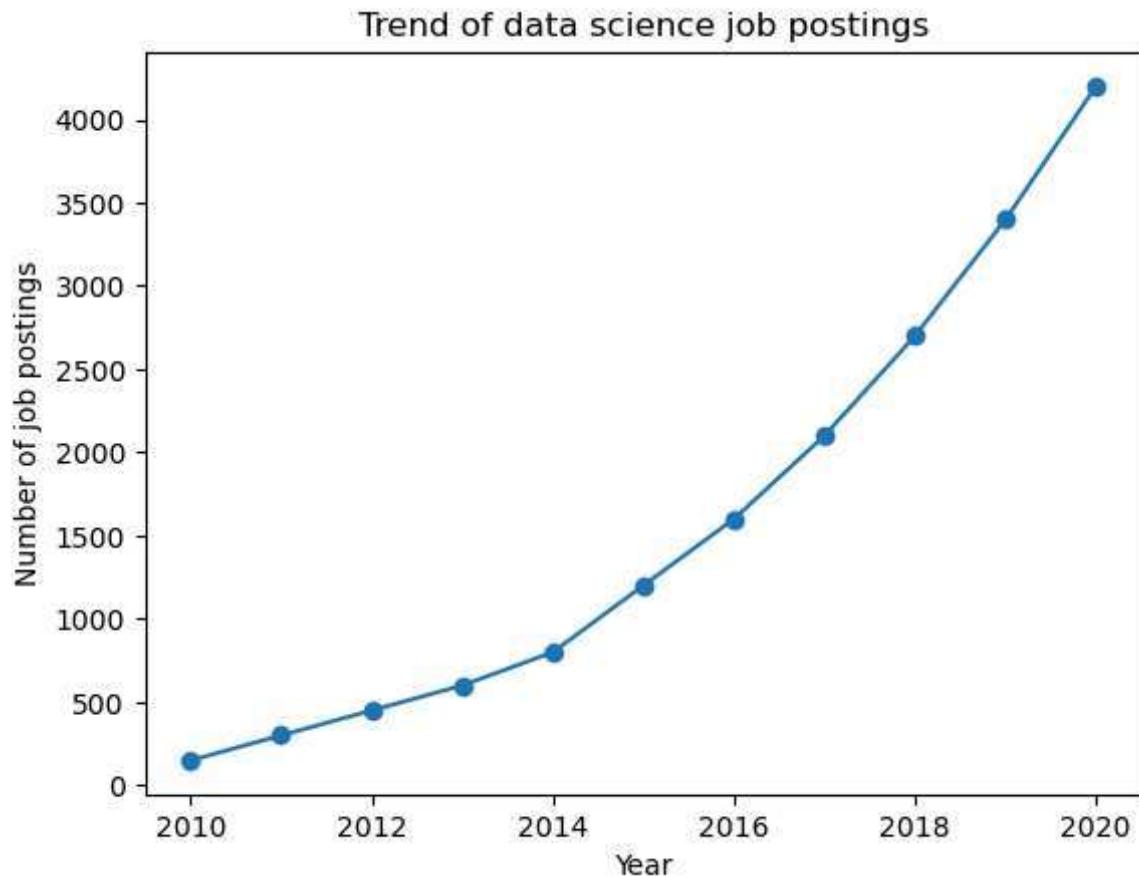


Fundamentals of Data Science

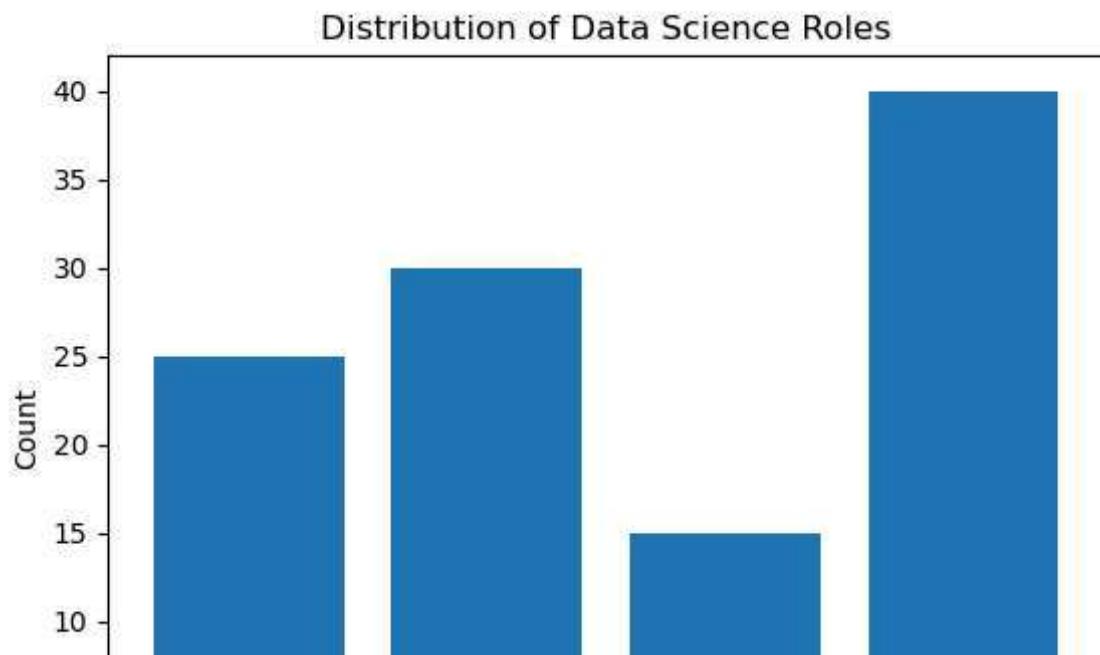
Exp 1

```
In [2]: import pandas as pd
import matplotlib.pyplot as plt
data = {'Year':list(range(2010,2021)),
        'job posting':[150,300,450,600,800,1200,1600,2100,2700,3400,4200]}

df=pd.DataFrame(data)
plt.plot(df['Year'],df['job posting'],marker='o')
plt.title('Trend of data science job postings')
plt.xlabel('Year')
plt.ylabel('Number of job postings')
plt.show()
```

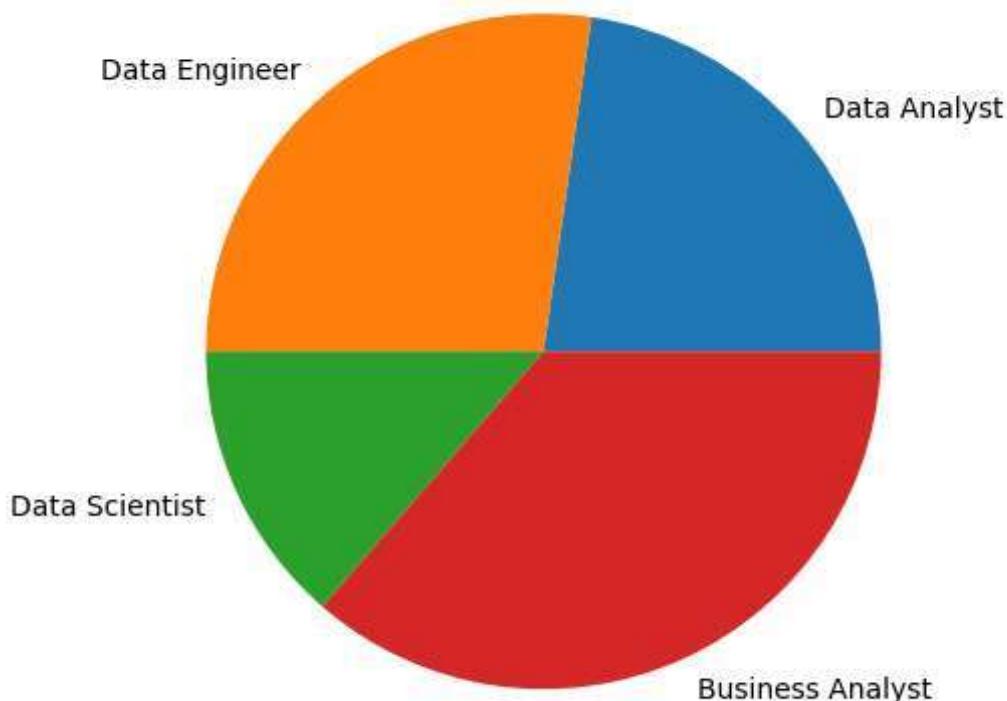


```
In [15]: import pandas as pd
import matplotlib.pyplot as plt
roles=['Data Analyst','Data Engineer','Data Scientist','Business Analyst']
count=[25,30,15,40]
plt.bar(roles,count)
plt.title('Distribution of Data Science Roles')
plt.xlabel('Roles')
plt.ylabel('Count')
plt.show()
```



```
In [11]: import matplotlib.pyplot as plt

roles = ['Data Analyst', 'Data Engineer', 'Data Scientist', 'Business Analyst']
count = [25, 30, 15, 40]
plt.pie(count, labels=roles)
plt.axis('equal')
plt.show()
```



```
In [12]: import pandas as pd
structured_data = pd.DataFrame({
    'ID':[1,2,3],
    'Name':['Divya','Dinisha','Dhivya'],
    'Age':[18,18,18],
    'Marks':[81,80,82],})
print("Structured data:")
print(structured_data)
```

Structured data:

	ID	Name	Age	Marks
0	1	Divya	18	81
1	2	Dinisha	18	80
2	3	Dhivya	18	82

```
In [25]: import pandas as pd  
unstructured_data="This is an example of unstructured data. It can be a piece of text, an image, or a video file."  
print("Unstructured Data:")  
print(unstructured_data)
```

Unstructured Data:

This is an example of unstructured data. It can be a piece of text, an image, or a video file.

```
In [24]: semi_structured={'ID':1,'NAme':'Divya','Attributes':{'Age':18,'Marks':81}}  
print("Unstructured Data:")  
print(semi_structured)
```

Unstructured Data:

```
{'ID': 1, 'NAme': 'Divya', 'Attributes': {'Age': 18, 'Marks': 81}}
```

```
In [14]: from cryptography.fernet import Fernet  
key=Fernet.generate_key()  
f=Fernet(key)  
token=f.encrypt(b"My name is Divyadharshini K")  
token  
b'...'  
f.decrypt(token)  
b'My name is Divyadharshini'  
key=Fernet.generate_key()  
cipher_suite=Fernet(key)  
plain_text=b'My name is Divyadharshini'  
cipher_text=cipher_suite.encrypt(plain_text)  
decrypt_text=cipher_suite.decrypt(cipher_text)  
print("Original Data",plain_text)  
print("Encrypted Data",cipher_text)  
print("Decrypted Data",decrypt_text)
```

Original Data b'My name is Divyadharshini'

Encrypted Data b'gAAAAABmwWr_Du53R9UajjHu5i9iaucNWFxkW9pB-eN2hH1_Wtg4NhboWo1oP
6GExQ0u0jaW24i7sm2zTxrPY3KUliZ2D_sx0Z5K4WjD1Xxv3mCM881caaY='

Decrypted Data b'My name is Divyadharshini'

```
In [ ]:
```

```
In [ ]:
```

```
In [3]: import pandas as pd  
db = pd.read_csv("Sales_Transactions_Dataset_Weekly.csv")
```

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

Out[4]:

	Product_Code	W0	W1	W2	W3	W4	W5	W6	W7	W8	...	Normalized 42	Normalized 43	Norma
0	P1	11	12	10	8	13	12	14	21	6	...	0.06	0.22	
1	P2	7	6	3	2	7	1	6	3	3	...	0.20	0.40	
2	P3	7	11	8	9	10	8	7	13	12	...	0.27	1.00	
3	P4	12	8	13	5	9	6	9	13	13	...	0.41	0.47	
4	P5	8	5	13	11	6	7	9	14	9	...	0.27	0.53	

5 rows × 107 columns

```
In [5]: db.tail()
```

Out[5]:

	Product_Code	W0	W1	W2	W3	W4	W5	W6	W7	W8	...	Normalized 42	Normalized 43	Norr
806	P815	0	0	1	0	0	2	1	0	0	...	0.00	0.33	
807	P816	0	1	0	0	1	2	2	6	0	...	0.43	0.43	
808	P817	1	0	0	0	1	1	2	1	1	...	0.50	0.00	
809	P818	0	0	0	1	0	0	0	0	1	...	0.00	0.00	
810	P819	0	1	0	0	0	0	0	0	0	...	0.00	0.00	

5 rows × 107 columns

```
In [6]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df=pd.read_csv('Sales_Transactions_Dataset_Weekly.csv')
print(df.head())
print(df.isnull().sum())
print(df.describe)
```

```

Product_Code W0 W1 W2 W3 W4 W5 W6 W7 W8 ... Normalized 42 \
0 P1 11 12 10 8 13 12 14 21 6 ... 0.06
1 P2 7 6 3 2 7 1 6 3 3 ... 0.20
2 P3 7 11 8 9 10 8 7 13 12 ... 0.27
3 P4 12 8 13 5 9 6 9 13 13 ... 0.41
4 P5 8 5 13 11 6 7 9 14 9 ... 0.27

Normalized 43 Normalized 44 Normalized 45 Normalized 46 Normalized 47 \
\
0 0.22 0.28 0.39 0.50 0.00
1 0.40 0.50 0.10 0.10 0.40
2 1.00 0.18 0.18 0.36 0.45
3 0.47 0.06 0.12 0.24 0.35
4 0.53 0.27 0.60 0.20 0.20

Normalized 48 Normalized 49 Normalized 50 Normalized 51
0 0.22 0.17 0.11 0.39
1 0.50 0.10 0.60 0.00
2 1.00 0.45 0.45 0.36
3 0.71 0.35 0.29 0.35
4 0.13 0.53 0.33 0.40

[5 rows x 107 columns]
Product_Code 0
W0 0
W1 0
W2 0
W3 0
..
Normalized 47 0
Normalized 48 0
Normalized 49 0
Normalized 50 0
Normalized 51 0
Length: 107, dtype: int64
<bound method NDFrame.describe of
  Product_Code W0 W1 W2 W3 W4 W5 W
  6 W7 W8 ... Normalized 42 \
0 P1 11 12 10 8 13 12 14 21 6 ... 0.06
1 P2 7 6 3 2 7 1 6 3 3 ... 0.20
2 P3 7 11 8 9 10 8 7 13 12 ... 0.27
3 P4 12 8 13 5 9 6 9 13 13 ... 0.41
4 P5 8 5 13 11 6 7 9 14 9 ... 0.27
.. ...
806 P815 0 0 1 0 0 2 1 0 0 ... 0.00
807 P816 0 1 0 0 1 2 2 6 0 ... 0.43
808 P817 1 0 0 0 1 1 2 1 1 ... 0.50
809 P818 0 0 0 1 0 0 0 0 1 ... 0.00
810 P819 0 1 0 0 0 0 0 0 0 ... 0.00

Normalized 43 Normalized 44 Normalized 45 Normalized 46 \
0 0.22 0.28 0.39 0.50
1 0.40 0.50 0.10 0.10
2 1.00 0.18 0.18 0.36
3 0.47 0.06 0.12 0.24
4 0.53 0.27 0.60 0.20
.. ...
806 0.33 0.33 0.00 0.00

```

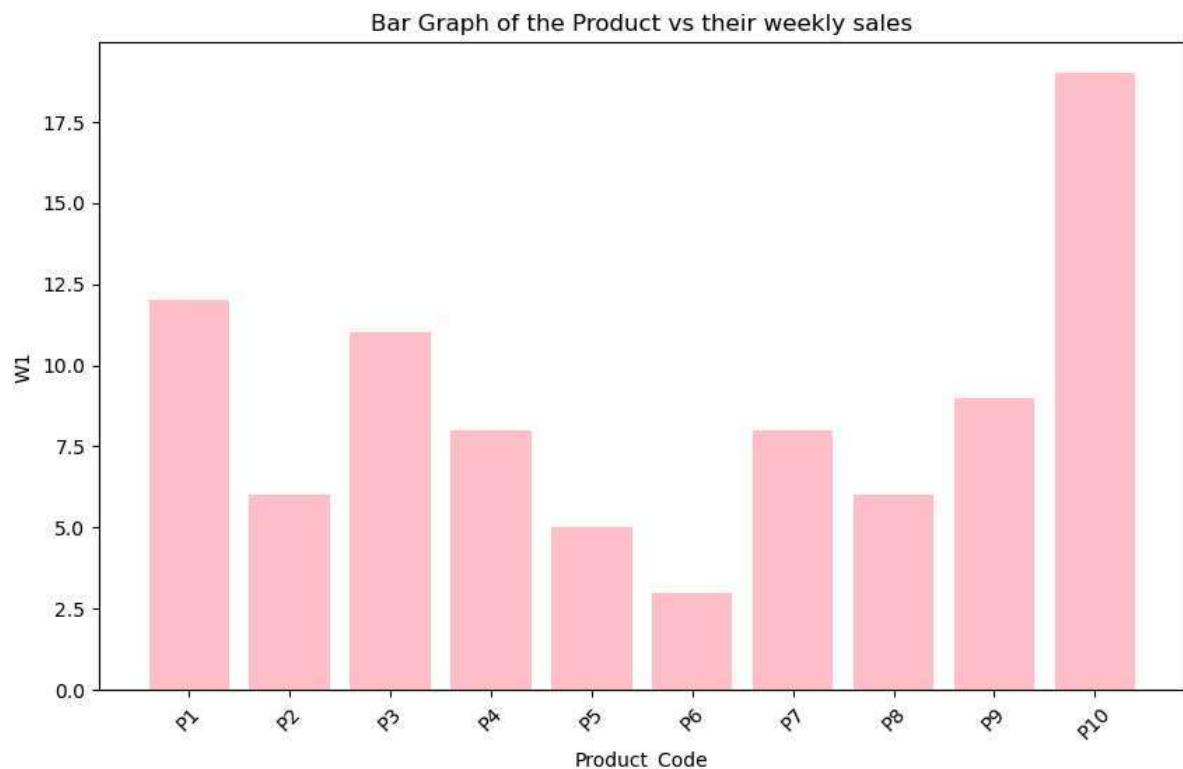
807	0.43	0.57	0.29	0.57	
808	0.00	0.00	0.50	0.50	
809	0.00	0.00	0.50	0.50	
810	0.00	0.00	0.00	0.00	
	Normalized 47	Normalized 48	Normalized 49	Normalized 50	Normalized 5
1					
0	0.00	0.22	0.17	0.11	0.3
9					
1	0.40	0.50	0.10	0.60	0.0
0					
2	0.45	1.00	0.45	0.45	0.3
6					
3	0.35	0.71	0.35	0.29	0.3
5					
4	0.20	0.13	0.53	0.33	0.4
0					
..
...					
806	0.33	0.00	0.00	0.67	0.0
0					
807	0.71	0.71	0.71	0.86	0.7
1					
808	0.00	0.00	0.00	1.00	0.7
5					
809	0.00	0.00	0.00	1.00	0.0
0					
810	0.00	0.00	0.00	0.00	0.3
3					

[811 rows x 107 columns]>

```
In [3]: import pandas as pd
import matplotlib.pyplot as plt

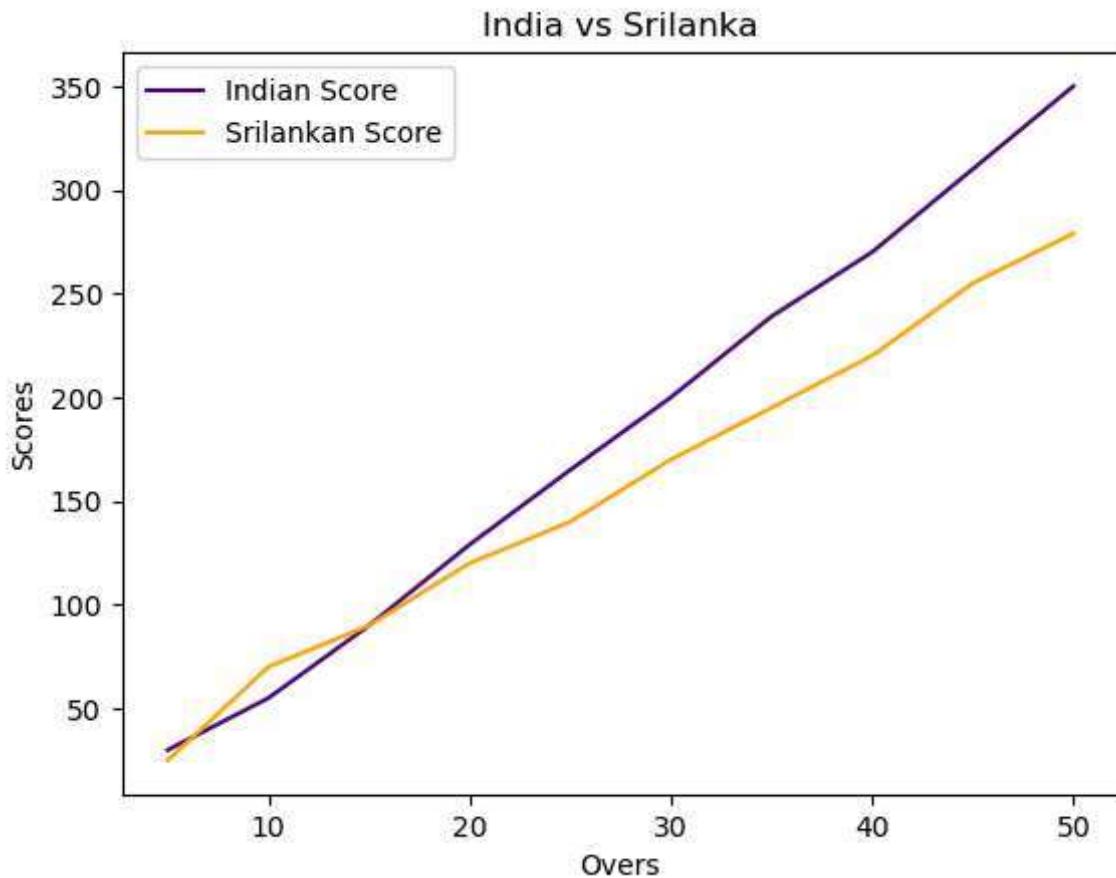
df = pd.read_csv('Sales_Transactions_Dataset_Weekly.csv')

subset_df = df.head(10)
plt.figure(figsize=(10,6))
plt.bar(subset_df['Product_Code'], subset_df['W1'], color='pink')
plt.xlabel('Product_Code')
plt.ylabel('W1')
plt.title('Bar Graph of the Product vs their weekly sales')
plt.xticks(rotation=45)
plt.show()
```

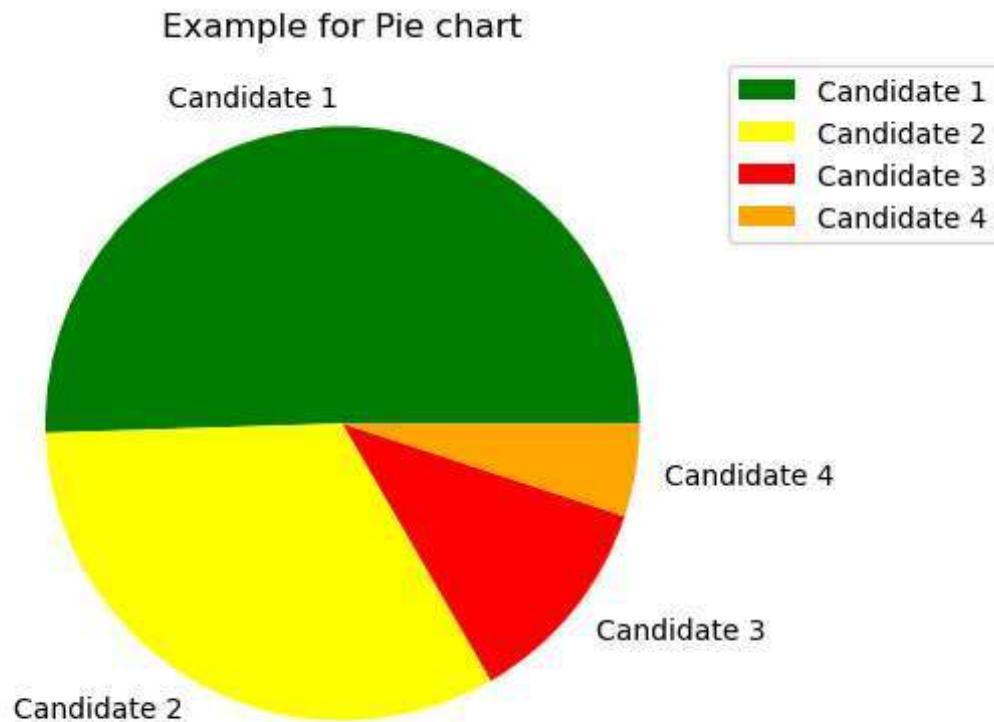


```
In [ ]:
```

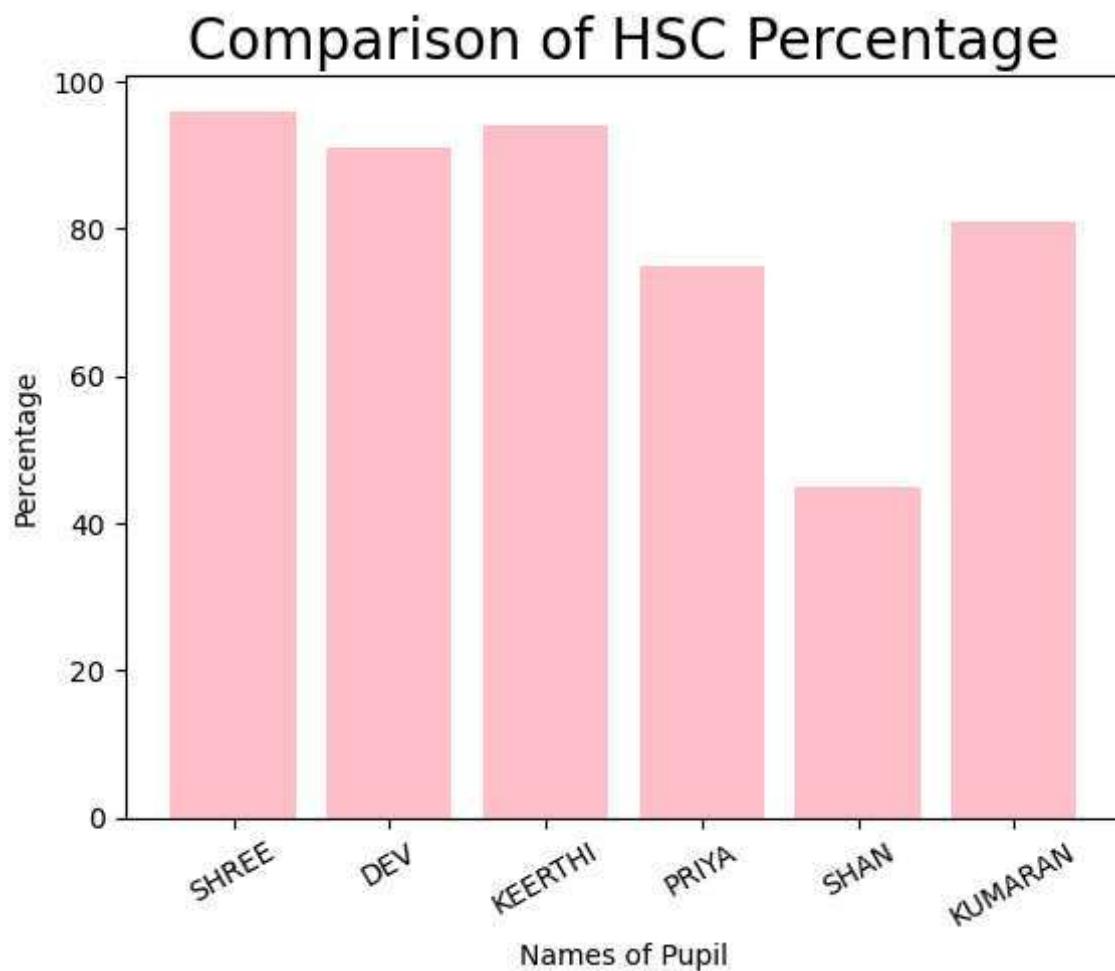
```
In [11]: import matplotlib.pyplot as cricket
Overs=list(range(5,51,5))
Indian_Score=[30,55,90,129,165,200,239,270,310,350]
Srilankan_Score=[25,70,90,120,140,170,195,220,255,279]
cricket.plot(Overs,Indian_Score,label='Indian Score',color='indigo')
cricket.plot(Overs,Srilankan_Score,label='Srilankan Score',color='orange')
cricket.title("India vs Srilanka")
cricket.xlabel("Overs")
cricket.ylabel("Scores")
cricket.legend()
cricket.show()
```



```
In [15]: import numpy as np
import matplotlib.pyplot as election
roles=['Candidate 1','Candidate 2','Candidate 3','Candidate 4']
count=np.array([100,65,23,10])
colours = ['green','yellow','red','orange']
election.pie(count,labels=roles,colors=colours)
election.legend(loc="upper left",bbox_to_anchor=(1,1))
election.title("Example for Pie chart")
election.show()
```



```
In [13]: import matplotlib.pyplot as hscmark
import numpy as np
Names = ['SHREE', 'DEV', 'KEERTHI', 'PRIYA', 'SHAN', 'KUMARAN']
xaxis = np.arange(len(Names))
Percentage_hsc = [96, 91, 94, 75, 45, 81]
hscmark.bar(Names, Percentage_hsc,color='pink')
hscmark.xticks(xaxis, Names, rotation=30)
hscmark.xlabel('Names of Pupil')
hscmark.ylabel('Percentage')
hscmark.title('Comparison of HSC Percentage', fontsize=20, color='black')
hscmark.show()
```



```
In [ ]:
```

In []:

```
In [4]: import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import gutenberg

# Download necessary data
nltk.download('gutenberg')
nltk.download('punkt')

# Load the raw text from the Gutenberg corpus
sample = gutenberg.raw("austen-emma.txt")

# Tokenize the text
token = word_tokenize(sample)

# Get the first 50 tokens
wlist = []
for i in range(50):
    wlist.append(token[i])

# Compute word frequencies
wordfreq = [wlist.count(w) for w in wlist]

# Print the pairs of words and their frequencies
print("Pairs\n" + str(list(zip(wlist, wordfreq))))
```

```
[nltk_data] Downloading package gutenberg to
[nltk_data]     C:\Users\DELL\AppData\Roaming\nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\DELL\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!

Pairs
[('[' , 1), ('Emma' , 2), ('by' , 1), ('Jane' , 1), ('Austen' , 1), ('1816' , 1),
(']' , 1), ('VOLUME' , 1), ('I' , 2), ('CHAPTER' , 1), ('I' , 2), ('Emma' , 2), ('Woo
dhouse' , 1), (',' , 5), ('handsome' , 1), (',' , 5), ('clever' , 1), (',' , 5), ('an
d' , 3), ('rich' , 1), (',' , 5), ('with' , 2), ('a' , 1), ('comfortable' , 1), ('hom
e' , 1), ('and' , 3), ('happy' , 1), ('disposition' , 1), (',' , 5), ('seemed' , 1),
('to' , 1), ('unite' , 1), ('some' , 1), ('of' , 2), ('the' , 2), ('best' , 1), ('ble
ssings' , 1), ('of' , 2), ('existence' , 1), (';' , 1), ('and' , 3), ('had' , 1), ('l
ived' , 1), ('nearly' , 1), ('twenty-one' , 1), ('years' , 1), ('in' , 1), ('the' ,
2), ('world' , 1), ('with' , 2)]
```

In []:

Exp 1.b

Pandas and Numpy Built in Function

```
import pandas as pd
import numpy as np

# Load the data
data = pd.read_csv('50_Startups.csv')

print("First few rows of the dataset:\n", data.head())
```

First few rows of the dataset:

	R&D Spend	Administration Spend	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

```
print("\nData Information:\n")
print(data.info())
```

Data Information:

```
<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
None
```

```
print("\nSummary Statistics:\n", data.describe())
```

Summary Statistics:

	R&D Spend	Administration Spend	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

```
rd_mean = np.mean(data['R&D Spend'])
rd_median = np.median(data['R&D Spend'])
rd_mode = data['R&D Spend'].mode()[0]
```

```
print("\nR&D Spend - Mean:", rd_mean)
print("R&D Spend - Median:", rd_median)
print("R&D Spend - Mode:", rd_mode)
```

R&D Spend - Mean: 73721.6156
R&D Spend - Median: 73051.08
R&D Spend - Mode: 0.0

Start coding or generate with AI.

Exp 2

Outlier detection

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

→ array([22, 48, 70, 7, 46, 92, 25, 31, 38, 12, 92, 4, 5, 55, 70, 61])

array.mean()

→ 42.375

np.percentile(array,25)

→ 19.5

Generate create a dataframe with 2 columns and 10 rows  
```

```
np.percentile(array,50)

→ 42.0

np.percentile(array,75)

→ 63.25

np.percentile(array,100)

→ 92.0

def outDetection(array):
    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)
    print(f"Lower range: {lr}, Upper range: {ur}")

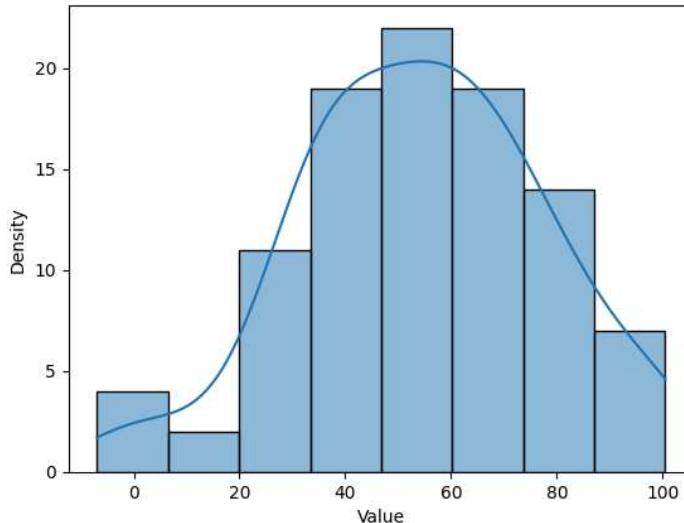
array = np.random.normal(50, 25, 100) # Example data
lr, ur = outDetection(array)

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

sns.histplot(new_array, kde=True)
plt.xlabel("Value")
plt.ylabel("Density")
plt.title("Distribution of Filtered Data")
plt.show()
```

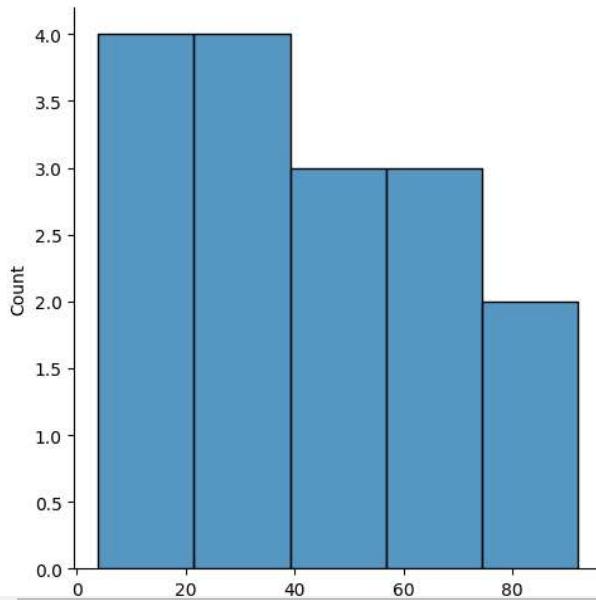


Distribution of Filtered Data



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

<seaborn.axisgrid.FacetGrid at 0x7d0420ce9720>



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

array([51.85014358, 34.83238679, 77.04117029, 99.26725507,
 26.25845082, 36.55006652, 29.2200325 , 0.50667154,
 92.68224723, 68.30384579, 85.22523265, 55.66082601,
 17.78219061, 30.61821963, 40.97552024, 38.12614615,
 65.53605085, 72.35388289, 35.05798932, 65.28154547,
 61.81971967, 32.88327386, 76.46605909, 78.38958672,
 5.3214148 , 42.68977065, 35.26354792, 35.57908961,
 63.13906371, 30.39185285, 52.19093272, 59.33087816,
 32.95300973, 44.81792357, 66.60719456, 64.22163439,
 57.54990457, 74.23770306, 48.42495291, 83.83619692,
 67.54061757, 78.38961525, 74.7177832 , 61.91326406,
 27.28447422, 99.22865942, 1.20433312, 64.77780984,
 58.48901255, 99.43959417, 25.34749132, 60.67309344,
 40.39807124, 88.98561119, 56.66616456, 76.25656706,
 14.30884977, 42.29200712, 57.87269478, 46.95598493,
 -6.92314481, 31.07105886, 43.14610168, 60.6575837 ,
 43.99197508, 51.06009871, 67.72425833, 57.53197055,
 69.21108884, 100.49349834, 57.58021145, 56.38544687,
 85.59862204, 53.43489242, 47.31269244, 45.15913115,
 47.64678649, 37.9176987 , 46.88784929, 65.72125821,

```
31.95537796, 72.94356225, 86.22206455, 38.10331451,
74.79789395, 53.18122882, 23.71646192, 37.42530255,
55.99624825, 48.16180217, 72.39876339, 85.0479005 ,
34.62866742, 70.10360734, 99.4590022 , 43.74685633,
84.41366831, 50.21390904])
```

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

→ (-9.910459929045743, 117.75504029782823)
```

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

```
→ array([ 51.85014358, 34.83238679, 77.04117029, 99.26725507,
26.25845082, 36.55006652, 29.2200325 , 0.50667154,
92.68224723, 68.30384579, 85.22523265, 55.66082601,
17.78219061, 30.61821963, 40.97552024, 38.12614615,
65.53605085, 72.35388289, 35.05798932, 65.28154547,
61.81971967, 32.88327386, 76.46605909, 78.38958672,
5.3214148 , 42.68977065, 35.26354792, 35.57908961,
63.13906371, 30.39185285, 52.19093272, 59.33087816,
32.95300973, 44.81792357, 66.60719456, 64.22163439,
57.54990457, 74.23770306, 48.42495291, 83.83619692,
67.54061757, 78.38961525, 74.7177832 , 61.91326406,
27.28447422, 99.22865942, 1.20433312, 64.77780984,
58.48901255, 99.43959417, 25.34749132, 60.67309344,
40.39807124, 88.98561119, 56.66616456, 76.25656706,
14.30884977, 42.29200712, 57.87269478, 46.95598493,
-6.92314481, 31.07105886, 43.14610168, 60.6575837 ,
43.99197508, 51.06009871, 67.72425833, 57.53197055,
69.21108884, 100.49349834, 57.58021145, 56.38544687,
85.59862204, 53.43489242, 47.31269244, 45.15913115,
47.64678649, 37.9176987 , 46.88784929, 65.72125821,
31.95537796, 72.94356225, 86.22206455, 38.10331451,
74.79789395, 53.18122882, 23.71646192, 37.42530255,
55.99624825, 48.16180217, 72.39876339, 85.0479005 ,
34.62866742, 70.10360734, 99.4590022 , 43.74685633,
84.41366831, 50.21390904])
```

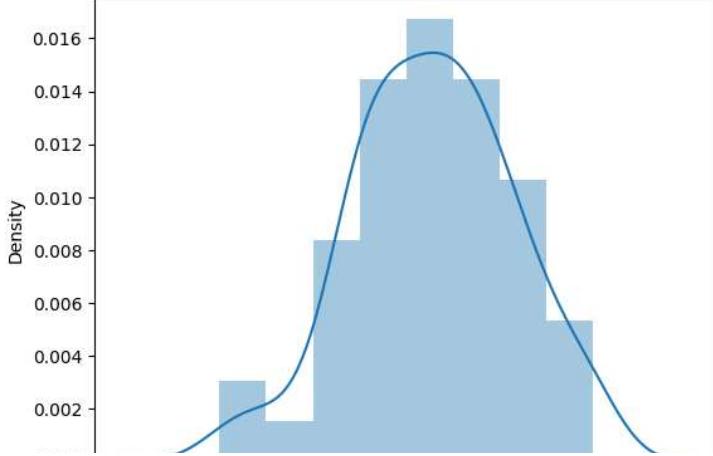
```
sns.distplot(final_array)
```

```
→ <ipython-input-28-7ba96ada5b76>: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)
<Axes: xlabel='Density'>
```



Exp 3

Missing and inappropriate data

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

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

NoOfPax	EstimatedSalary	Age_Group.1
0	2	40000 20-25
1	3	59000 30-35
2	2	30000 25-30
3	2	120000 20-25
4	2	45000 35+
5	2	122220 35+
6	-1	21122 35+
7	-10	345673 20-25
8	3	-99999 25-30
9	3	-99999 25-30
10	4	87777 30-35

```
df.duplicated()
```

0
0 False
1 False
2 False
3 False
4 False
5 False
6 False
7 False
8 False
9 True
10 False

```
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	grid icon
0	1	20-25	4	Ibis	veg	1300	2	40000	20-25	info icon
1	2	30-35	5	LemonTree	Non-Veg	2000	3	59000	30-35	edit icon
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	

Next steps: [Generate code with df](#)[View recommended plots](#)[New interactive sheet](#)

len(df)

10

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

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

print(df.columns)

```
Index(['CustomerID', 'Rating(1-5)', 'Hotel', 'FoodPreference', 'Bill',
       'NoOfPax', 'EstimatedSalary'],
      dtype='object')
```

import numpy as np

```
df.loc[df.CustomerID < 0, 'CustomerID'] = np.nan
df.loc[df.Bill < 0, 'Bill'] = np.nan
df.loc[df.EstimatedSalary < 0, 'EstimatedSalary'] = np.nan
df
```

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary	grid icon
0	1.0	20-25	4	Ibis	veg	1300.0	2	40000.0	info icon
1	2.0	30-35	5	LemonTree	Non-Veg	2000.0	3	59000.0	edit icon
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	

Next steps: [Generate code with df](#)[View recommended plots](#)[New interactive sheet](#)

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

→ <ipython-input-13-55d18ab59348>: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 t
  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 or:
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus

df['NoOfPax'].loc[(df['NoOfPax']<1) | (df['NoOfPax']>20)]=np.nan
<ipython-input-13-55d18ab59348>: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
df['NoOfPax'].loc[(df['NoOfPax']<1) | (df['NoOfPax']>20)]=np.nan
```

CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary	Actions
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

Next steps: [Generate code with df](#) [View recommended plots](#) [New interactive sheet](#)

```
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)
print(df.FoodPreference.unique())

→ <bound method Series.unique of 0      Veg
  1    Non-Veg
  2      Veg
  3      Veg
  4      Veg
  5  Non-Veg
  6      Veg
  7      Veg
  8  Non-Veg
  9  Non-Veg
Name: FoodPreference, dtype: object>
```

```
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
```

```
→ <ipython-input-19-9197dedb9332>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(...)

df.EstimatedSalary.fillna(round(df.EstimatedSalary.mean()), inplace=True)
<ipython-input-19-9197dedb9332>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(...)

df.NoOfPax.fillna(round(df.NoOfPax.median()), inplace=True)
<ipython-input-19-9197dedb9332>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(...)

df['Rating(1-5)'].fillna(round(df['Rating(1-5)'].median()), inplace=True)
<ipython-input-19-9197dedb9332>:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(...)
```

df.Bill.fillna(round(df.Bill.mean()), inplace=True)

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary	Actions
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	

Next steps:

[Generate code with df](#)

[View recommended plots](#)

[New interactive sheet](#)

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

Exp 4

Data Preprocessing

```
import numpy as np
import pandas as pd
df=pd.read_csv("/content/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	NaN	50.0	83000.0	No
9	France	37.0	67000.0	Yes

Double-click (or enter) to edit

```
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
```

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

```
→ <table>
  <thead>
    <tr>
      <th>Country</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>France</td>
    </tr>
  </tbody>
</table>
```

Country
France

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

```
→ <table>
```



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

```
→ <code>
pandas.core.series.Series
def __init__(data=None, index=None, dtype: Dtype | None=None, name=None, copy: bool | None=None,
fastpath: bool=False) -> None
    INDEX = None; THE RESULTING SERIES IS INDEXED WITH THE INDEX VALUES.
    dtype : str, numpy.dtype, or ExtensionDtype, optional
        Data type for the output Series. If not specified, this will be
        inferred from 'data'.
        See the :ref:`user guide <basics.dtypes>` for more usages.
    name : Hashable, default None
</code>
```

```
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	France	50.0	83000.0	No
9	France	37.0	67000.0	Yes

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

	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

```
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	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

```
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.Purchased.replace(['No','Yes'],[0,1],inplace=True)
```

updated_dataset

	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

Start coding or [generate](#) with AI.



Exp 5

EDA-Qualitative and Quantitative Plots

Start coding or generate with AI.

Generate

print hello world using rot13

Close

Focus the last run cell

```
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

Next steps:

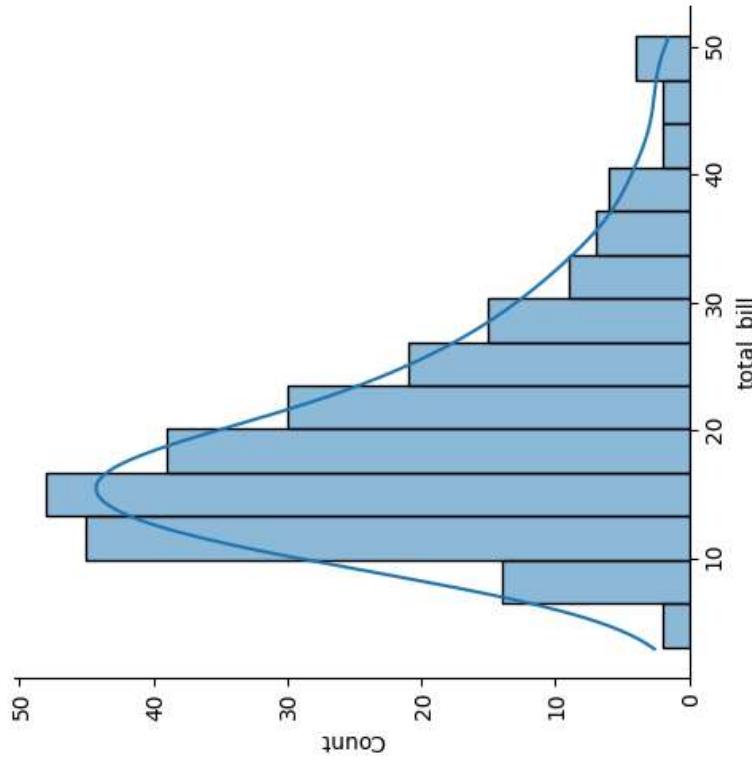
Generate code with tips

View recommended plots

New interactive sheet

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

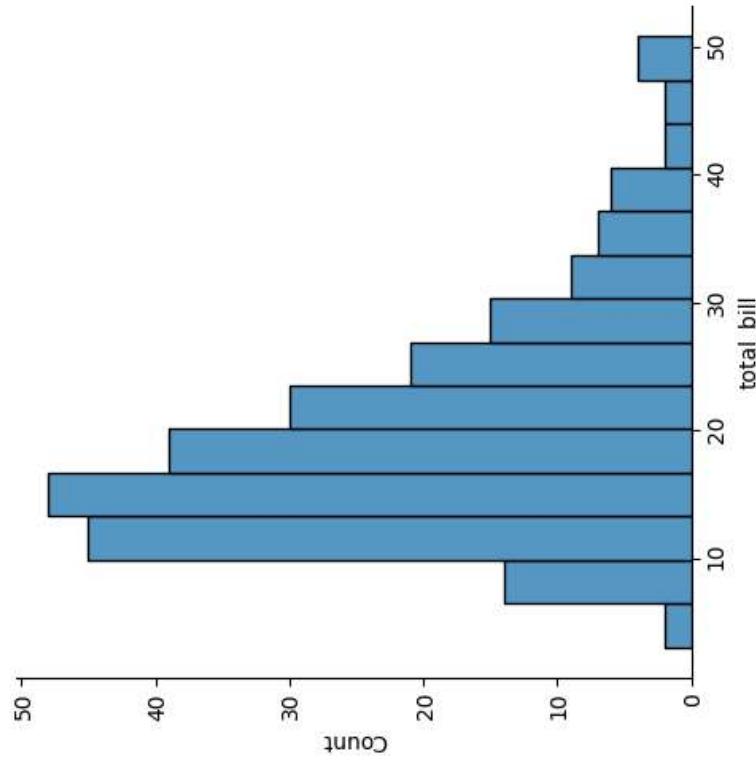
```
→ <seaborn.axisgrid.FacetGrid at 0x7ed92f79f8b0>
```



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

Focus the last run cell

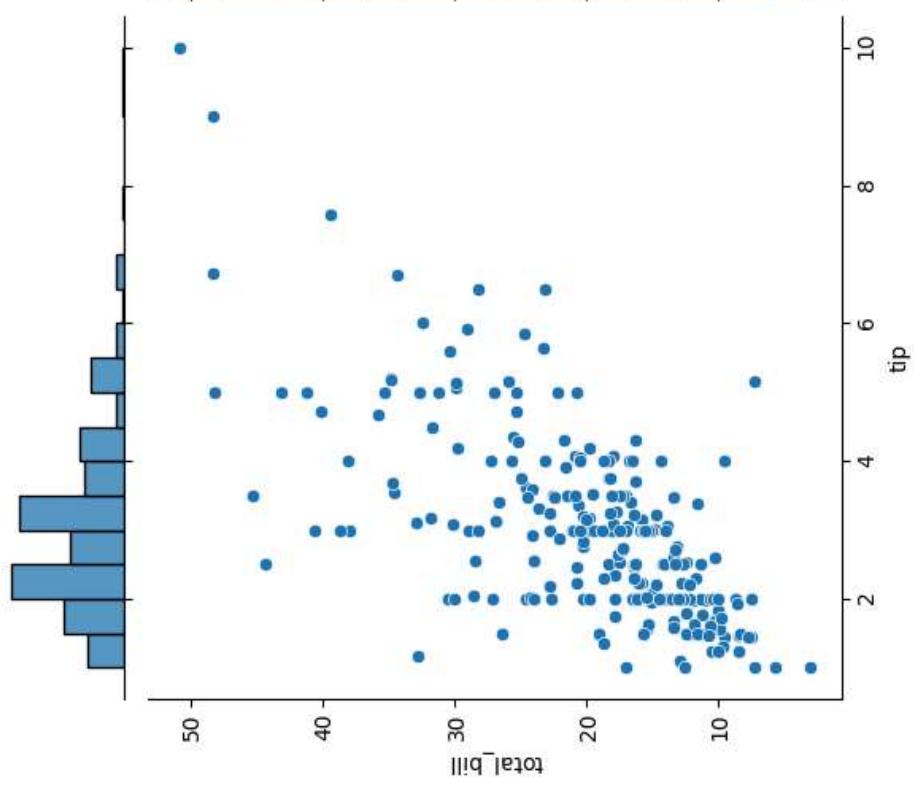
```
→ <seaborn.axisgrid.FacetGrid at 0x7ed96b8ad2d0>
```



Focus the last run cell

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

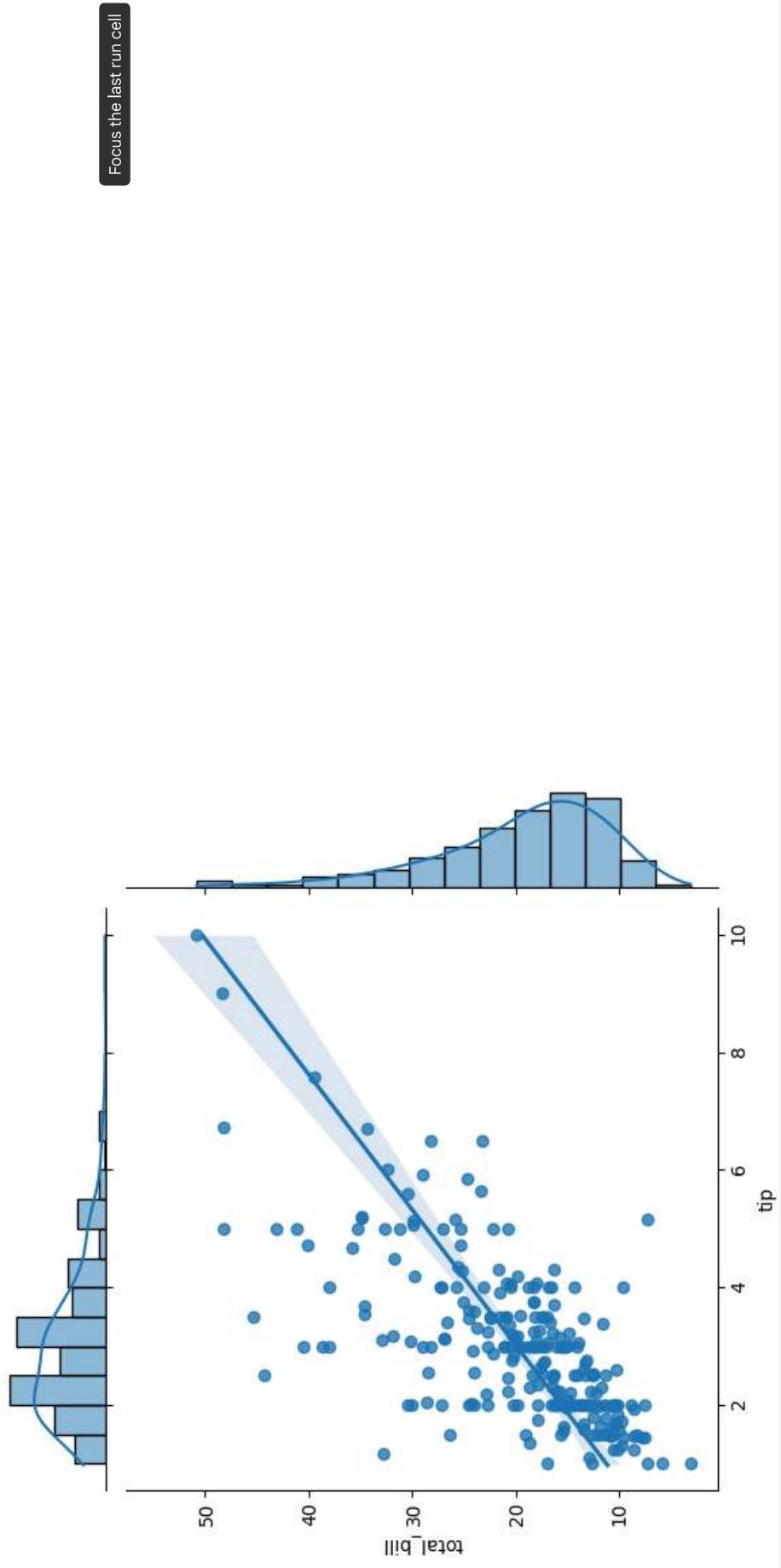
```
→ <seaborn.axisgrid.JointGrid at 0x7ed92e7d1750>
```



Focus the last run cell

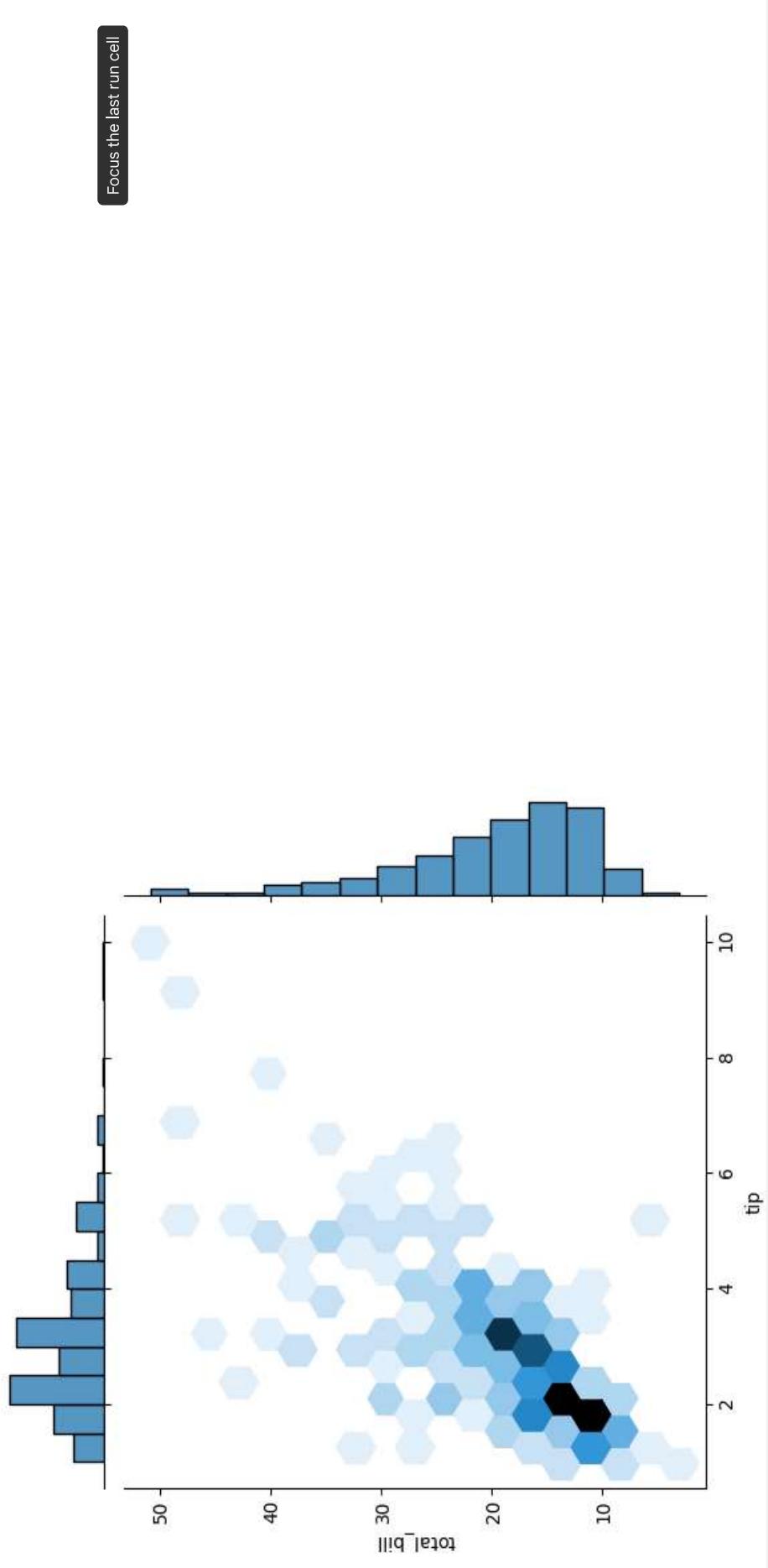
```
sns.jointplot(x=tips.total_bill,y=tips.tip,kind="reg")
```

```
→ <seaborn.axisgrid.JointGrid at 0x7ed92e7d16c0>
```



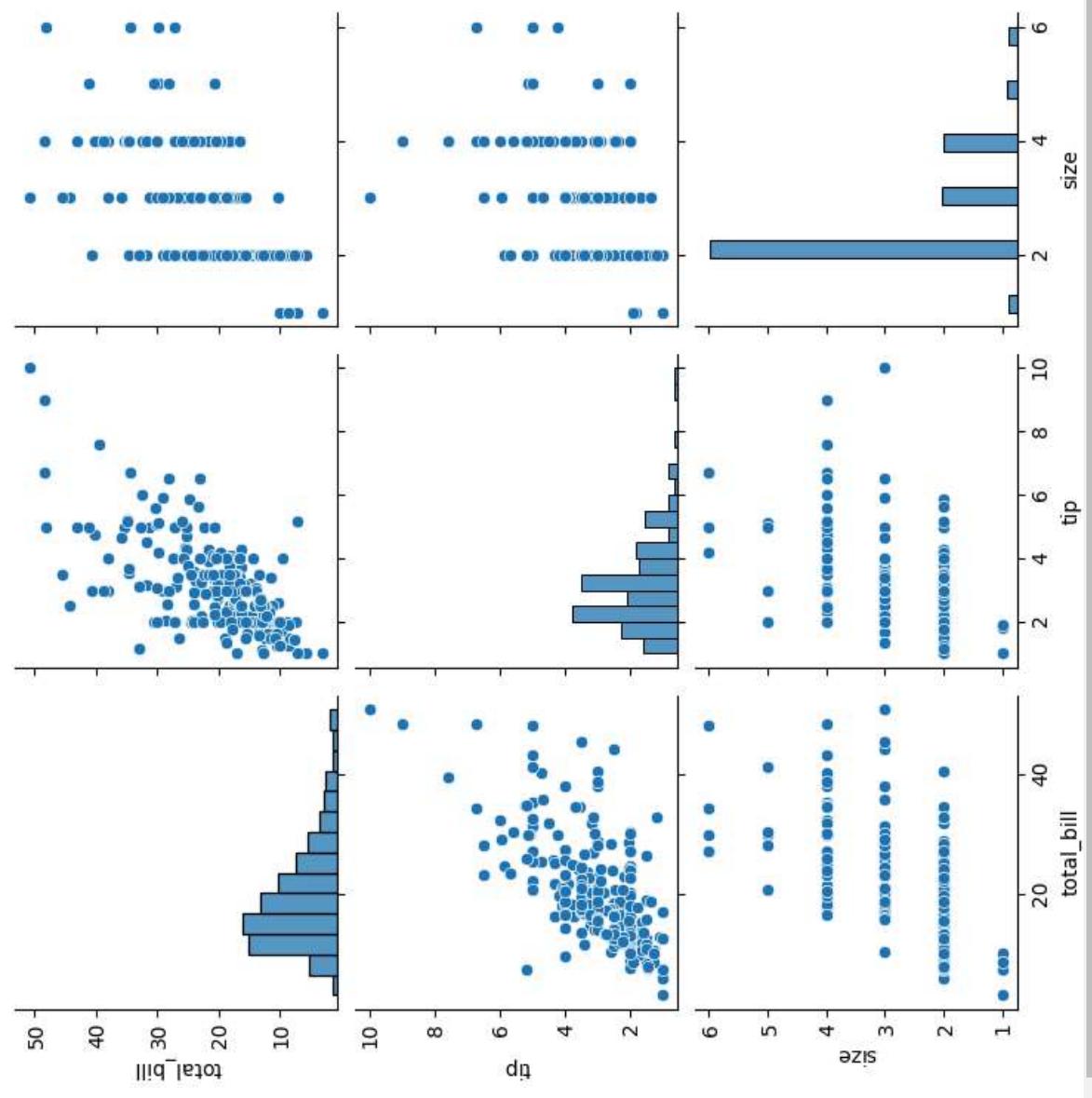
```
sns.jointplot(x=tips.total_bill,y=tips.tip,kind="hex")
```

```
→ <seaborn.axisgrid.JointGrid at 0x7ed92bbe52860>
```



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

```
→ <seaborn.axisgrid.PairGrid at 0x7ed92b44c2e0>
```

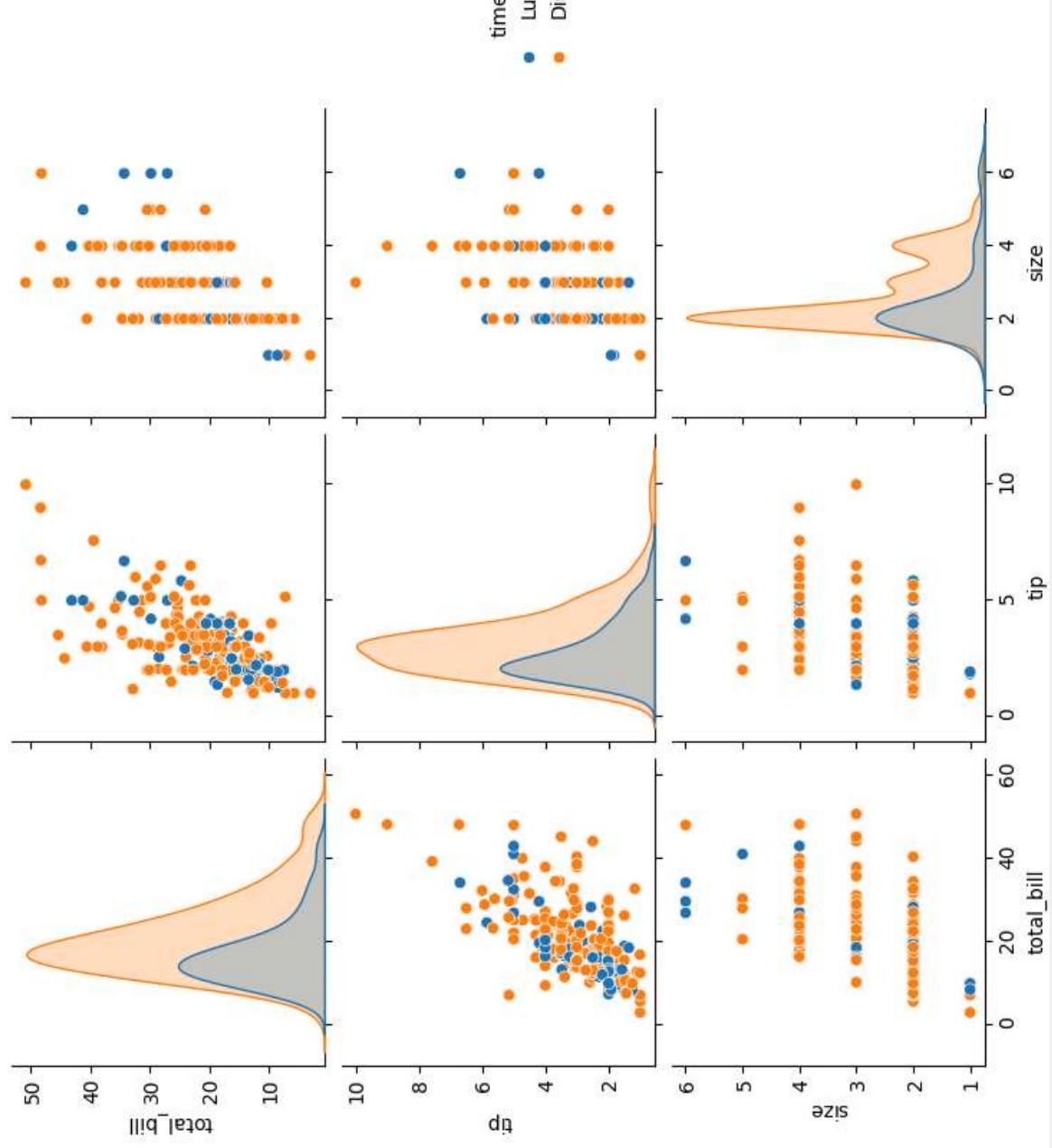


Focus the last run cell

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

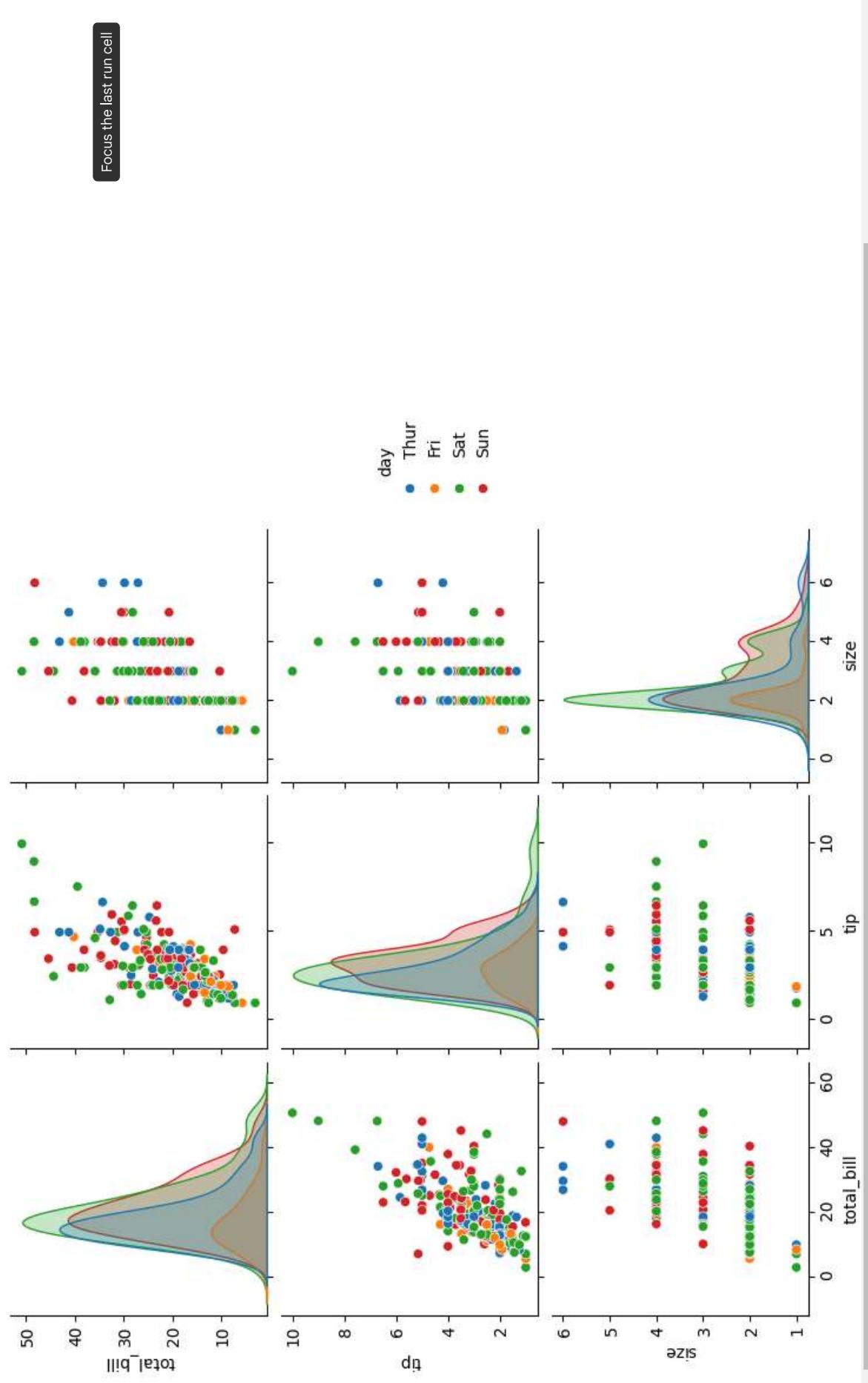


<seaborn.axisgrid.PairGrid at 0x7ed9abb86f80>



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

<seaborn.axisgrid.PairGrid at 0x7ed92a836a40>



```
sns.heatmap(tips.corr(numeric_only=True), annot=True)
```

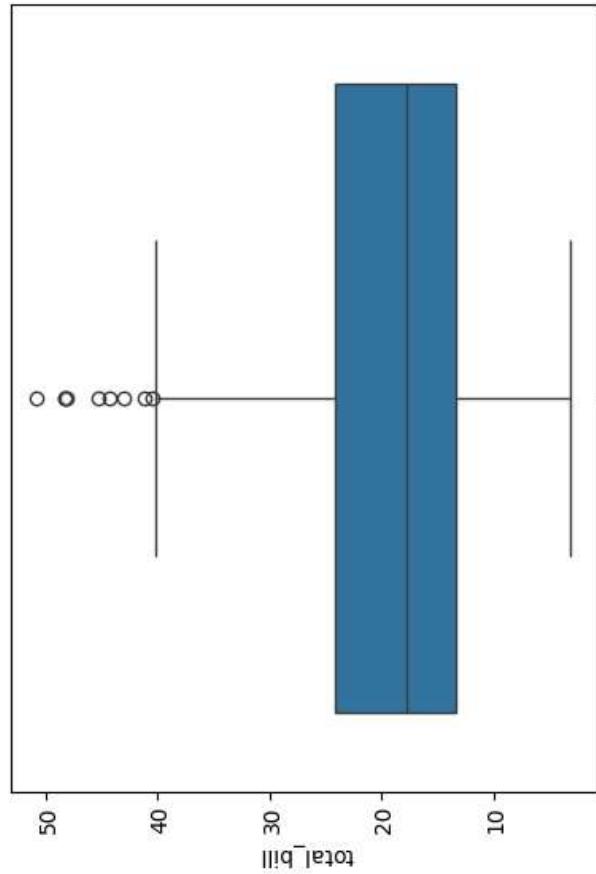
<Axes: >



Focus the last run cell

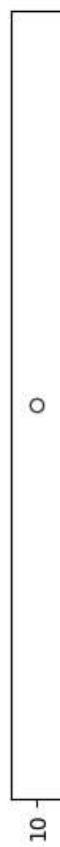
`sns.boxplot(tips.total_bill)`

```
→ <Axes: ylabel='total_bill'>
```



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

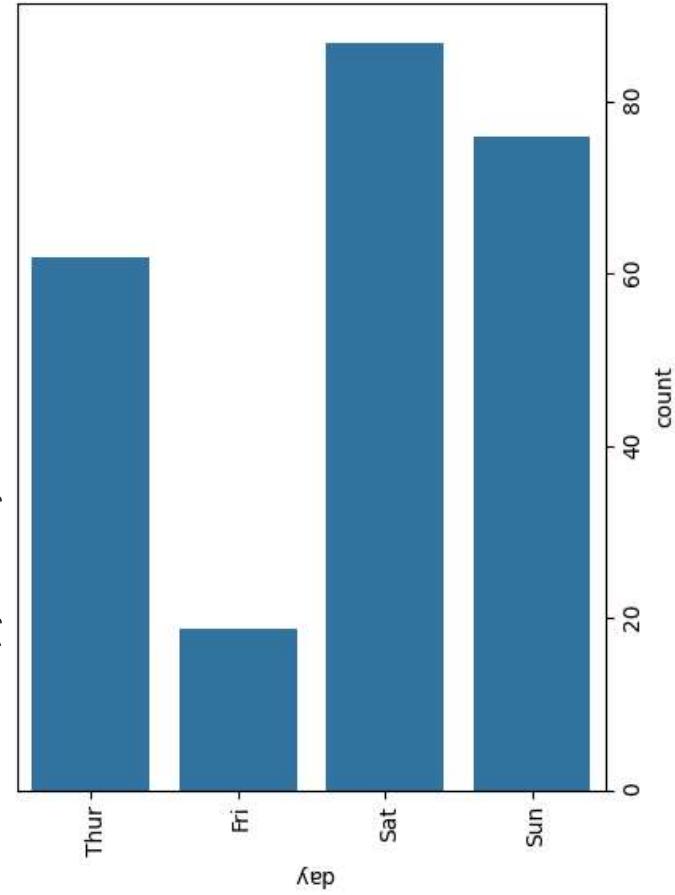
```
→ <Axes: ylabel='tip'>
```



```
sns.countplot(tips.day)
```

Focus the last run cell

```
→ <Axes: xlabel='count', ylabel='day'>
```



```
sns.countplot(tips.sex)
```

```
→ <Axes: xlabel='count', ylabel='sex'>
```



Exp 6

Random Sampling and Sampling distribution

Start coding or generate with AI.

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

# Parameters for the population
population_mean = 50
population_std = 10
population_size = 100000

# Generate the population
population = np.random.normal(population_mean, population_std, population_size)

# Define sample sizes and number of samples
sample_sizes = [30, 50, 100]
num_samples = 1000

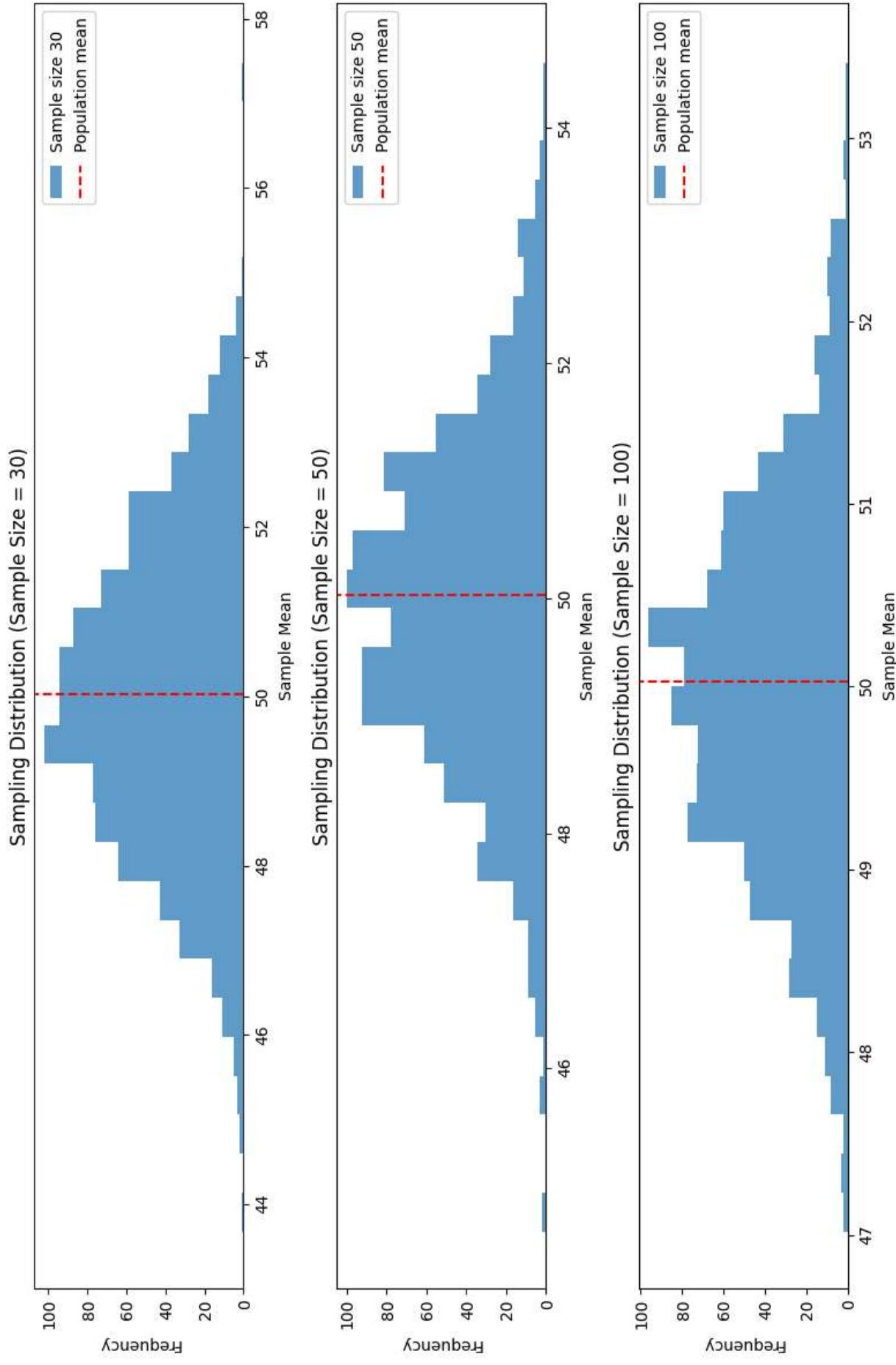
# Dictionary to store sample means
sample_means = {size: [] for size in sample_sizes}

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

# Plotting the histograms
plt.figure(figsize=(12, 8))

for i, size in enumerate(sample_sizes, start=1):
    plt.subplot(len(sample_sizes), 1, i)
    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()
```



Exp 7

Z-Test

Start coding or generate with AI.

```
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 mean
sample_std = np.std(sample_data, ddof=1) # Sample standard deviation with Bessel's correction
n = len(sample_data) # Number of observations

# Calculate the Z-statistic
z_statistic = (sample_mean - population_mean) / (sample_std / np.sqrt(n))

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

# Print results
print(f"Sample Mean: {sample_mean:.2f}")
print(f"Sample Standard Deviation: {sample_std:.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: No significant difference from 150 grams.")
```

Sample Mean: 150.20
Sample Standard Deviation: 1.71
Z-Statistic: 0.6406

P-Value: 0.5218

Fail to reject the null hypothesis: No significant difference from 150 grams.

Exp 8

T-Test

Start coding or generate with AI.

Generate print hello world using rot13

Close

Q

```
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}")
# 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.
```

Exp 9

ANOVA Test

Start coding or generate with AI.

```
Generate print hello world using rot13 Close Q

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_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: 9.6729838826683818
→ Treatment B Mean Growth: 11.137680744437432
https://colab.research.google.com/drive/1I-tNv5JOnQylnzz2wsqYxFXGaGTS-WQK#scrollTo=U3SHPWhy2Xmnx&printMode=true
```

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	3.593	7.2252	True
B	C	4.1276	0.0	2.4946	5.7605	True

Start coding or generate with AI.

 Generate 10 random numbers using numpy



Close

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

dataset = pd.read_csv('Social_Network_Ads.csv')

print(dataset.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

X = dataset[['Age', 'EstimatedSalary']]
y = dataset['Purchased']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_scaled_df = pd.DataFrame(X_scaled, columns=['Age_scaled', 'EstimatedSalary_scaled'])

Age_scaled  EstimatedSalary_scaled
0    -1.781797           -1.490046
1    -0.253587           -1.460681
2    -1.113206           -0.785290
3    -1.017692           -0.374182
4    -1.781797            0.183751
```

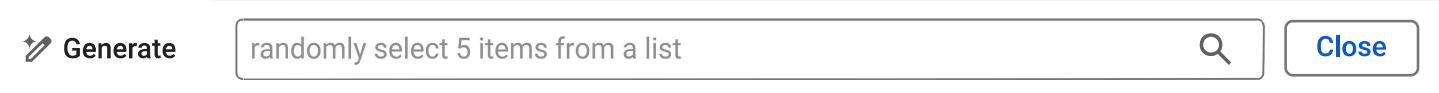
Start coding or generate with AI.

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

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'>
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.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	

 Generate

a slider using jupyter widgets



Close

```
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_s
```

 Generate

print hello world using rot13



Close

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

▼ LinearRegression  

LinearRegression()

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



0.9645401573418146

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



0.9024461774180497

```
model.coef_
```

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

 Generate

a slider using jupyter widgets



Close

```
model.intercept_
```

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

 Generate

a slider using jupyter widgets



Close

```
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: 25
```

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

```
Estimated Salary for 25.0 years of experience is: [[260916.96608755]]
```

Start coding or generate with AI.

Generate 10 random numbers using numpy Close

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

Generate a slider using jupyter widgets Close

```
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

Next steps: [Generate code with df](#) [View recommended plots](#) [New interactive sheet](#)

Generate print hello world using rot13 Close

```
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]
```

```
test: 0.86, train: 0.83, random State: 319
Test: 0.87, Train: 0.84, Random State: 321
Test: 0.91, Train: 0.83, Random State: 322
Test: 0.86, Train: 0.84, Random State: 331
Test: 0.85, Train: 0.84, Random State: 333
Test: 0.90, Train: 0.84, Random State: 336
Test: 0.87, Train: 0.84, Random State: 337
Test: 0.88, Train: 0.83, Random State: 343
Test: 0.90, Train: 0.84, Random State: 351
Test: 0.87, Train: 0.83, Random State: 352
Test: 0.90, Train: 0.82, Random State: 354
Test: 0.87, Train: 0.84, Random State: 357
Test: 0.87, Train: 0.84, Random State: 358
Test: 0.85, Train: 0.85, Random State: 361
Test: 0.88, Train: 0.83, Random State: 362
Test: 0.93, Train: 0.82, Random State: 363
Test: 0.88, Train: 0.83, Random State: 366
Test: 0.89, Train: 0.85, Random State: 369
Test: 0.89, Train: 0.84, Random State: 371
Test: 0.90, Train: 0.83, Random State: 376
Test: 0.91, Train: 0.81, Random State: 377
Test: 0.88, Train: 0.84, Random State: 378
Test: 0.88, Train: 0.84, Random State: 379
Test: 0.86, Train: 0.84, Random State: 381
Test: 0.86, Train: 0.83, Random State: 382
Test: 0.88, Train: 0.85, Random State: 386
Test: 0.84, Train: 0.84, Random State: 388
Test: 0.86, Train: 0.85, Random State: 390
Test: 0.88, Train: 0.82, Random State: 394
Test: 0.88, Train: 0.85, Random State: 399
Test: 0.88, Train: 0.84, Random State: 400
```

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

# Splitting the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2, random_state=42)

# Initializing and fitting the logistic regression model
finalModel = LogisticRegression()
finalModel.fit(x_train, y_train)
```

```
→ LogisticRegression ⓘ ?  
LogisticRegression()
```

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

```
→ 0.8375
0.8875
```

```
Generate 10 random numbers using numpy 
```

```
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
```

Start coding or generate with AI.

