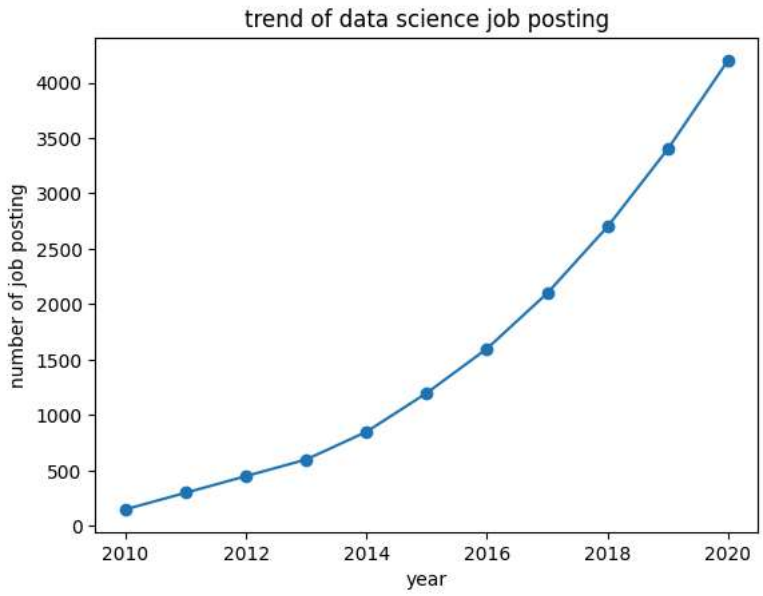
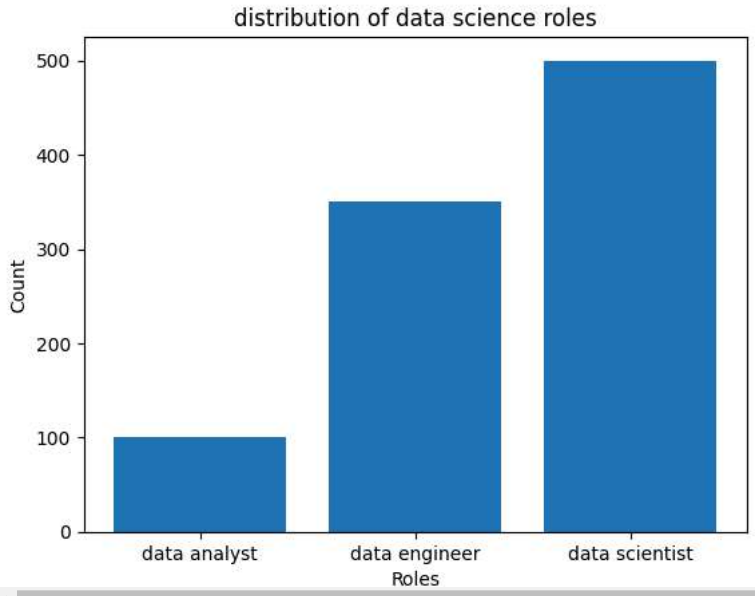
```
import pandas as pd
import matplotlib.pyplot as plt
data={'year':list(range(2010,2021)),'job posting':[150,300,450,600,850,1200,1600,2100,2700,3400,4200]}
df=pd.DataFrame(data)
plt.plot(df['year'],df['job posting'],marker='o')
plt.title('trend of data science job posting')
plt.xlabel('year')
plt.ylabel('number of job posting')
plt.show()
```

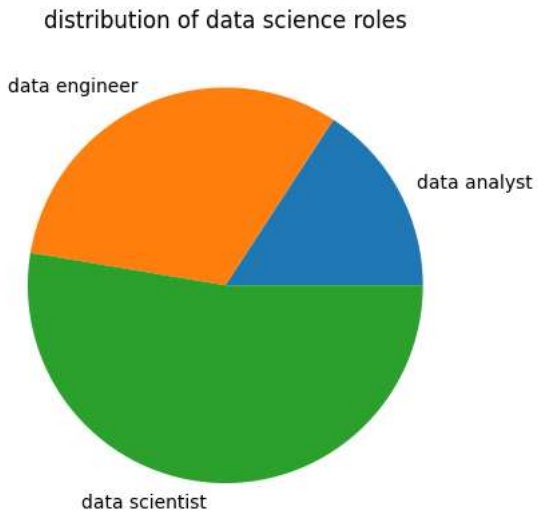


```
import matplotlib.pyplot as plt
Roles=['data analyst','data engineer','data scientist']
Count=[100,350,500]
plt.bar(Roles,Count)
plt.title('distribution of data science roles')
plt.xlabel('Roles')
plt.ylabel('Count')
plt.show()
```



```
import matplotlib.pyplot as plt
Roles=['data analyst','data engineer','data scientist']
Count=[150,300,500]
plt.pie(Count,labels=Roles)
plt.title('distribution of data science roles')
```

```
Text(0.5, 1.0, 'distribution of data science roles')
```



```
structured_data = pd.DataFrame({
'ID': [1, 2, 3],
'Name': ['Alice', 'Bob', 'Charlie'],
'Age': [25, 30, 35]
})
print('Structured Data:\n', structured_data)
```

```
Structured Data:
      ID     Name  Age
0   1    Alice   25
1   2      Bob   30
2   3  Charlie   35
```

```
import pandas as pd
semistructured_data={
    'ID':1,
    'NAME':'ALICE',
    'ATTRIBUTES':{'HEIGHT': 165, 'WEIGHT': 65}
}
print(semistructured_data)
```

```
{'ID': 1, 'NAME': 'ALICE', 'ATTRIBUTES': {'HEIGHT': 165, 'WEIGHT': 65}}
```

```
data= 'colour is the visual perception based on the electromagnetic spectrum'
print(data)
```

```
colour is the visual perception based on the electromagnetic spectrum
```

```
from cryptography.fernet import Fernet
key = Fernet.generate_key()
f = Fernet(key)
token = f.encrypt(b"College")
token
b'...'
f.decrypt(token)
b'College'
key = Fernet.generate_key()
cipher_suite = Fernet(key)
plain_text = b"College"
cipher_text= cipher_suite.encrypt(plain_text)
decrypted_text=cipher_suite.decrypt(cipher_text)
print("Original Data:",plain_text)
print("Encrypted Data:",cipher_text)
print("Decrypted Data:",decrypted_text)
```

```
Original Data: b'College'
Encrypted Data: b'gAAAAABnPyQF4o274mJE_gFVK05m7gNzb-LbPhvTTDYgFsCfV520WAAZ4CbX9_RmFg2TZkl0WbzcWiv0XspcURXNTjDxiSlkBQ=='
Decrypted Data: b'College'
```

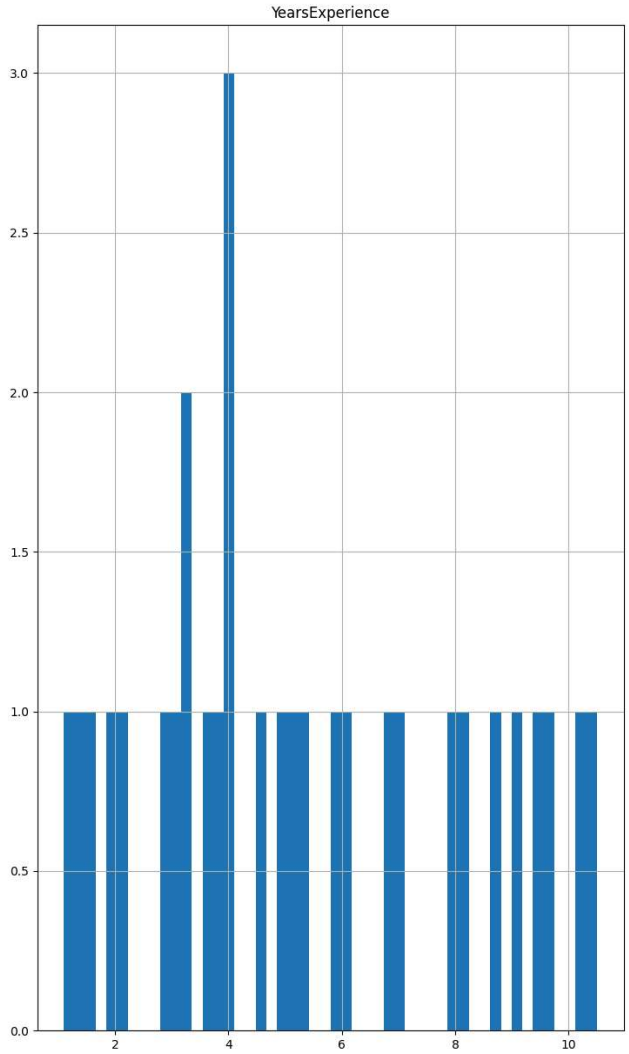
```
import pandas as pd
db=pd.read_csv("Salary_data.csv")
db.head()
```

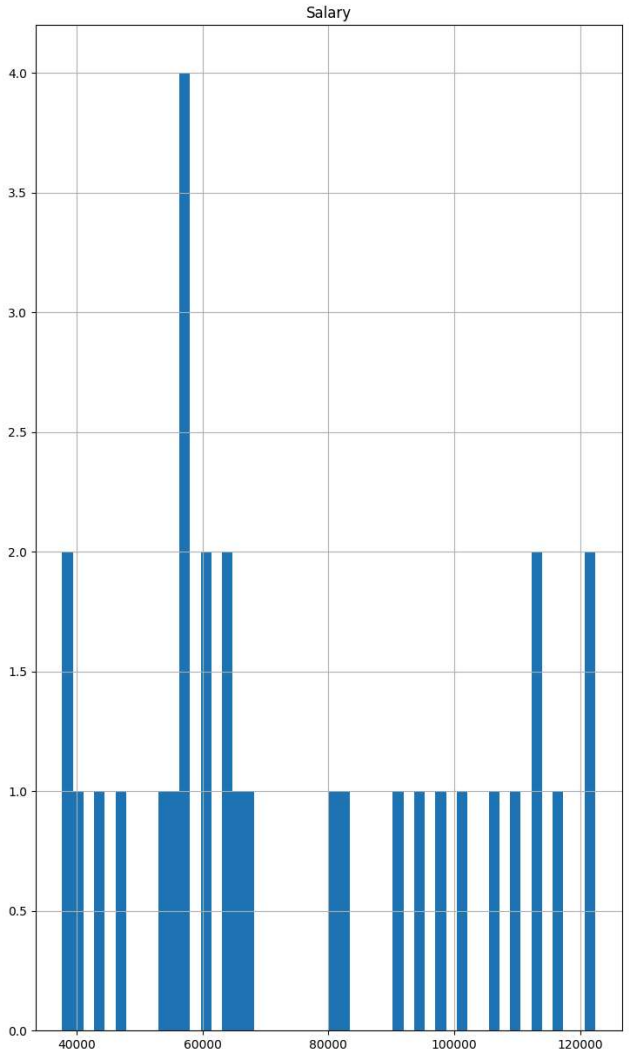
	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891

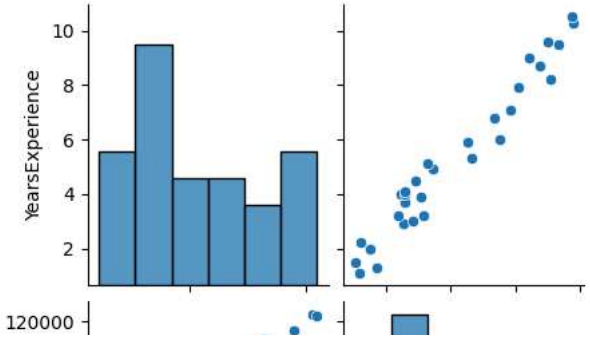
Next steps: Generate code with db | View recommended plots | New interactive sheet

```
print(db.info())
print(db.describe())
import matplotlib.pyplot as plt
import seaborn as sns
db.hist(bins=50,figsize=(20,15))
plt.show()
sns.pairplot(db)
plt.show()
```

```
<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
None
       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
```

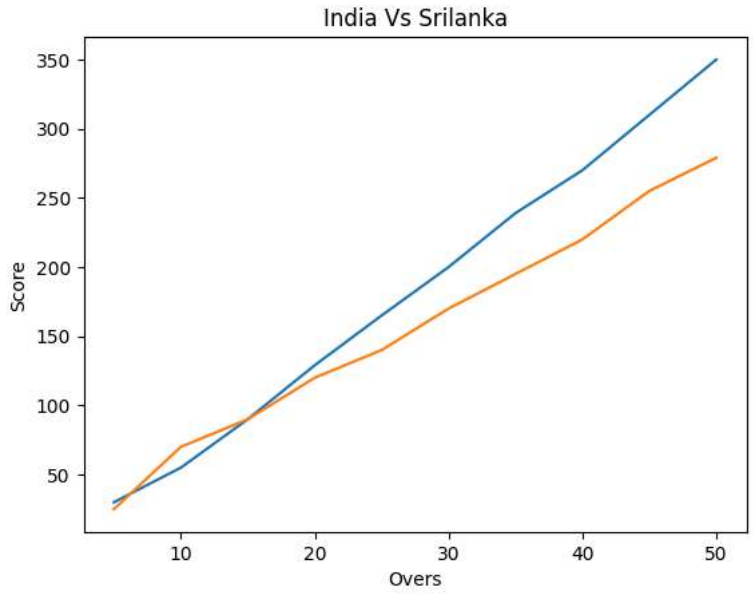






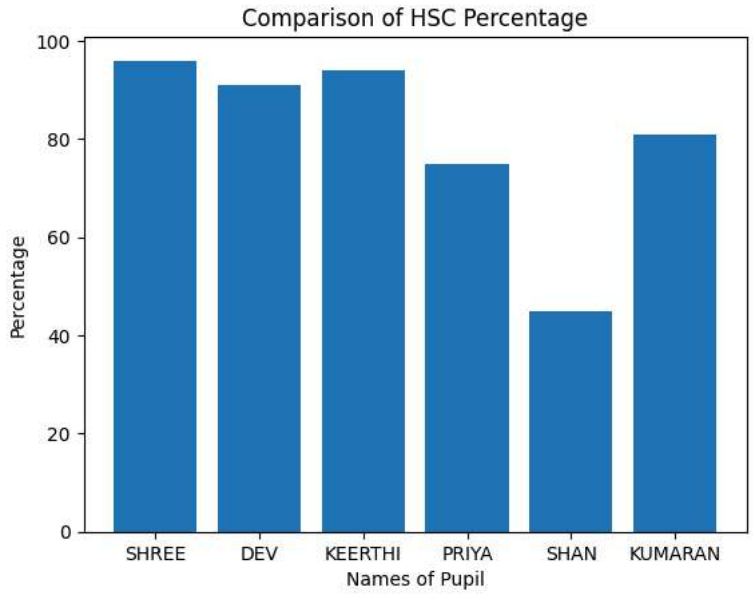
```
import matplotlib.pyplot as cricket
Overs=list(range(5,51,5))
Indian_Score=[30,55,90,129,165,200,239,270,310,350]
Srilankan_Score=[25,70,90,120,140,170,195,220,255,279]
cricket.plot(Overs,Indian_Score)
cricket.plot(Overs,Srilankan_Score)
plt.title('India Vs Srilanka')
plt.xlabel('Overs')
plt.ylabel('Score')
```

```
Text(0, 0.5, 'Score')
```

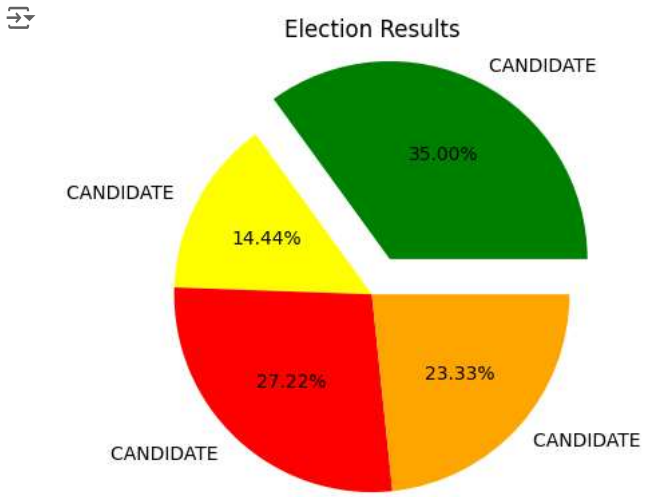


```
import matplotlib.pyplot as plt
import numpy as np
Names = ['SHREE','DEV','KEERTHI','PRIYA','SHAN','KUMARAN']
xaxis = np.arange(len(Names))
Percentage_hsc = [96, 91, 94, 75, 45, 81]
plt.bar(Names,Percentage_hsc)
plt.title('Comparison of HSC Percentage')
plt.xlabel('Names of Pupil')
plt.ylabel('Percentage')
```

```
Text(0, 0.5, 'Percentage')
```



```
import matplotlib.pyplot as election

labels = ['CANDIDATE ', 'CANDIDATE ', 'CANDIDATE ', 'CANDIDATE ']
Votes = [315, 130, 245, 210]
colors = ['green', 'yellow', 'red', 'orange']
explode = (0.2, 0, 0, 0)
# Plotting the pie chart
election.pie(Votes, labels=labels, colors=colors, explode=explode, autopct='%0.2f%%')
election.title('Election Results')
election.show()
```



```
import numpy as np
import pandas as pd
list=[[1,'Smith',50000],[2,'Jones',60000]]
df=pd.DataFrame(list)
df
```

	0	1	2
0	1	Smith	50000
1	2	Jones	60000

Next steps: Generate code with df | View recommended plots | New interactive sheet

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

	Empd	Name	Salary
0	1	Smith	50000
1	2	Jones	60000

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
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
```

```
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
```

```
df.head()
```

	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

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.tail()
```

	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

```
df.describe()
```

	R&D Spend	Administration	Marketing Spend	Profit
count	50.000000	50.000000	50.000000	50.000000
mean	73721.615600	121344.639600	211025.097800	112012.639200
std	45902.256482	28017.802755	122290.310726	40306.180338
min	0.000000	51283.140000	0.000000	14681.400000
25%	39936.370000	103730.875000	129300.132500	90138.902500
50%	73051.080000	122699.795000	212716.240000	107978.190000
75%	101602.800000	144842.180000	299469.085000	139765.977500
max	165349.200000	182645.560000	471784.100000	192261.830000

```
import numpy as np
array=np.random.randint(1,100,16) # randomly generate 16 numbers between 1 to 100
array
```

```
array([41, 22, 30, 37, 67, 46, 19,  1, 97, 29, 50, 19,  9, 32, 61, 54])
```

```
array.mean()
```

```
36.625
```

```
np.percentile(array,25)
```

```
13.5
```

```
np.percentile(array,50)
```

```
27.5
```

```
np.percentile(array,75)
```

```
58.5
```

```
np.percentile(array,100)
```

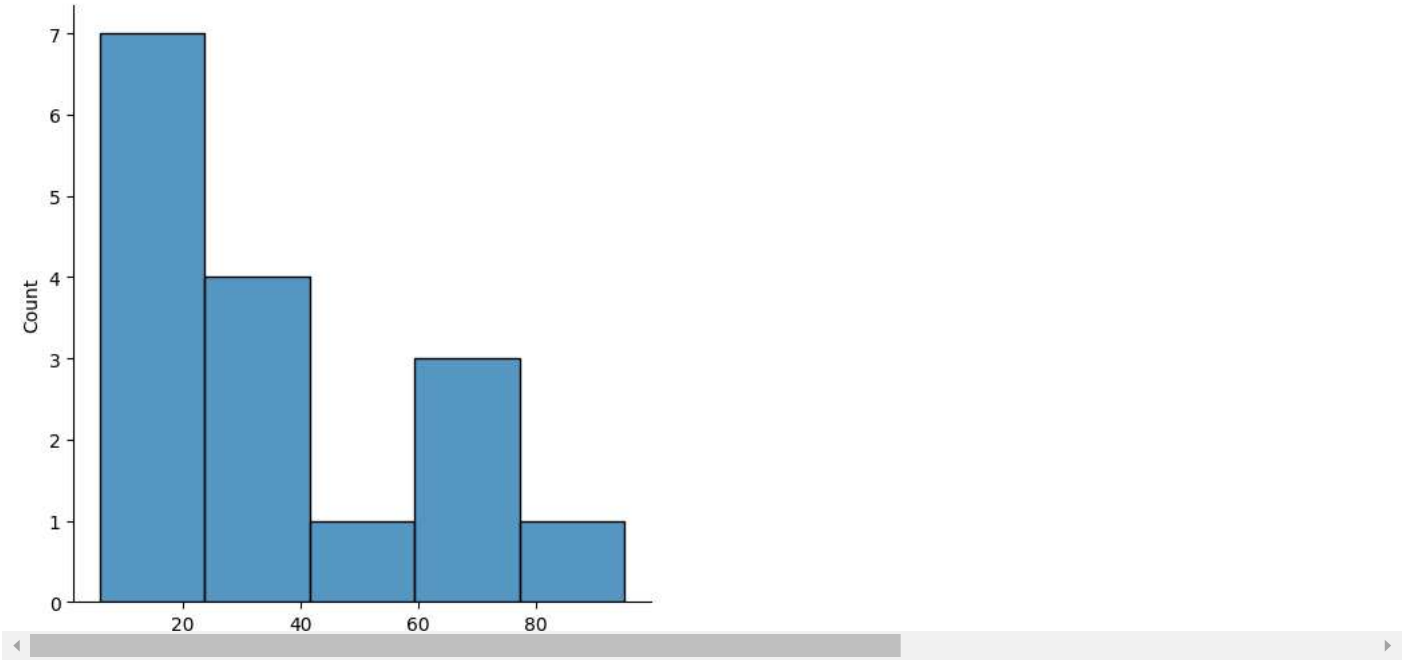
```
95.0
```

```
#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
```

```
(-54.0, 126.0)
```

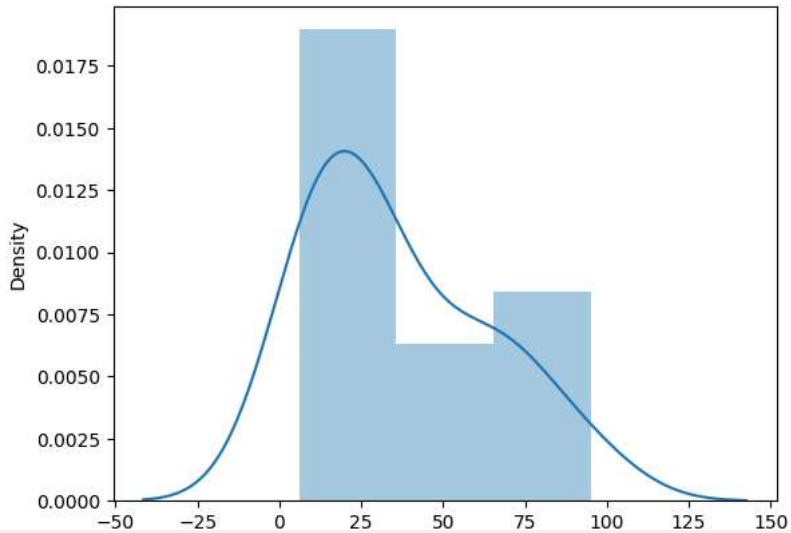
```
import seaborn as sns
%matplotlib inline
sns.displot(array)
```

```
<seaborn.axisgrid.FacetGrid at 0x1dbcd642bc8>
```



```
sns.distplot(array)
```

```
c:\users\asus\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.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

  """Entry point for launching an IPython kernel.
<AxesSubplot:ylabel='Density'>
```

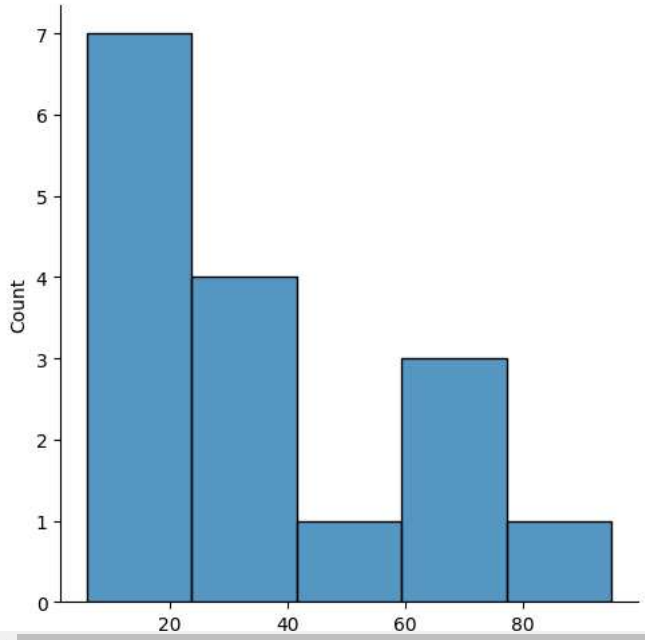



```
new_array=array[(array>lr) & (array<ur)]
new_array
```

```
array([ 6, 66,  9, 37, 29, 40, 12, 23, 20, 77, 26, 95, 56, 14,  9, 67])
```

```
sns.displot(new_array)
```

```
<seaborn.axisgrid.FacetGrid at 0x1dbcfbd1f08>
```




```
lr1,ur1=outDetection(new_array)
lr1,ur1
```

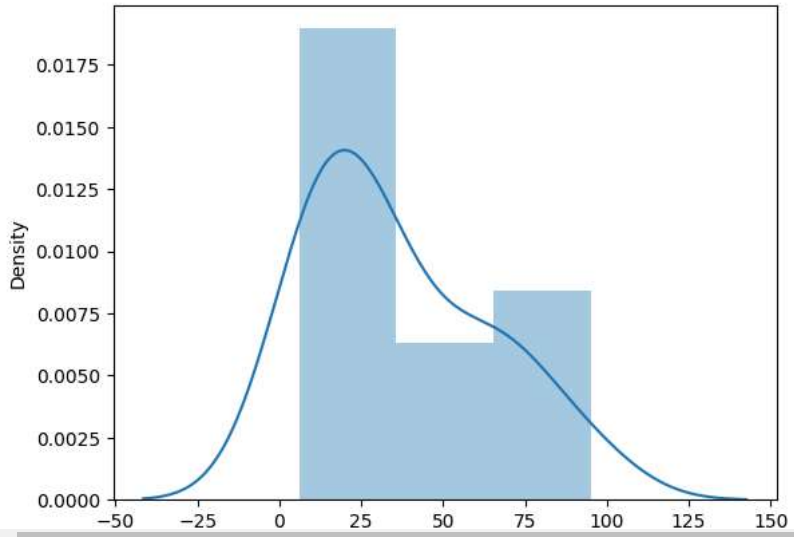
```
(-54.0, 126.0)
```

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

```
array([ 6, 66,  9, 37, 29, 40, 12, 23, 20, 77, 26, 95, 56, 14,  9, 67])
```

```
sns.distplot(final_array)
```

```
c:\users\asus\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.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

  """Entry point for launching an IPython kernel.
<AxesSubplot:ylabel='Density'>
```



Start coding or generate with AI.

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

	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

```
df.duplicated()
```

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

```
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
```

```
df.drop_duplicates(inplace=True)
df
```

	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

```
len(df)
```

```
10
```

```
index=np.array(list(range(0,len(df))))
df.set_index(index,inplace=True)
index
```

```
df
```

	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

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

	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

```
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\asus\appdata\local\programs\python\python37\lib\site-packages\pandas\core\indexing.py:1732: 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-cc
  self._setitem_single_block(indexer, value, name)
```

	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

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

```
c:\users\asus\appdata\local\programs\python\python37\lib\site-packages\pandas\core\indexing.py:1732: 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-cc
  self._setitem_single_block(indexer, value, name)
```

	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

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

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

```
df.Hotel.unique()
```

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

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

```
<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>
```

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

```
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
```

	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	Veg	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	Veg	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

Start coding or generate with AI.

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

	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

```
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
```

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

```
0    France
dtype: object
```

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

```
'France'
```

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

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

```
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
```

	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

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

	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

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

```
updated_dataset
```

	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

```
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
```

```
updated_dataset
```

	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

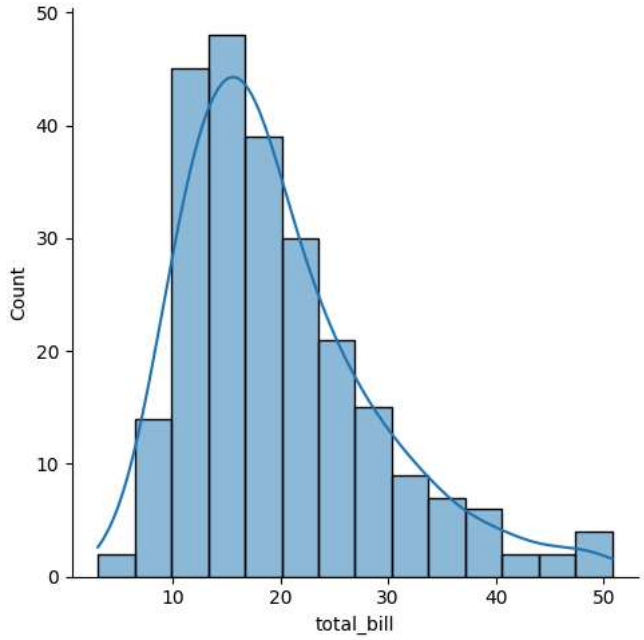
```
Start coding or generate with AI.
```

```
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()
```

	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

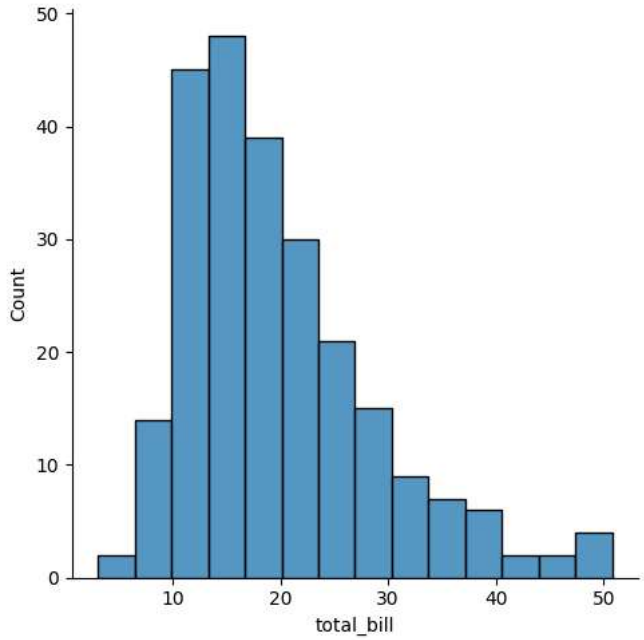
```
sns.displot(tips.total_bill,kde=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x132efab8348>
```



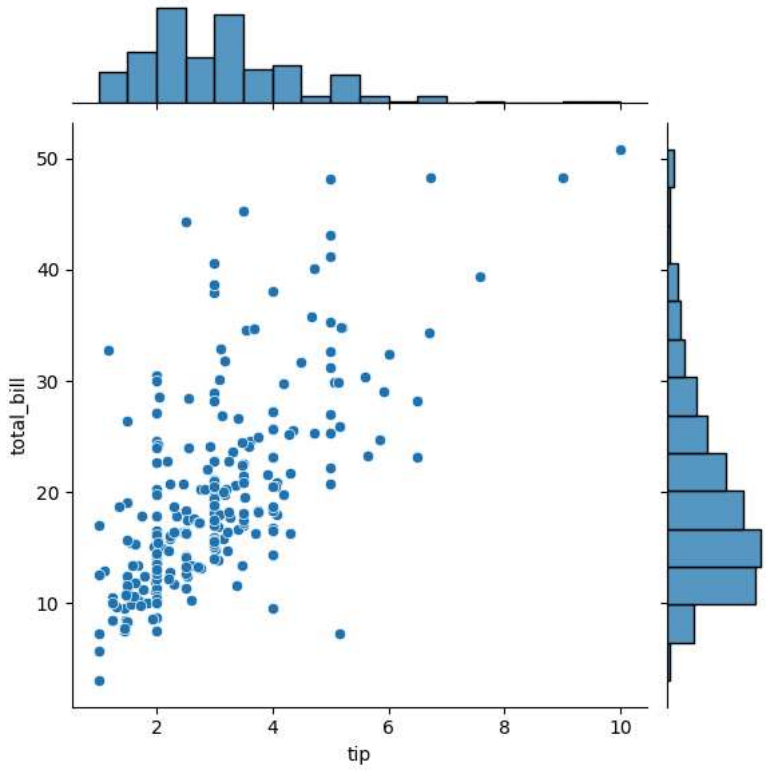

```
sns.displot(tips.total_bill,kde=False)
```

```
<seaborn.axisgrid.FacetGrid at 0x132f1e88148>
```



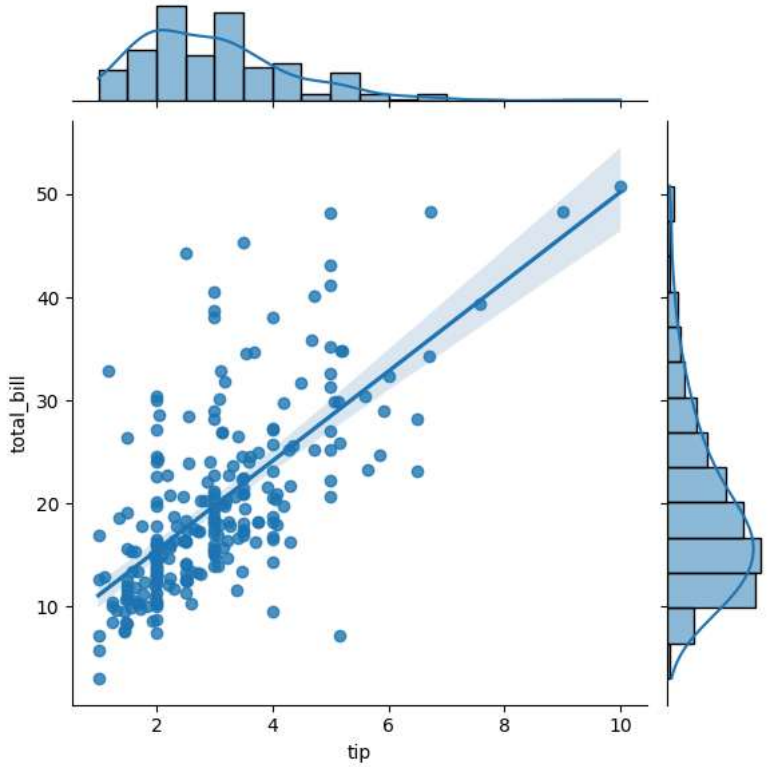

```
sns.jointplot(x=tips.tip,y=tips.total_bill)
```

```
<seaborn.axisgrid.JointGrid at 0x132f1f636c8>
```



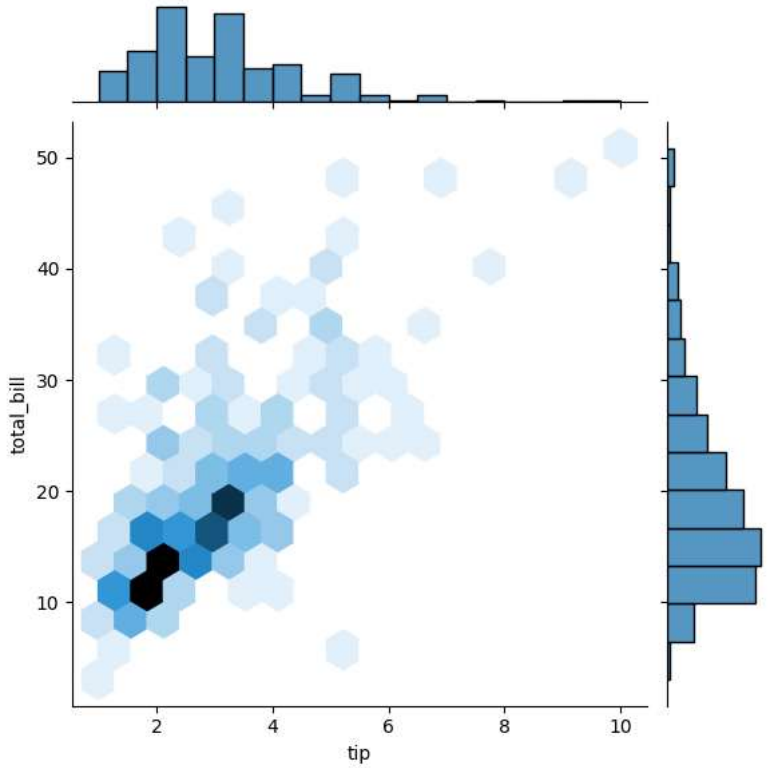

```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="reg")
```

```
<seaborn.axisgrid.JointGrid at 0x132f2224e88>
```



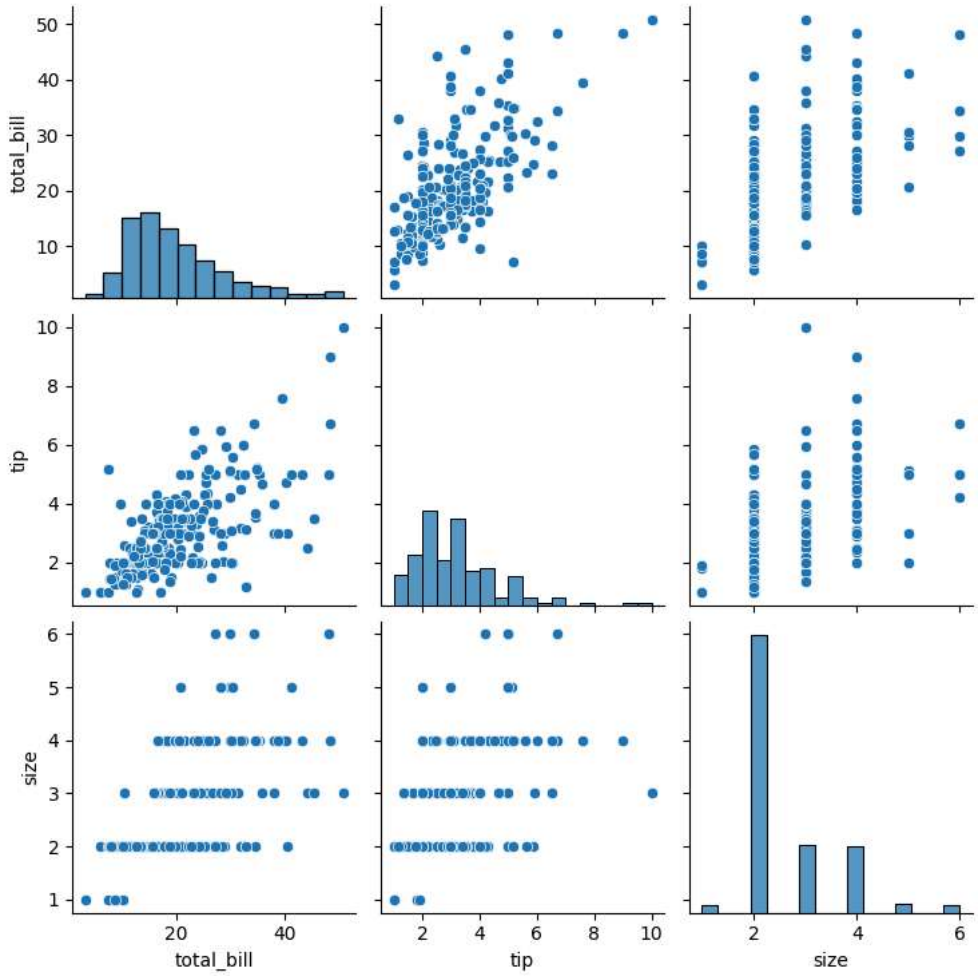

```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="hex")
```

```
<seaborn.axisgrid.JointGrid at 0x132f26f5d08>
```




```
sns.pairplot(tips)
```

```
<seaborn.axisgrid.PairGrid at 0x132f26f7708>
```

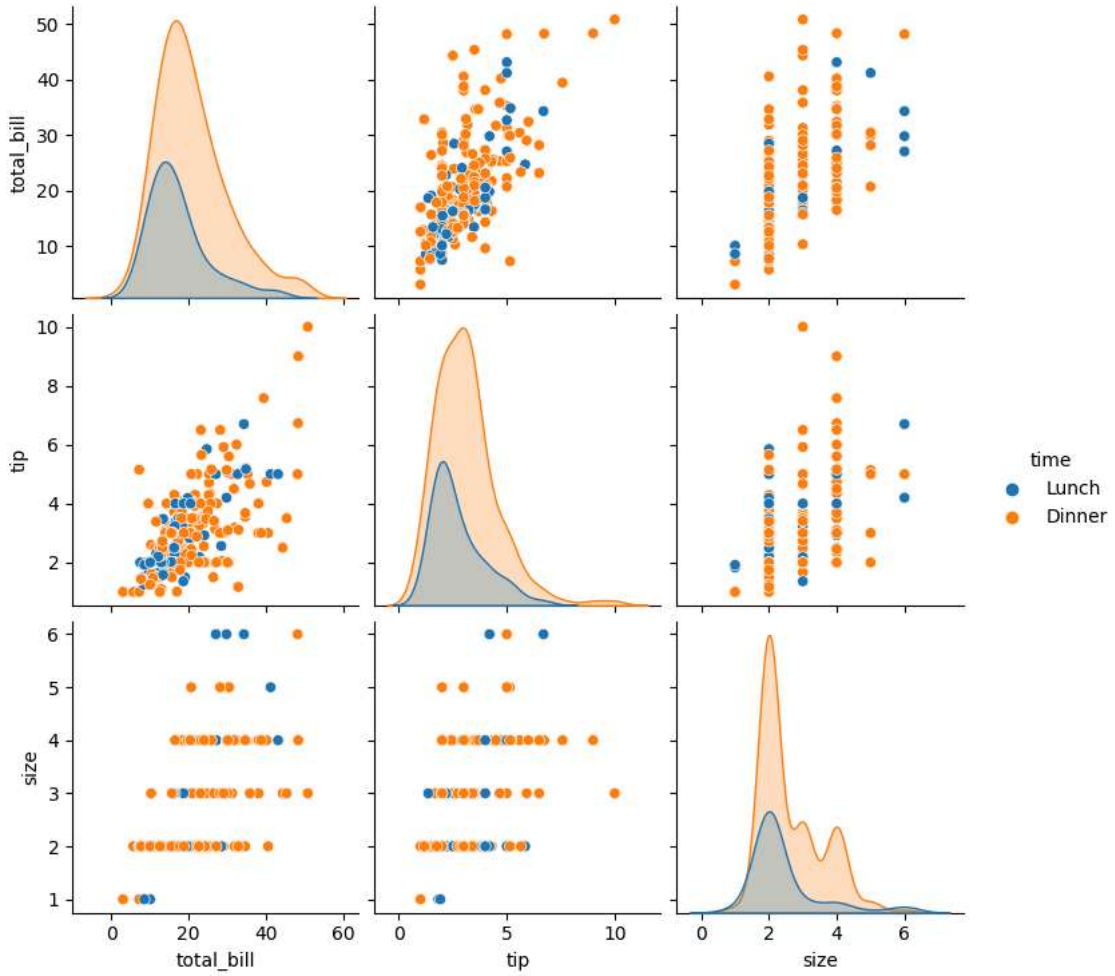



```
tips.time.value_counts()
```

```
Dinner    176
Lunch      68
Name: time, dtype: int64
```

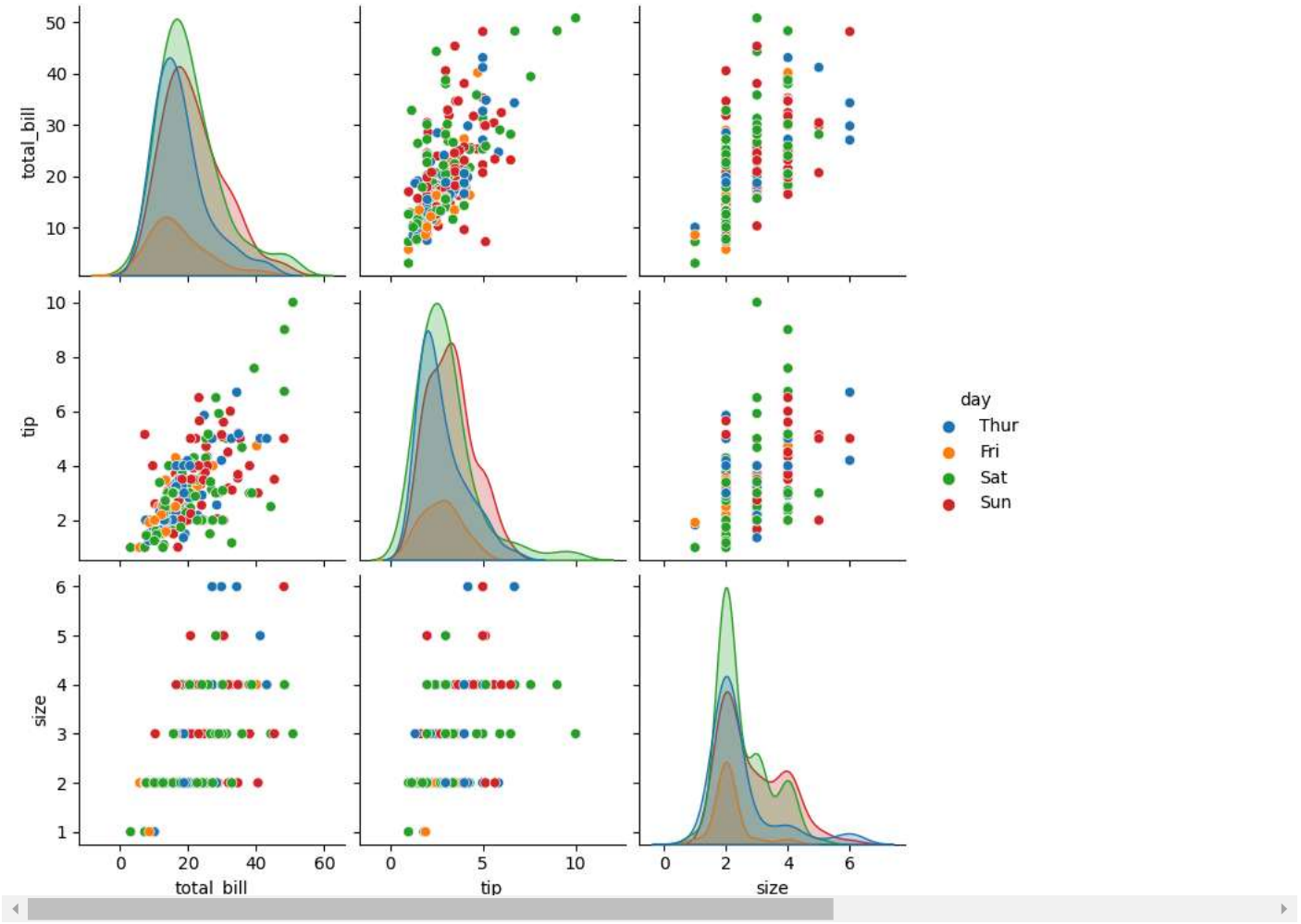
```
sns.pairplot(tips,hue='time')
```

```
<seaborn.axisgrid.PairGrid at 0x132f3420d88>
```



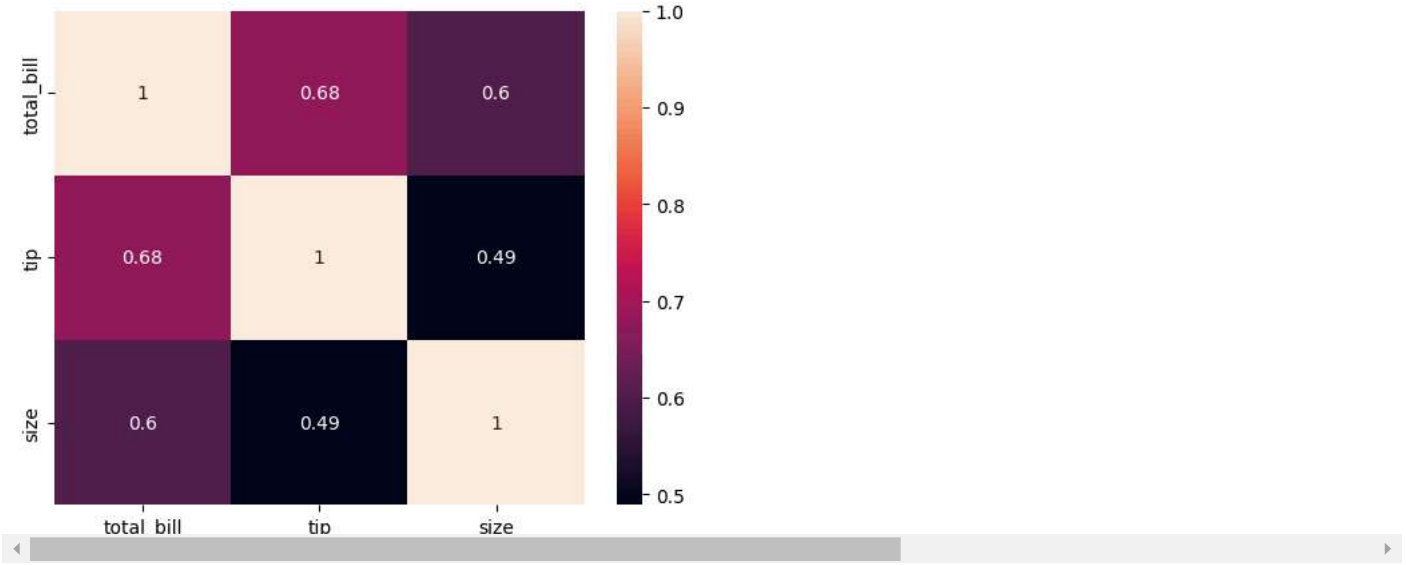

```
sns.pairplot(tips,hue='day')
```

```
<seaborn.axisgrid.PairGrid at 0x132f4c14088>
```



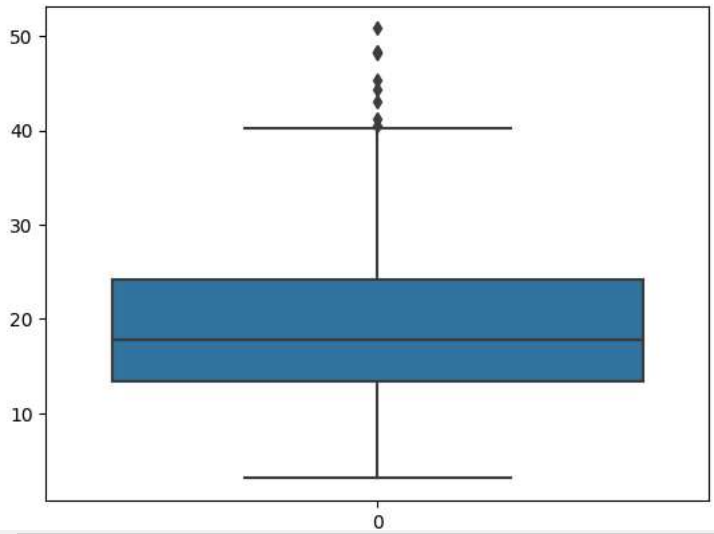

```
sns.heatmap(tips.select_dtypes(include=['number']).corr(), annot=True)
```

```
<AxesSubplot:>
```



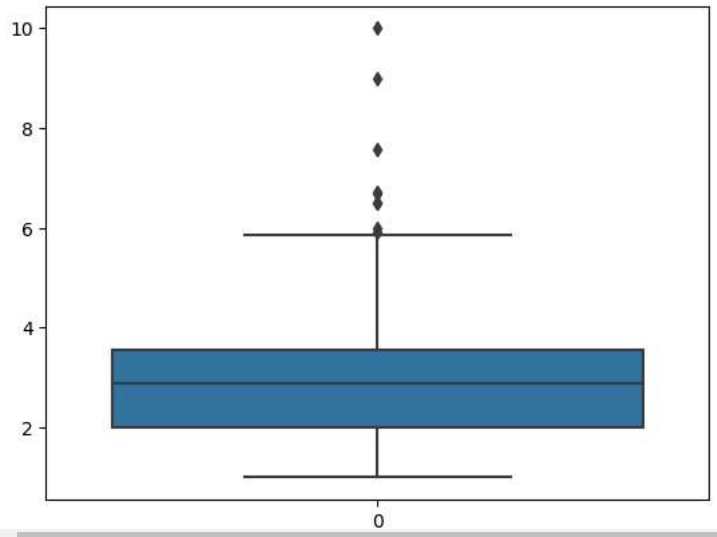

```
sns.boxplot(tips.total_bill)
```

```
<AxesSubplot:>
```



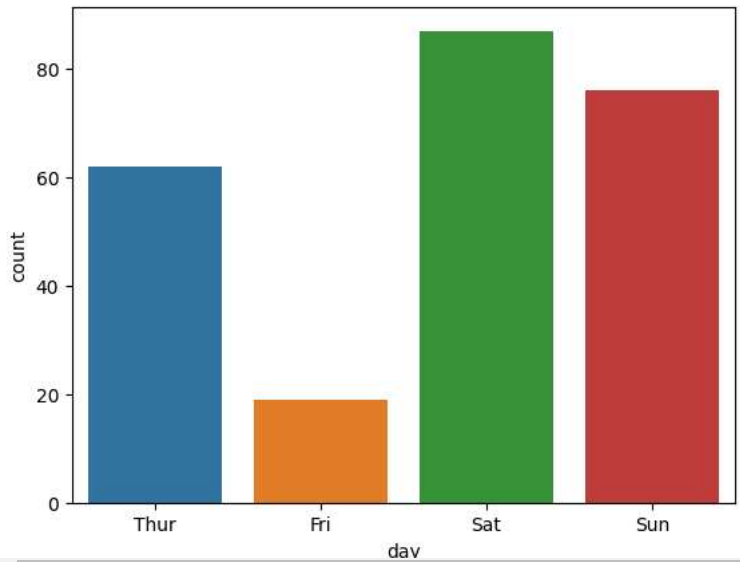

```
sns.boxplot(tips.tip)
```

```
<AxesSubplot:>
```



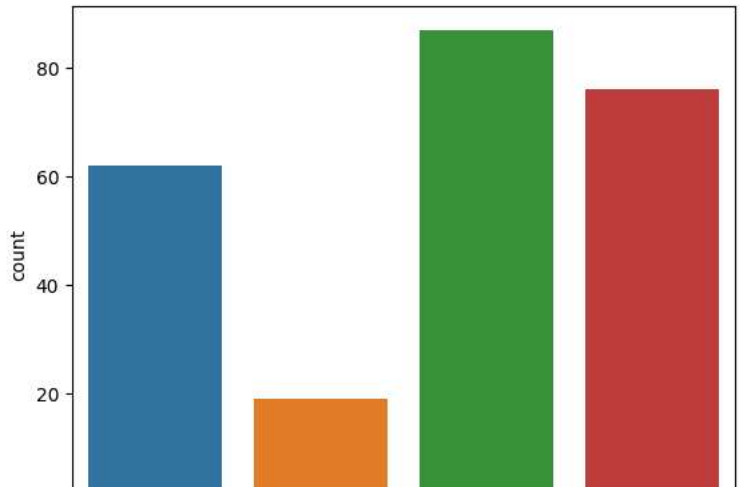

```
sns.countplot(x='day', data=tips)
```

```
<AxesSubplot:xlabel='day', ylabel='count'>
```

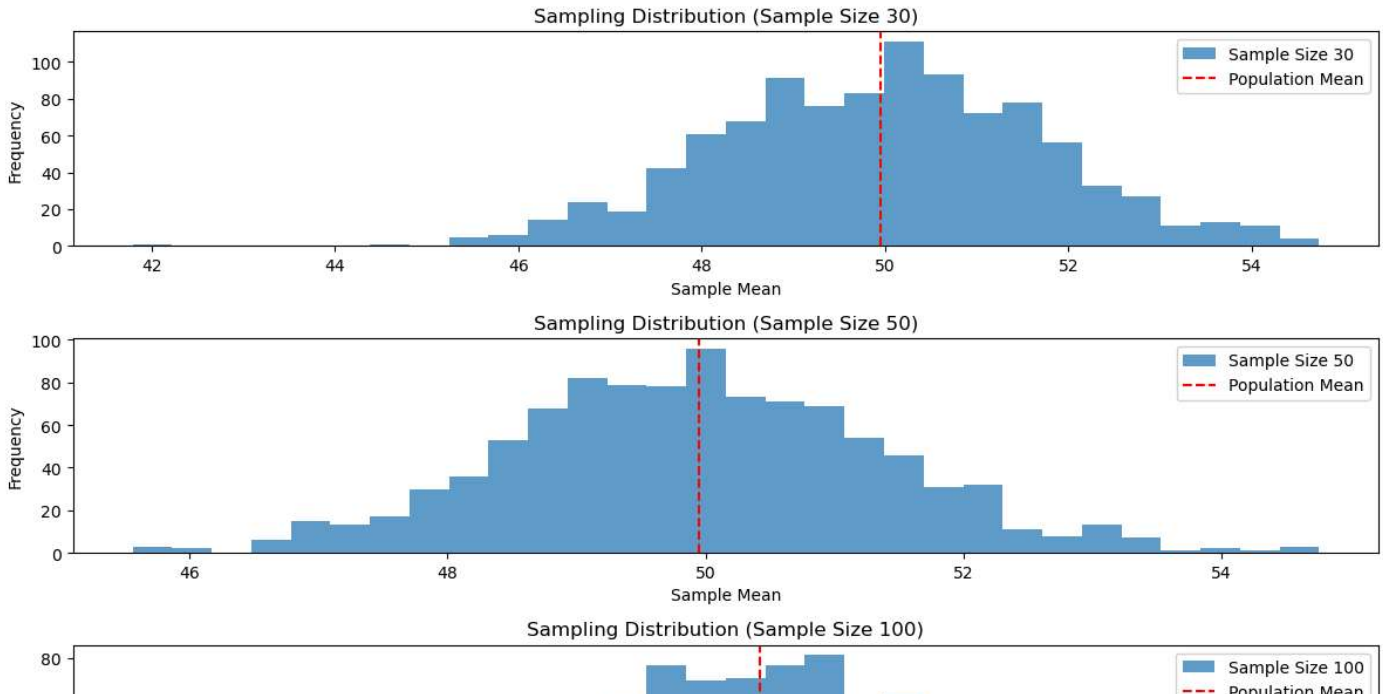


```
sns.countplot(x='day', data=tips)
```

```
<AxesSubplot:xlabel='day', ylabel='count'>
```



```
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]
num_samples = 1000

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()
```


Sampling Distribution (Sample Size 30)
Sample Size 30
Population Mean
Frequency
Sample Mean
0
20
40
60
80
100
42
44
46
48
50
52
54

Sampling Distribution (Sample Size 50)
Sample Size 50
Population Mean
Frequency
Sample Mean
0
20
40
60
80
100
46
48
50
52
54

Sampling Distribution (Sample Size 100)
Sample Size 100
80

```
import math
import numpy as np
from statsmodels.stats.weightstats import ztest
from scipy.stats import norm
sample_marks = [650,730,510,670,480,800,690,530,590,620,710,670,640,780,650,490,800,600,510,700]
# Method 1 : Using Z-score
sample_mean = np.mean(sample_marks)
sample_size = np.count_nonzero(sample_marks)
population_mean = 600
population_std = 100
alpha = 0.05
z_score = (sample_mean-population_mean)/(population_std/math.sqrt(sample_size))
critical_value = 1.645 # from z table
if(z_score<critical_value):
    print('Null hypothesis is accepted!')
else:
    print('Null hypothesis is rejected. \nAlternate hypothesis is accepted!')

# Method 2: Using built in function of ztest
ztest_score, pval = ztest(sample_marks,value=population_mean,alternative='larger')
print('Z-test Score:',ztest_score,'\nP-value:',pval)
if(pval>alpha):
     print('Null hypothesis is accepted!')
else:
    print('Null hypothesis is rejected. \nAlternate hypothesis is accepted!')

# Method 3: Creating a function
def ztest(x,mu,sigma,n):
    deno = sigma/math.sqrt(n)
    z = (x-mu)/deno
    p = 2*(1-norm.cdf(abs(z)))
    return z,p
s_mean = np.mean(sample_marks)
p_mean = 600
p_std = 100
s_size = np.count_nonzero(sample_marks)
ztest(s_mean,p_mean,p_std,s_size)
ztest(641,600,100,20)
```

```
Null hypothesis is rejected.
Alternate hypothesis is accepted!
Z-test Score: 1.831744911595958
P-value: 0.03349471703839336
Null hypothesis is rejected.
Alternate hypothesis is accepted!
(1.8335757415498277, 0.06671699590108493)
```

Start coding or generate with AI.

```
import numpy as np
from scipy import stats
# Given student scores
student_scores = np.array([72, 89, 65, 73, 79, 84, 63, 76, 85, 75])
# Hypothesized population mean
mu = 70
# Perform one-sample t-test
t_stat, p_value = stats.ttest_1samp(student_scores, mu)
print("T statistic:", t_stat)
print("P-value:", p_value)
# Setting significance level
alpha = 0.05
# Interpret the results
if p_value < alpha:
    print("Reject the null hypothesis; there is a significant difference between the sample mean and the hypothesized population mean.")
else:
    print("Fail to reject the null hypothesis; there is no significant difference between the sample mean and the hypothesized population mea
```

```
T statistic: 2.2894683580127317
P-value: 0.047816221110566944
Reject the null hypothesis; there is a significant difference between the sample mean and the hypothesized population mean.
```

Start coding or generate with AI.

```
import numpy as np
from scipy.stats import f_oneway
# Sample data: Exam scores for three teaching methods
np.random.seed(42)
method_A_scores = np.random.normal(loc=80, scale=10, size=30)
method_B_scores = np.random.normal(loc=85, scale=10, size=30)
method_C_scores = np.random.normal(loc=90, scale=10, size=30)

# Perform one-way ANOVA
f_statistic, p_value = f_oneway(method_A_scores, method_B_scores, method_C_scores)

print("F-Statistic:", f_statistic)
print("P-Value:", p_value)
```

```
F-Statistic: 12.20952551797281
P-Value: 2.1200748140507065e-05
```

Start coding or generate with AI.

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

	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

```
df.head()
```

	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

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

```
SimpleImputer()
```

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

```
SimpleImputer()
```

```
SimpleImputer()
```

```
SimpleImputer()
```

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

```
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)
```

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

```
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.],
       [0., 1., 0.],
       [1., 0., 0.]])
```

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

```
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)
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit(final_set)
feat_standard_scaler=sc.transform(final_set)
feat_standard_scaler
```

```
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]])
```

```
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
```

```
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]])
```

Start coding or generate with AI.

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

	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

```
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
```

```
df.dropna(inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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: 720.0 bytes
```

```
df.describe()
```

	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

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

```
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)
```

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
```

```
LinearRegression()
```

```
model.score(x_train,y_train)
```

```
0.9645401573418146
```

```
model.score(x_test,y_test)
```

```
0.9024461774180497
```

```
model.coef_
```

```
array([[9423.81532303]])
```

```
model.intercept_
```

```
array([25321.58301178])
```

```
import pickle
pickle.dump(model,open('SalaryPred.model','wb'))
```

```
model=pickle.load(open('SalaryPred.model','rb'))
```

```
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)
```

```
Enter Years of Experience: 44
```

```
print("Estimated Salary for {} years of experience is {}: ".format(yr_of_exp, Salary))
```

```
Estimated Salary for 44.0 years of experience is [[439969.45722514]]:
```

Start coding or generate with AI.

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

	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

```
df.head()
```

	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

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

```
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],
```

label

```
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, 1, 0, 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)
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
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 0.6875 Train0.63125 Random State 3
Test 0.7375 Train0.61875 Random State 4
Test 0.6625 Train0.6375 Random State 5
Test 0.65 Train0.640625 Random State 6
Test 0.675 Train0.634375 Random State 7
Test 0.675 Train0.634375 Random State 8
Test 0.65 Train0.640625 Random State 10
Test 0.6625 Train0.6375 Random State 11
Test 0.7125 Train0.625 Random State 13
Test 0.675 Train0.634375 Random State 16
Test 0.7 Train0.628125 Random State 17
Test 0.7 Train0.628125 Random State 21
Test 0.65 Train0.640625 Random State 24
Test 0.6625 Train0.6375 Random State 25
Test 0.75 Train0.615625 Random State 26
Test 0.675 Train0.634375 Random State 27
Test 0.7 Train0.628125 Random State 28
Test 0.6875 Train0.63125 Random State 29
Test 0.6875 Train0.63125 Random State 31
Test 0.6625 Train0.6375 Random State 37
Test 0.7 Train0.628125 Random State 39
```

```
Test 0.7 Train0.628125 Random State 40
Test 0.65 Train0.640625 Random State 42
Test 0.725 Train0.621875 Random State 46
Test 0.65 Train0.640625 Random State 48
Test 0.675 Train0.634375 Random State 50
Test 0.65 Train0.640625 Random State 51
Test 0.65 Train0.640625 Random State 54
Test 0.7 Train0.634375 Random State 55
Test 0.65 Train0.640625 Random State 56
Test 0.6625 Train0.6375 Random State 58
Test 0.6875 Train0.63125 Random State 59
Test 0.7 Train0.628125 Random State 60
Test 0.6625 Train0.6375 Random State 62
Test 0.6875 Train0.63125 Random State 63
Test 0.65 Train0.640625 Random State 66
Test 0.7 Train0.628125 Random State 70
Test 0.65 Train0.640625 Random State 74
Test 0.65 Train0.640625 Random State 75
Test 0.6875 Train0.63125 Random State 76
Test 0.6875 Train0.63125 Random State 80
Test 0.675 Train0.634375 Random State 81
Test 0.875 Train0.8375 Random State 82
Test 0.7 Train0.628125 Random State 83
Test 0.675 Train0.634375 Random State 84
Test 0.675 Train0.634375 Random State 86
Test 0.65 Train0.640625 Random State 87
Test 0.675 Train0.634375 Random State 90
Test 0.65 Train0.640625 Random State 91
Test 0.7 Train0.628125 Random State 93
Test 0.7375 Train0.61875 Random State 94
Test 0.65 Train0.640625 Random State 97
Test 0.7 Train0.628125 Random State 99
Test 0.675 Train0.634375 Random State 101
Test 0.6625 Train0.6375 Random State 102
Test 0.725 Train0.621875 Random State 103
Test 0.65 Train0.640625 Random State 106
```

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

```
LogisticRegression()
```

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

```
0.625
0.7125
```

```
from sklearn.metrics import classification_report
print(classification_report(label, finalModel.predict(features), zero_division=1))
```

```
              precision    recall  f1-score   support

           0       0.64      1.00      0.78       257
           1       1.00      0.00      0.00       143

    accuracy                           0.64       400
   macro avg       0.82      0.50      0.39       400
weighted avg       0.77      0.64      0.50       400
```

Start coding or generate with AI.