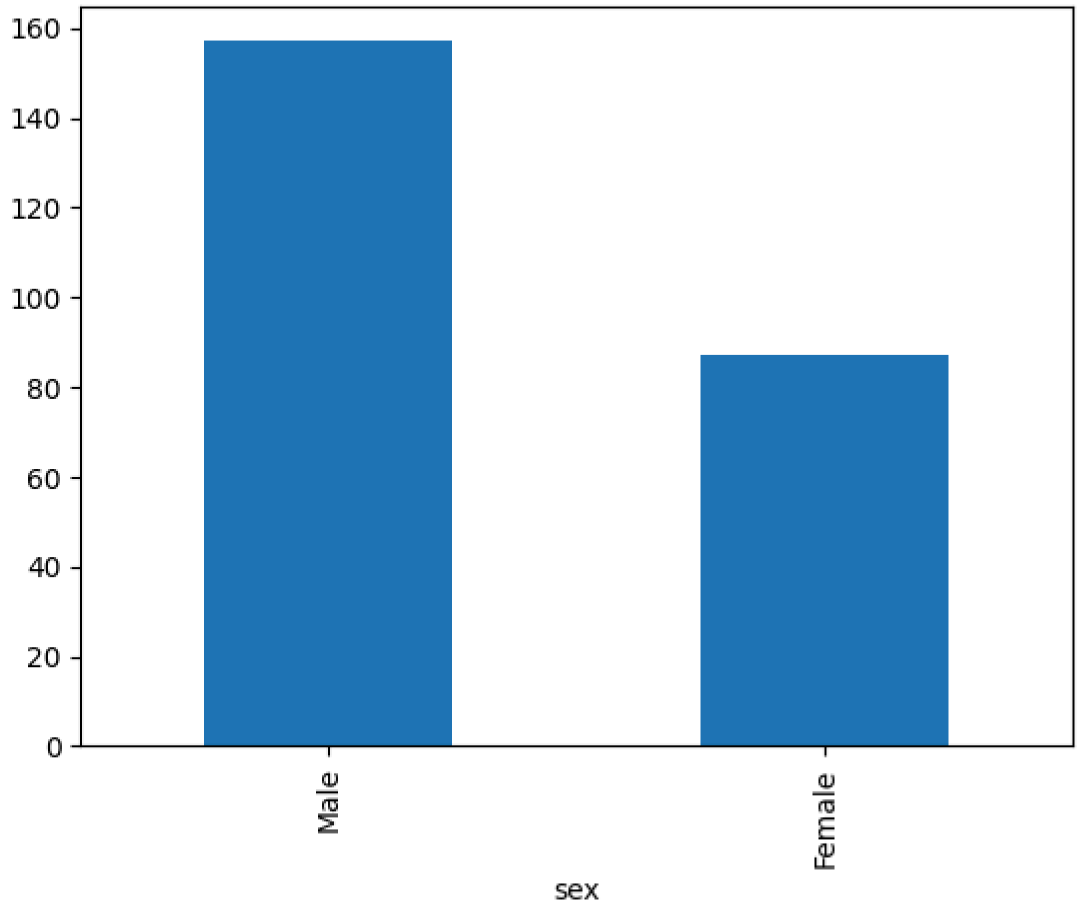
In [33]:
```
tips.sex.value_counts().plot(kind='pie')
```

Out[33]: <Axes: ylabel='count'>



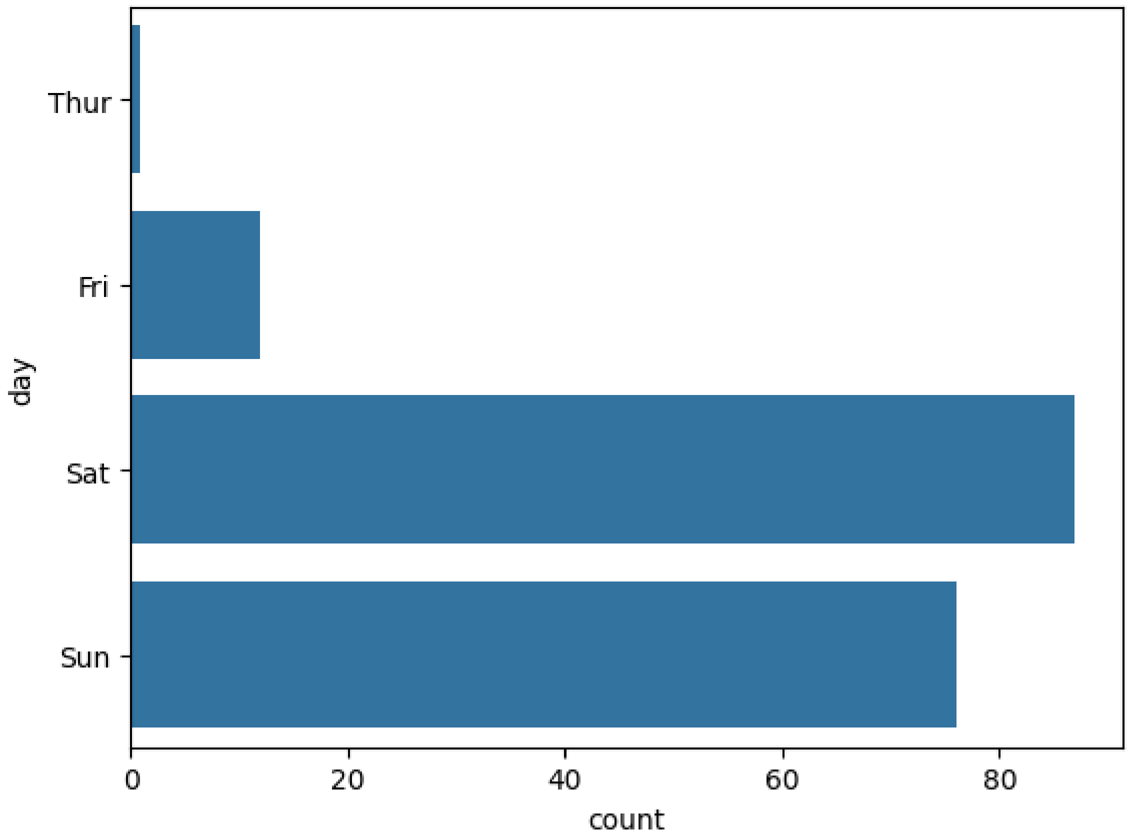
In [34]:
```
tips.sex.value_counts().plot(kind='bar')
```

Out[34]: <Axes: xlabel='sex'>



In [35]:
```
sns.countplot(tips[tips.time=='Dinner']['day'])
```

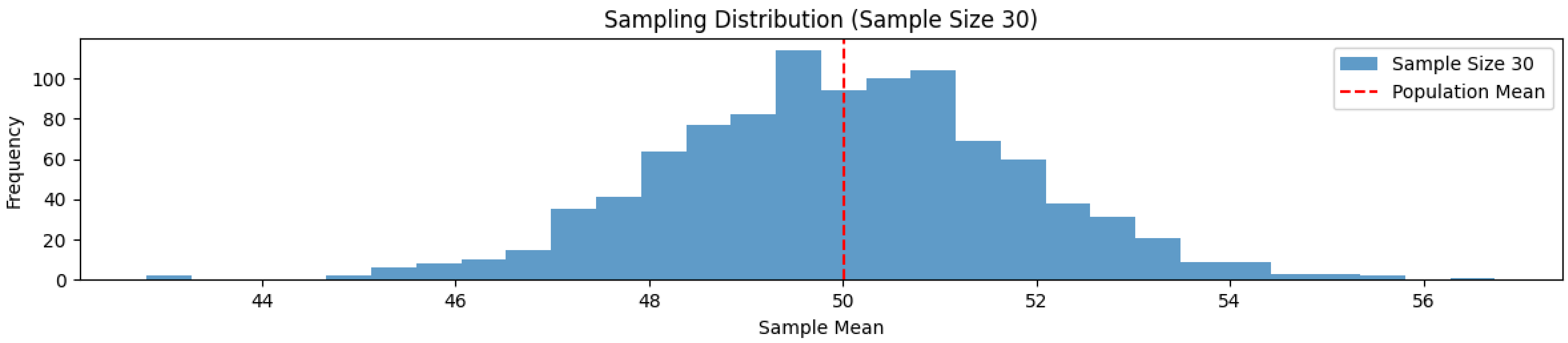
Out[35]: <Axes: xlabel='count', ylabel='day'>

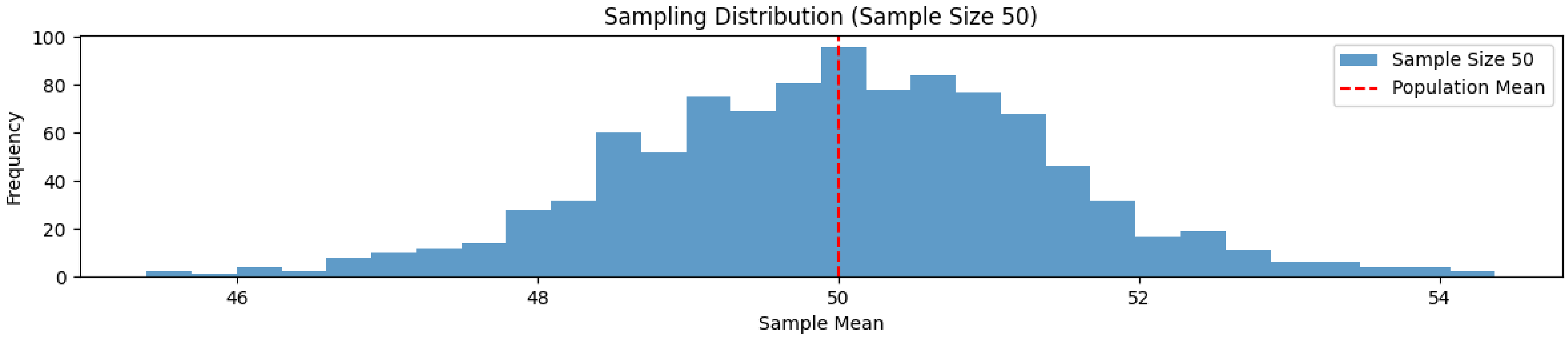


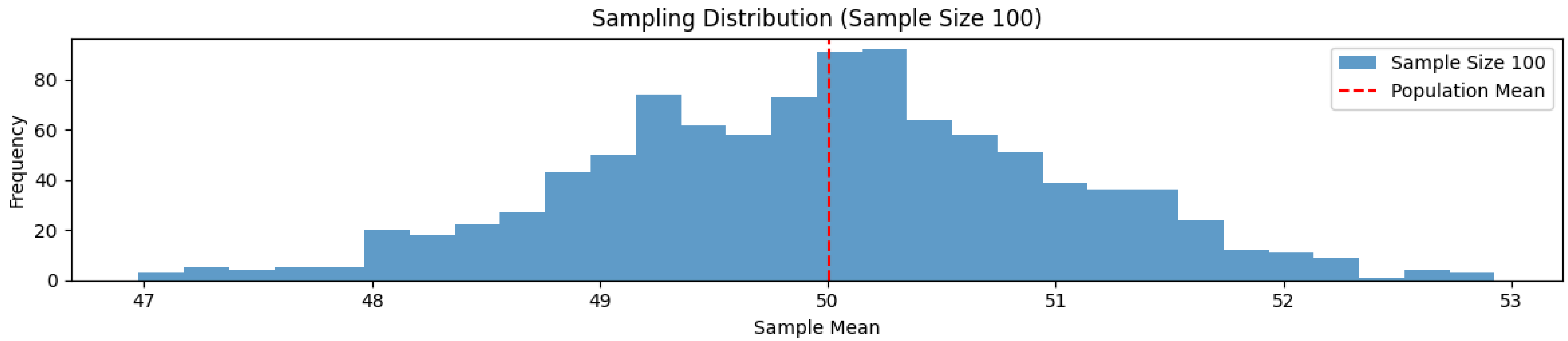
In [ ]:

In [41]:
```
# 6) Random Sampling and Sampling Distribution
# 230701032
# Aravinthaa S
# Date : 10.09.2024
```

```python
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Generate a population (e.g., normal distribution)
population_mean = 50
population_std = 10
population_size = 100000
population = np.random.normal(population_mean, population_std, population_size)

# Step 2: Random sampling
sample_sizes = [30, 50, 100]  # different sample sizes to consider
num_samples = 1000  # number of samples for each sample size
sample_means = {}

for size in sample_sizes:
    sample_means[size] = []

    for _ in range(num_samples):
        sample = np.random.choice(population, size=size, replace=False)
        sample_means[size].append(np.mean(sample))

# Step 3: Plotting sampling distributions
plt.figure(figsize=(12, 8))

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

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







In [ ]:

In [42]:

```
# 7) Z-Test
# 230701032
# Aravinthaa S
# Date : 10.09.2024

import numpy as np
import scipy.stats as stats

# Define the sample data (hypothetical weights in grams)
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])

# 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 standard deviation

# Number of observations
n = len(sample_data)

# 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}")

# Decision based on the significance level
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The average weight is significantly different from 150 grams.")
else:
    print("Fail to reject the null hypothesis: There is no significant difference in average weight from 150 grams.")
```

```
Sample Mean: 150.20
Z-Statistic: 0.6406
P-Value: 0.5218
Fail to reject the null hypothesis: There is no significant difference in average weight from 150 grams.
```

In [ ]:

In [43]:
```
# 8) T-Test
# 230701032
# Aravinthaa S
# Date : 08.10.2024

import numpy as np
import scipy.stats as stats

# Set a random seed for reproducibility
np.random.seed(42)

# Generate hypothetical sample data (IQ scores)
sample_size = 25
sample_data = np.random.normal(loc=102, scale=15, size=sample_size)  # Mean IQ of 102, SD of 15

# Population mean under the null hypothesis
population_mean = 100

# Calculate sample statistics
sample_mean = np.mean(sample_data)
sample_std = np.std(sample_data, ddof=1)  # Using sample standard deviation

# Number of observations
n = len(sample_data)

# Calculate the T-statistic and p-value
t_statistic, p_value = stats.ttest_1samp(sample_data, population_mean)

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

```python
# Decision based on the significance level
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The average IQ score is significantly different from 100.")
else:
    print("Fail to reject the null hypothesis: There is no significant difference in average IQ score from 100.")
```

```
Sample Mean: 99.55
T-Statistic: -0.1577
P-Value: 0.8760
Fail to reject the null hypothesis: There is no significant difference in average IQ score from 100.
```

In [ ]:

In [44]:
```python
# 9) Annova TEST
# 230701032
# Aravinthaa S
# Date : 08.10.2024

import numpy as np
import scipy.stats as stats

# Set a random seed for reproducibility
np.random.seed(42)

# Generate hypothetical growth data for three treatments (A, B, C)
n_plants = 25

# Growth data (in cm) for Treatment A, B, and C
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)

# Combine all data into one array
all_data = np.concatenate([growth_A, growth_B, growth_C])

# Treatment labels for each group
treatment_labels = ['A'] * n_plants + ['B'] * n_plants + ['C'] * n_plants

# Perform one-way ANOVA
f_statistic, p_value = stats.f_oneway(growth_A, growth_B, growth_C)

# Print results
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))
print()
print(f"F-Statistic: {f_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")

# Decision based on the significance level
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference in mean growth rates among the three treatments.")
else:
    print("Fail to reject the null hypothesis: There is no significant difference in mean growth rates among the three treatments.")

# Additional: Post-hoc analysis (Tukey's HSD) if ANOVA is significant
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)
```

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

F-Statistic: 36.1214
P-Value: 0.0000
Reject the null hypothesis: There is a significant difference in mean growth rates among the three treatments.

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    0.0  3.9593 7.2252   True
     B      C   4.1276    0.0  2.4946 5.7605   True
----------------------------------------------------
```

In [ ]:

In [45]:

```
# 10) Fedature Scaling
# 230701032
# Aravinthaa S
# Date : 22.10.2024

import numpy as np
import pandas as pd
df=pd.read_csv('pre-process_datasample.csv')
df
```

Out[45]:

	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	NaN	50.0	83000.0	No
9	France	37.0	67000.0	Yes

In [46]:

```
df.head()
```

Out[46]:

	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 [47]:
```
df.Country.fillna(df.Country.mode()[0],inplace=True)
features=df.iloc[:,:-1].values
```

```
C:\Users\91950\AppData\Local\Temp\ipykernel_13364\3424832005.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


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

In [49]:
```
label=df.iloc[:,-1].values
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[49]:
```
▾ SimpleImputer
SimpleImputer()
```

In [50]:
```
Salary.fit(features[:,[2]])
```

Out[50]:
```
▾ SimpleImputer
SimpleImputer()
```

In [51]:
```
SimpleImputer()
```

Out[51]:
```
▾ SimpleImputer
SimpleImputer()
```

In [52]:
```
features[:,[1]]=age.transform(features[:,[1]])
features[:,[2]]=Salary.transform(features[:,[2]])
features
```

Out[52]:
```
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],
       ['France', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

In [53]:
```
from sklearn.preprocessing import OneHotEncoder
oh = OneHotEncoder(sparse_output=False)
Country=oh.fit_transform(features[:,[0]])
Country
```

Out[53]:
```
array([[1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.]])
```

In [54]:
```
final_set=np.concatenate((Country,features[:,[1,2]]),axis=1)
final_set
```

Out[54]:
```
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],
       [1.0, 0.0, 0.0, 50.0, 83000.0],
       [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

In [55]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit(final_set)
feat_standard_scaler=sc.transform(final_set)
feat_standard_scaler
```

Out[55]:
```
array([[ 1.00000000e+00, -5.00000000e-01, -6.54653671e-01,
         7.58874362e-01,  7.49473254e-01],
       [-1.00000000e+00, -5.00000000e-01,  1.52752523e+00,
        -1.71150388e+00, -1.43817841e+00],
       [-1.00000000e+00,  2.00000000e+00, -6.54653671e-01,
        -1.27555478e+00, -8.91265492e-01],
       [-1.00000000e+00, -5.00000000e-01,  1.52752523e+00,
        -1.13023841e-01, -2.53200424e-01],
       [-1.00000000e+00,  2.00000000e+00, -6.54653671e-01,
         1.77608893e-01,  6.63219199e-16],
       [ 1.00000000e+00, -5.00000000e-01, -6.54653671e-01,
        -5.48972942e-01, -5.26656882e-01],
       [-1.00000000e+00, -5.00000000e-01,  1.52752523e+00,
         0.00000000e+00, -1.07356980e+00],
       [ 1.00000000e+00, -5.00000000e-01, -6.54653671e-01,
         1.34013983e+00,  1.38753832e+00],
       [ 1.00000000e+00, -5.00000000e-01, -6.54653671e-01,
         1.63077256e+00,  1.75214693e+00],
       [ 1.00000000e+00, -5.00000000e-01, -6.54653671e-01,
        -2.58340208e-01,  2.93712492e-01]])
```

In [56]:
```
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[56]:
```
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],
       [1.        , 0.        , 0.        , 1.        , 1.        ],
       [1.        , 0.        , 0.        , 0.43478261, 0.54285714]])
```

In [ ]:

In [57]:
```python
# 11) Linear Regression
# 230701032
# Aravinthaa S
# Date : 29.10.2024

import numpy as np
import pandas as pd
df=pd.read_csv('Salary_data.csv')
df
```

Out[57]:

	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 [58]:

```
df.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 [59]:
```
df.dropna(inplace=True)
```

In [60]:
```
df.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 [61]:
```
df.describe()
```

Out[61]:

	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 [62]:
```
features=df.iloc[:,[0]].values
label=df.iloc[:,[1]].values
```

In [63]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2, random_state=42)
```

In [64]:
```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
```

Out[64]:

```
▾ LinearRegression
LinearRegression()
```

In [65]:
```
model.score(x_train,y_train)
```

Out[65]:
```
0.9645401573418146
```

In [66]:
```
model.score(x_test,y_test)
```

Out[66]: 0.9024461774180497

In [67]:
```
model.coef_
```
Out[67]: array([[9423.81532303]])

In [68]:
```
model.intercept_
```
Out[68]: array([25321.58301178])

In [69]:
```
import pickle
pickle.dump(model,open('SalaryPred.model','wb'))
```

In [70]:
```
model=pickle.load(open('SalaryPred.model','rb'))
```

In [71]:
```
yr_of_exp=float(input("Enter Years of Experience: "))
yr_of_exp_NP=np.array([[yr_of_exp]])
Salary=model.predict(yr_of_exp_NP)
```

In [73]:
```
print("Estimated Salary for {} years of experience is {}: " .format(yr_of_exp,Salary))
```
```
Estimated Salary for 44.0 years of experience is [[439969.45722514]]:
```

In [ ]:

In [75]:
```
# 12) Logistic Regression
# 230701032
# Aravinthaa S
# Date : 05.11.2024

import numpy as np
import pandas as pd
df=pd.read_csv('Social_Network_Ads.csv')
df
```

Out[75]:

	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
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

In [76]:
```
df.head()
```

Out[76]:

	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

In [77]:

```
features=df.iloc[:,[2,3]].values
label=df.iloc[:,4].values
features
```

```
Out[77]: 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 [78]:
```python
label
```

Out[78]:
```
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 [79]:
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [80]:
```python
for i in range(1,401):
  x_train,x_test,y_train,y_test=train_test_split(features,label,test_size=0.2,random_state=i)
  model=LogisticRegression()
  model.fit(x_train,y_train)
  train_score=model.score(x_train,y_train)
  test_score=model.score(x_test,y_test)
  if test_score>train_score:
    print("Test {} Train{} Random State {}".format(test_score,train_score,i))
```

```
Test 0.9 Train0.840625 Random State 4
Test 0.8625 Train0.85 Random State 5
Test 0.8625 Train0.859375 Random State 6
Test 0.8875 Train0.8375 Random State 7
Test 0.8625 Train0.8375 Random State 9
Test 0.9 Train0.840625 Random State 10
Test 0.8625 Train0.85625 Random State 14
Test 0.85 Train0.84375 Random State 15
Test 0.8625 Train0.85625 Random State 16
Test 0.875 Train0.834375 Random State 18
Test 0.85 Train0.84375 Random State 19
Test 0.875 Train0.84375 Random State 20
Test 0.8625 Train0.834375 Random State 21
Test 0.875 Train0.840625 Random State 22
Test 0.875 Train0.840625 Random State 24
Test 0.85 Train0.834375 Random State 26
Test 0.85 Train0.840625 Random State 27
Test 0.8625 Train0.834375 Random State 30
Test 0.8625 Train0.85625 Random State 31
Test 0.875 Train0.853125 Random State 32
Test 0.8625 Train0.84375 Random State 33
Test 0.875 Train0.83125 Random State 35
Test 0.8625 Train0.853125 Random State 36
Test 0.8875 Train0.840625 Random State 38
Test 0.875 Train0.8375 Random State 39
Test 0.8875 Train0.8375 Random State 42
Test 0.875 Train0.846875 Random State 46
Test 0.9125 Train0.83125 Random State 47
Test 0.875 Train0.83125 Random State 51
Test 0.9 Train0.84375 Random State 54
Test 0.85 Train0.84375 Random State 57
Test 0.875 Train0.84375 Random State 58
Test 0.925 Train0.8375 Random State 61
Test 0.8875 Train0.834375 Random State 65
Test 0.8875 Train0.840625 Random State 68
Test 0.9 Train0.83125 Random State 72
Test 0.8875 Train0.8375 Random State 75
Test 0.925 Train0.825 Random State 76
Test 0.8625 Train0.840625 Random State 77
Test 0.8625 Train0.859375 Random State 81
Test 0.875 Train0.8375 Random State 82
Test 0.8875 Train0.8375 Random State 83
Test 0.8625 Train0.853125 Random State 84
Test 0.8625 Train0.840625 Random State 85
Test 0.8625 Train0.840625 Random State 87
Test 0.875 Train0.846875 Random State 88
Test 0.9125 Train0.8375 Random State 90
Test 0.8625 Train0.85 Random State 95
Test 0.875 Train0.85 Random State 99
Test 0.85 Train0.840625 Random State 101
Test 0.85 Train0.840625 Random State 102
Test 0.9 Train0.825 Random State 106
Test 0.8625 Train0.840625 Random State 107
Test 0.85 Train0.834375 Random State 109
Test 0.85 Train0.840625 Random State 111
Test 0.9125 Train0.840625 Random State 112
Test 0.8625 Train0.85 Random State 115
Test 0.8625 Train0.840625 Random State 116
Test 0.875 Train0.834375 Random State 119
Test 0.9125 Train0.828125 Random State 120
Test 0.8625 Train0.859375 Random State 125
Test 0.85 Train0.846875 Random State 128
Test 0.875 Train0.85 Random State 130
Test 0.9 Train0.84375 Random State 133
Test 0.925 Train0.834375 Random State 134
Test 0.8625 Train0.85 Random State 135
```

```
Test 0.875 Train0.83125 Random State 138
Test 0.8625 Train0.85 Random State 141
Test 0.85 Train0.846875 Random State 143
Test 0.85 Train0.846875 Random State 146
Test 0.85 Train0.84375 Random State 147
Test 0.8625 Train0.85 Random State 148
Test 0.875 Train0.8375 Random State 150
Test 0.8875 Train0.83125 Random State 151
Test 0.925 Train0.84375 Random State 152
Test 0.85 Train0.840625 Random State 153
Test 0.9 Train0.84375 Random State 154
Test 0.9 Train0.840625 Random State 155
Test 0.8875 Train0.846875 Random State 156
Test 0.8875 Train0.834375 Random State 158
Test 0.875 Train0.828125 Random State 159
Test 0.9 Train0.83125 Random State 161
Test 0.85 Train0.8375 Random State 163
Test 0.875 Train0.83125 Random State 164
Test 0.8625 Train0.85 Random State 169
Test 0.875 Train0.840625 Random State 171
Test 0.85 Train0.840625 Random State 172
Test 0.9 Train0.825 Random State 180
Test 0.85 Train0.834375 Random State 184
Test 0.925 Train0.821875 Random State 186
Test 0.9 Train0.83125 Random State 193
Test 0.8625 Train0.85 Random State 195
Test 0.8625 Train0.840625 Random State 196
Test 0.8625 Train0.8375 Random State 197
Test 0.875 Train0.840625 Random State 198
Test 0.8875 Train0.8375 Random State 199
Test 0.8875 Train0.84375 Random State 200
Test 0.8625 Train0.8375 Random State 202
Test 0.8625 Train0.840625 Random State 203
Test 0.8875 Train0.83125 Random State 206
Test 0.8625 Train0.834375 Random State 211
Test 0.85 Train0.84375 Random State 212
Test 0.8625 Train0.834375 Random State 214
Test 0.875 Train0.83125 Random State 217
Test 0.9625 Train0.81875 Random State 220
Test 0.875 Train0.84375 Random State 221
Test 0.85 Train0.840625 Random State 222
Test 0.9 Train0.84375 Random State 223
Test 0.8625 Train0.853125 Random State 227
Test 0.8625 Train0.834375 Random State 228
Test 0.9 Train0.840625 Random State 229
Test 0.85 Train0.84375 Random State 232
Test 0.875 Train0.846875 Random State 233
Test 0.9125 Train0.840625 Random State 234
Test 0.8625 Train0.840625 Random State 235
Test 0.85 Train0.846875 Random State 236
Test 0.875 Train0.846875 Random State 239
Test 0.85 Train0.84375 Random State 241
Test 0.8875 Train0.85 Random State 242
Test 0.8875 Train0.825 Random State 243
Test 0.875 Train0.846875 Random State 244
Test 0.875 Train0.840625 Random State 245
Test 0.875 Train0.846875 Random State 246
Test 0.8625 Train0.859375 Random State 247
Test 0.8875 Train0.84375 Random State 248
Test 0.8625 Train0.85 Random State 250
Test 0.875 Train0.83125 Random State 251
Test 0.8875 Train0.84375 Random State 252
Test 0.8625 Train0.846875 Random State 255
Test 0.9 Train0.840625 Random State 257
Test 0.8625 Train0.85625 Random State 260
Test 0.8625 Train0.840625 Random State 266
```

```
Test 0.8625 Train0.8375 Random State 268
Test 0.875 Train0.840625 Random State 275
Test 0.8625 Train0.85 Random State 276
Test 0.925 Train0.8375 Random State 277
Test 0.875 Train0.846875 Random State 282
Test 0.85 Train0.846875 Random State 283
Test 0.85 Train0.84375 Random State 285
Test 0.9125 Train0.834375 Random State 286
Test 0.85 Train0.840625 Random State 290
Test 0.85 Train0.840625 Random State 291
Test 0.85 Train0.846875 Random State 292
Test 0.8625 Train0.8375 Random State 294
Test 0.8875 Train0.828125 Random State 297
Test 0.8625 Train0.834375 Random State 300
Test 0.8625 Train0.85 Random State 301
Test 0.8875 Train0.85 Random State 302
Test 0.875 Train0.846875 Random State 303
Test 0.8625 Train0.834375 Random State 305
Test 0.9125 Train0.8375 Random State 306
Test 0.875 Train0.846875 Random State 308
Test 0.9 Train0.84375 Random State 311
Test 0.8625 Train0.834375 Random State 313
Test 0.9125 Train0.834375 Random State 314
Test 0.875 Train0.8375 Random State 315
Test 0.9 Train0.846875 Random State 317
Test 0.9125 Train0.821875 Random State 319
Test 0.8625 Train0.85 Random State 321
Test 0.9125 Train0.828125 Random State 322
Test 0.85 Train0.846875 Random State 328
Test 0.85 Train0.8375 Random State 332
Test 0.8875 Train0.853125 Random State 336
Test 0.85 Train0.8375 Random State 337
Test 0.875 Train0.840625 Random State 343
Test 0.8625 Train0.84375 Random State 346
Test 0.8875 Train0.83125 Random State 351
Test 0.8625 Train0.85 Random State 352
Test 0.95 Train0.81875 Random State 354
Test 0.8625 Train0.85 Random State 356
Test 0.9125 Train0.840625 Random State 357
Test 0.8625 Train0.8375 Random State 358
Test 0.85 Train0.840625 Random State 362
Test 0.9 Train0.84375 Random State 363
Test 0.8625 Train0.853125 Random State 364
Test 0.9375 Train0.821875 Random State 366
Test 0.9125 Train0.840625 Random State 369
Test 0.8625 Train0.853125 Random State 371
Test 0.925 Train0.834375 Random State 376
Test 0.9125 Train0.828125 Random State 377
Test 0.8875 Train0.85 Random State 378
Test 0.8875 Train0.85 Random State 379
Test 0.8625 Train0.840625 Random State 382
Test 0.8625 Train0.859375 Random State 386
Test 0.85 Train0.8375 Random State 387
Test 0.875 Train0.828125 Random State 388
Test 0.85 Train0.84375 Random State 394
Test 0.8625 Train0.8375 Random State 395
Test 0.9 Train0.84375 Random State 397
Test 0.8625 Train0.84375 Random State 400
```

In [81]:

```
x_train,x_test,y_train,y_test=train_test_split(features,label,test_size=0.2,random_state=42)
finalModel=LogisticRegression()
finalModel.fit(x_train,y_train)
```

Out[81]:

```
▾ LogisticRegression
LogisticRegression()
```

In [82]:

```
print(finalModel.score(x_train,y_train))
print(finalModel.score(x_test,y_test))
```

```
0.8375
0.8875
```

In [83]:

```
from sklearn.metrics import classification_report
print(classification_report(label,finalModel.predict(features)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.93      0.89       257
           1       0.85      0.70      0.77       143

    accuracy                           0.85       400
   macro avg       0.85      0.81      0.83       400
weighted avg       0.85      0.85      0.84       400
```

In [ ]: