

Exp No:1.aAnalyze the trend of data science job postings over the last decade

Description: Use web scraping (e.g., BeautifulSoup) or APIs (e.g., LinkedIn API) to gather data on the number of data science job postings each year. Use pandas for data manipulation and matplotlib/seaborn for visualization.

Code:

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

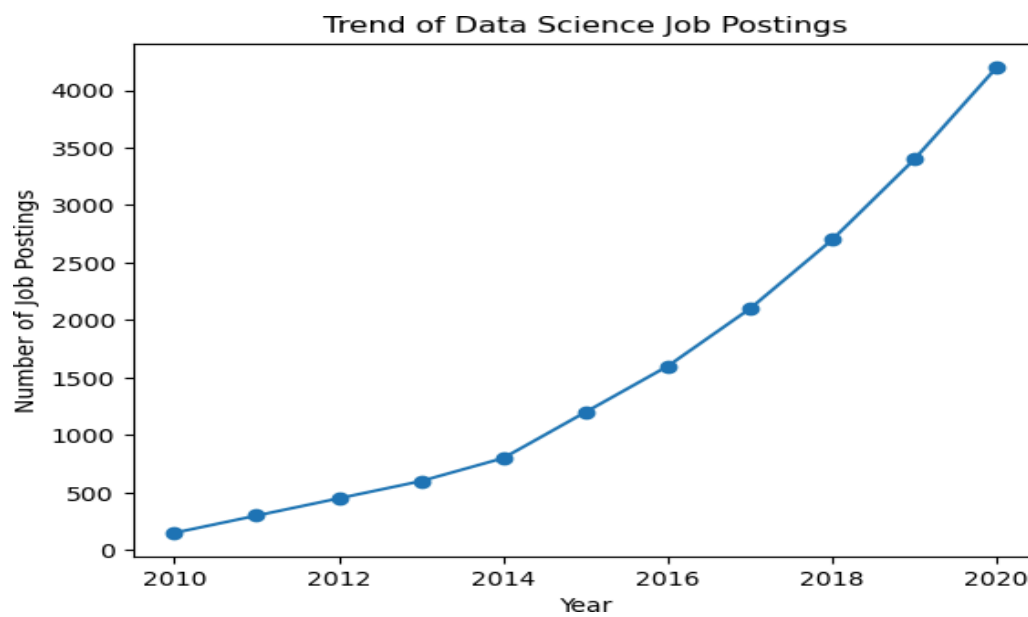
df = pd.DataFrame(data)
plt.plot(df['Year'], df['Job Postings'], marker='o')
plt.title('Trend of Data Science Job Postings')
plt.xlabel('Year')
plt.ylabel('Number of Job Postings')
plt.show()
```

Sample Data Input:

Year =2010, 2021

Job Postings=150, 300, 450, 600, 800, 1200, 1600, 2100, 2700, 3400, 4200

Sample Output:



Exp No:1.b Analyze and visualize the distribution of various data science roles (Data Analyst, Data Engineer, Data Scientist, etc.) from a dataset.

Description: Use a dataset of job postings and categorize them into different roles. Visualize the distribution using pie charts or bar plots.

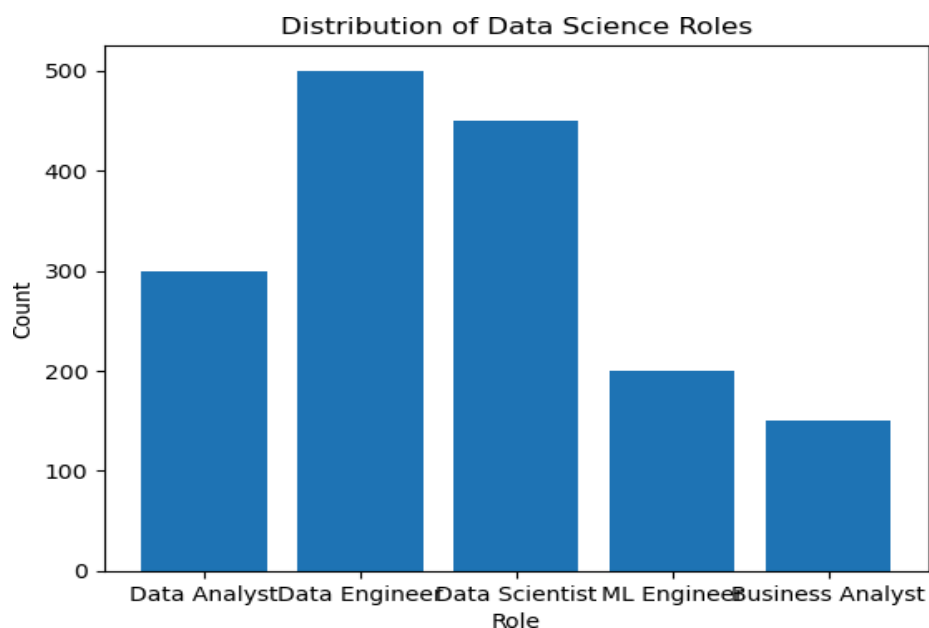
```
roles = ['Data Analyst', 'Data Engineer', 'Data Scientist',  
        'ML Engineer', 'Business Analyst']  
counts = [300, 500, 450, 200, 150]  
plt.bar(roles, counts)  
plt.title('Distribution of Data Science Roles')  
plt.xlabel('Role')  
plt.ylabel('Count')  
plt.show()
```

Sample Data Input:

roles = Data Analyst, Data Engineer, Data Scientist, ML Engineer, Business Analyst.

counts = 300, 500, 450, 200, 150.

Sample Output:



Exp No:1.c Conduct an experiment to differentiate Structured , Un-structured and Semi structured data based on data sets given.

Description: Create small datasets for each type and explain their characteristics.

```
# Structured data example

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

# Unstructured data example
unstructured_data = "This is an example of unstructured data. It can be a
piece of text, an image, or a video file."
print("\nUnstructured Data:\n", unstructured_data)

# Semi-structured data example (JSON)
semi_structured_data = {'ID': 1, 'Name': 'Alice', 'Attributes': {'Height':
165, 'Weight': 68}} print("\nSemi-structured Data:\n",
semi_structured_data)
```

OUTPUT:

Structured Data:

	ID	Name	Age
0	1	Alice	25
1	2	Bob	30
2	3	Charlie	35

Unstructured Data:

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

Semi-structured Data:

```
{'ID': 1, 'Name': 'Alice', 'Attributes': {'Height': 165, 'Weight': 68}}
}
```

Exp No:1.d Conduct an experiment to encrypt and decrypt given sensitive data.

Description: Use the cryptography library to encrypt and decrypt a piece of data.

```
# Generate key and encrypt data
from cryptography.fernet import
    Fernet key =
    Fernet.generate_key()
f = Fernet(key)
token = f.encrypt(b'Rajalakshmi Engineering
    College") token
b'...'
f.decrypt(token)
b'Rajalakshmi Engineering
    College' key =
    Fernet.generate_key()
    cipher_suite = Fernet(key)
plain_text = b'Rajalakshmi Engineering College."
cipher_text =
    cipher_suite.encrypt(plain_text) #
```

Decrypt data

```
decrypted_text =
```

OUTPUT:

```
Original Data: b'Rajalakshmi Engineering College.'
Encrypted Data: b'gAAAAABmmnHEhy1ZC1-Dox-URNAvYqFFGfJemMb_fKmXqhATe3AAn
XEOOnSzaNRINI4zAbNK8lgx2pprrNn-j6agbWDf30OwbrZsBuz8fnDdX7N35WwloDu3m6pVv
kwv8GYDSI8Fov1tV'
Decrypted Data: b'Rajalakshmi Engineering College.'
```

Exp No:2 Upload and Analyze the data set given in csv format and perform data preprocessing and visualization.

Description: Use sample data set sales-data.csv.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# Load the data into a pandas DataFrame
file_path='C:\sales_data.csv'
df = pd.read_csv(file_path)
# Display the first few rows of the DataFrame
print(df.head())
# Check for missing
values
print(df.isnull().sum(
))
# Fill or drop missing values if necessary
df['Sales'].fillna(df['Sales'].mean(), inplace=True)
df.dropna(subset=['Product', 'Quantity', 'Region'],
inplace=True) # Summary statistics
print(df.describe())
# Group by product and calculate the total sales and
quantity product_summary =
```

```
# Bar plot of total sales by
product
plt.figure(figsize=(10, 6))
plt.bar(product_summary['Product'], product_summary['Sales'])
plt.xlabel('Product')
plt.ylabel('Total Sales')
plt.title('Total Sales by
Product') plt.show()

# Line plot of sales over time
df['Date'] =
pd.to_datetime(df['Date'])
sales_over_time = df.groupby('Date').agg({'Sales':
'sum'}).reset_index() plt.figure(figsize=(10, 6))
plt.plot(sales_over_time['Date'],
sales_over_time['Sales']) plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.title('Sales Over Time')
plt.show()

# Pivot table to analyze sales by region and product
pivot_table = df.pivot_table(values='Sales', index='Region',
columns='Product', aggfunc=np.sum, fill_value=0)
print(pivot_table)
```

OUTPUT:

```
Date      Product Sales Quantity Region
0  2023-01-01  Product A      200          4  North
1  2023-01-02  Product B      150          3  South
2  2023-01-03  Product A      220          5  North
3  2023-01-04  Product C      300          6   East
4  2023-01-05  Product B      180          4   West
```

```
Date      0
```

```
Product    0
```

```
Sales      0
```

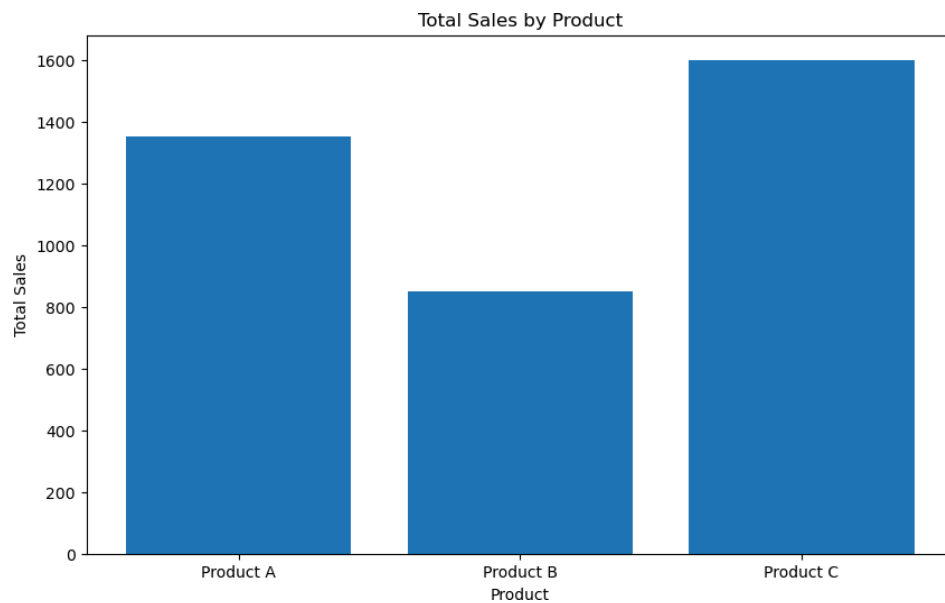
```
Quantity    0
```

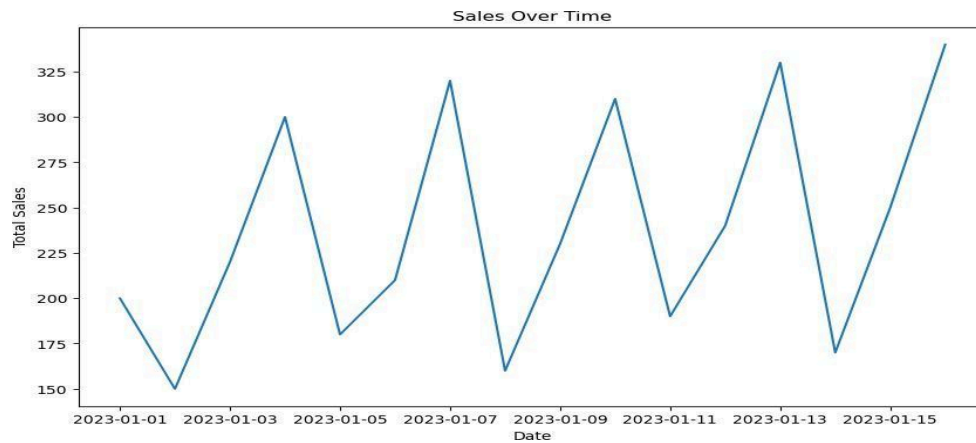
```
Region      0
```

```
dtype: int64
```

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

```
      Product Sales  Quantity
0  Product A      1350         33
1  Product B       850         17
2  Product C      1600         36
```





```
Product Product A Product B Product C Region
```

```
East      0      0      1600
```

```
North    1350      0      0
```

```
South      0     480      0
```

```
West      0     370      0
```

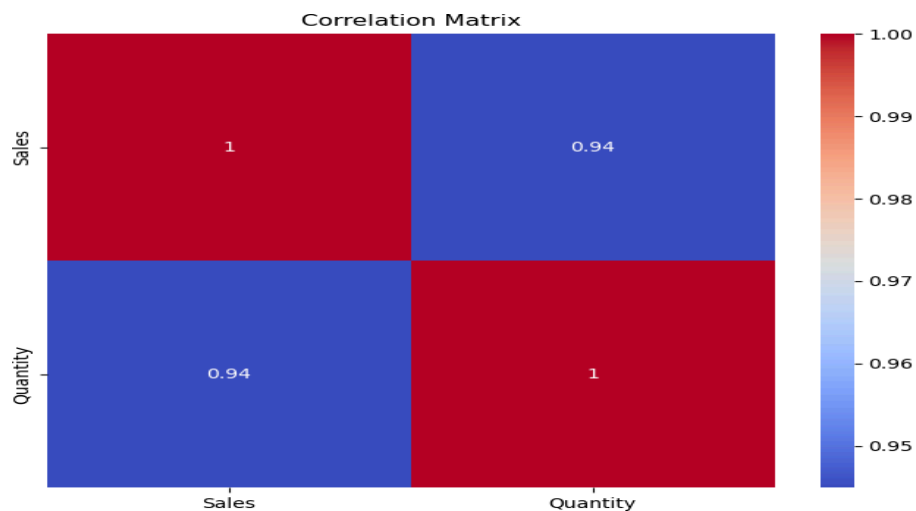
```
          Sales  Quantity
```

```
Sales     1.000000  0.944922
```

```
Quantity  0.944922  1.000000
```

```
C:\Users\Ayyadurai\AppData\Local\Temp\ipykernel_9648\511106317.py:49: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated
. In a future version, it will default to False. Select only valid columns or
specify the value of numeric_only to silence this warning.
```

```
correlation_matrix = df.corr()
```

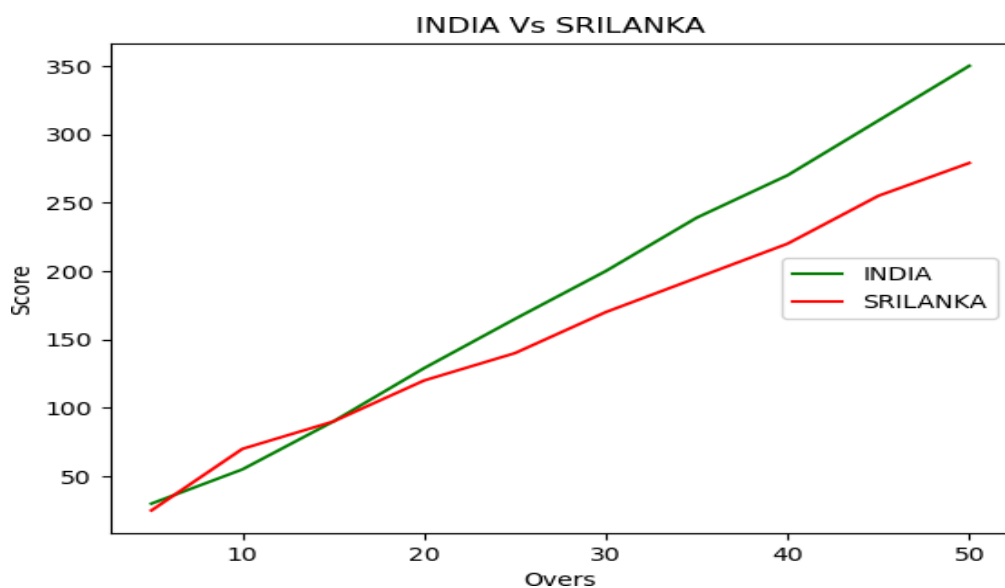


Exp No:3.a Conduct an experiment to show data visualization using line plot

Description: Take any sample data either through csv file or data fetched directly through code.

```
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=[25,70,90,120,140,170,195,2
20,255,279]
cricket.plot(Overs,Indian_Score)
cricket.plot(Overs,Srilankan_Score)
cricket.show()
cricket.title("INDIA Vs
SRILANKA")
cricket.xlabel("Overs")
cricket.ylabel("Score")
cricket.legend()
cricket.plot(Overs,Indian_Score,color="green",label="INDIA")
cricket.plot(Overs,Srilankan_Score,color="red",label="SRILAN
```

OUTPUT:

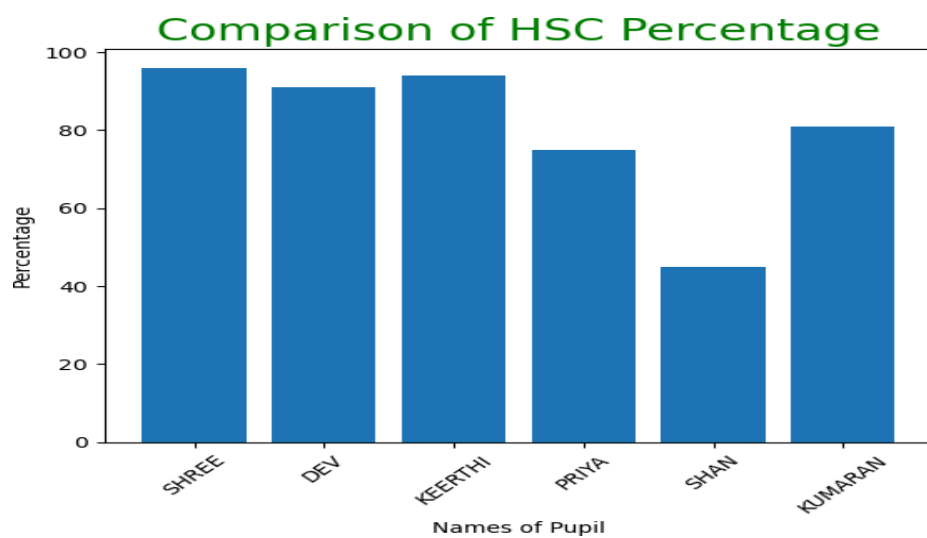


Exp No:3.b Conduct an experiment to show data visualization using bar chart

Description: Take any sample data either through csv file or data fetched directly through code.

```
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)
hscmark.xticks(xaxis, Names,
rotation=45)
hscmark.xlabel("Names of Pupil")
hscmark.ylabel("Percentage")
hscmark.title("Comparison of HSC Percentage", fontsize=20,
color="green")
hscmark.show()
```

OUTPUT:

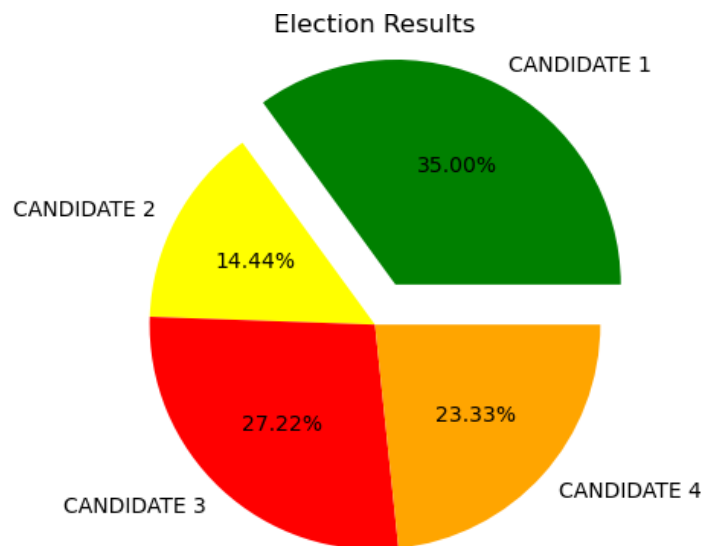


Exp No:3.c Conduct an experiment to show data visualization using pie chart

Description: Take any sample data either through csv file or data fetched directly through code.

```
import matplotlib.pyplot as
election # Election data
labels = ['CANDIDATE 1', 'CANDIDATE 2', 'CANDIDATE 3',
'CANDIDATE 4']
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()
```

OUTPUT:



Exp No:4 To Count the frequency of occurrence of a word in a body of text is often needed during text processing.

Description: Import the word_tokenize function and gutenber.

```
import nltk
from nltk.tokenize import
    word_tokenize from nltk.corpus
    import gutenber
# Download the gutenber corpus if not already installed
    nltk.download('gutenber')
nltk.download('punkt') # Ensure the punkt tokenizer models are
    also downloaded # Load the text from the gutenber corpus
sample = gutenber.raw("austen-emma.txt") # Change to an existing
    text from the corpus # Tokenize the sample text
token = word_tokenize(sample)
# Create a list of the first 50 tokens
wlist = []
    for i in range(50):
        wlist.append(token[i])

# Calculate the frequency of each word in the
    list wordfreq = [wlist.count(w) for w in
        wlist]
```

OUTPUT:

```
Pairs
[(['', 1), ('Emma', 2), ('by', 1), ('Jane', 1), ('Austen', 1), ('1816',
1), (']', 1), ('VOLUME', 1), ('I', 2), ('CHAPTER', 1), ('I', 2), ('Emma',
2), ('Woodhouse', 1), (',', 5), ('handsome', 1), (',', 5), ('clever',
1), (',', 5), ('and', 3), ('rich', 1), (',', 5), ('with', 2), ('a', 1),
('comfortable', 1), ('home', 1), ('and', 3), ('happy', 1), ('disposi
tion', 1), (',', 5), ('seemed', 1), ('to', 1), ('unite', 1), ('some', 1),
('of', 2), ('the', 2), ('best', 1), ('blessings', 1), ('of', 2), ('e
xistence', 1), (';', 1), ('and', 3), ('had', 1), ('lived', 1), ('nearly',
1), ('twenty-one', 1), ('years', 1), ('in', 1), ('the', 2), ('world',
1), ('with', 2)]
```



```
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

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

```
↗
```

	Empd	Name	Salary
0	1	Smith	50000
1	2	Jones	60000

```
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("/content/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	91304.77	366168.42	Florida	166187.04

```
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	116083.80	45173.06	California	14681.40

```
import numpy as np
import pandas as pd
df=pd.read_csv("/content/employee.csv")
```

```
df.head()
```

	emp id	name	salary
0	1	SREE VARSSINI K S	5000
1	2	SREEMATHI B	6000
2	3	SREYA G	7000
3	4	SREYASKARI MULLAPUDI	5000
4	5	SRI AKASH U G	8000

```
df.tail()
```

	emp id	name	salary
2	3	SREYA G	7000
3	4	SREYASKARI MULLAPUDI	5000
4	5	SRI AKASH U G	8000
5	6	SRI HARSHAVARDHANAN R	3000
6	7	SRI HARSHAVARDHANAN R	6000

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 3 columns):
#   Column  Non-Null Count  Dtype
---  -
0   emp id   7 non-null       int64
1   name     7 non-null       object
2   salary   7 non-null       int64
dtypes: int64(2), object(1)
memory usage: 296.0+ bytes
```

```
df.salary
```

	salary
0	5000
1	6000
2	7000
3	5000
4	8000
5	3000
6	6000

```
type(df.salary)
```

```
pandas.core.series.Series
def __init__(data=None, index=None, dtype: Dtype | None=None, name=None, copy: bool | None=None,
fastpath: bool=False) -> None

One-dimensional ndarray with axis labels (including time series).

Labels need not be unique but must be a hashable type. The object
supports both integer- and label-based indexing and provides a host of
methods for performing operations involving the index. Statistical
```

```
df.salary.mean()
```

```
5714.285714285715
```



```
df.salary.median()
```

```
6000.0
```

```
df.salary.mode()
```

```
salary
0    5000
1    6000
```

```
df.salary.var()
```

```
2571428.5714285714
```

```
df.salary.std()
```

```
1603.5674514745463
```

```
df.describe()
```

```
emp id    salary
count  7.000000    7.000000
mean    4.000000  5714.285714
std     2.160247  1603.567451
min     1.000000   3000.000000
25%     2.500000   5000.000000
50%     4.000000   6000.000000
75%     5.500000   6500.000000
max     7.000000   8000.000000
```

```
df.describe(include='all')
```

```
emp id    name    salary
count  7.000000      7    7.000000
unique    NaN      6     NaN
top      NaN  SRI HARSHAVARDHANAN R    NaN
freq      NaN      2     NaN
mean    4.000000      NaN  5714.285714
std     2.160247      NaN  1603.567451
min     1.000000      NaN   3000.000000
25%     2.500000      NaN   5000.000000
50%     4.000000      NaN   6000.000000
75%     5.500000      NaN   6500.000000
max     7.000000      NaN   8000.000000
```

```
empCol=df.columns
```

```
empCol
```

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

```
emparray=df.values
```

```
emparray
```

```
array([[1, 'SREE VARSSINI K S', 5000],
       [2, 'SREEMATHI B', 6000],
       [3, 'SREYA G', 7000],
       [4, 'SREYASKARI MULLAPUDI', 5000],
       [5, 'SRI AKASH U G', 8000],
```

```
[6, 'SRI HARSHAVARDHANAN R', 3000],  
[7, 'SRI HARSHAVARDHANAN R', 6000]], dtype=object)
```

```
employee_DF=pd.DataFrame(emparray,columns=empCol)
```

```
employee_DF
```



	emp id	name	salary
0	1	SREE VARSSINI K S	5000
1	2	SREEMATHI B	6000
2	3	SREYA G	7000
3	4	SREYASKARI MULLAPUDI	5000
4	5	SRI AKASH U G	8000
5	6	SRI HARSHAVARDHANAN R	3000
6	7	SRI HARSHAVARDHANAN R	6000

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

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

```
array([27, 50, 44, 6, 58, 61, 23, 86, 67, 20, 75, 7, 79, 61, 90, 54])
```

```
array.mean()
```

```
50.5
```

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

```
26.0
```

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

```
56.0
```

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

```
69.0
```

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

```
90.0
```

```
#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
```

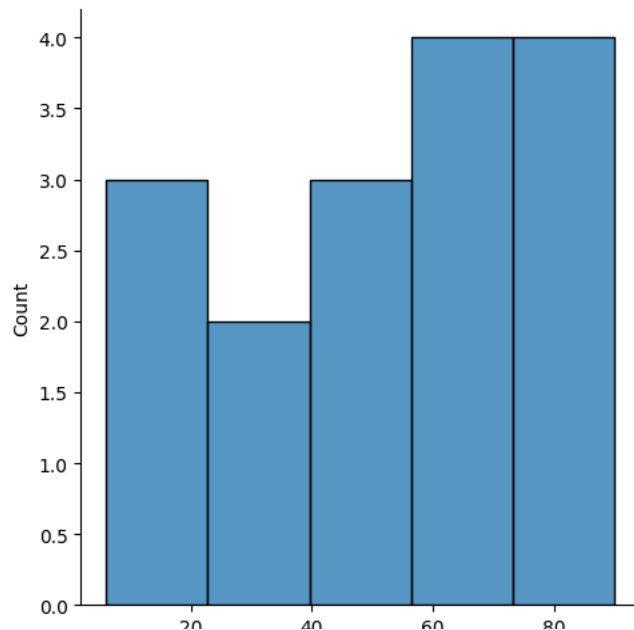
```
lr,ur=outDetection(array)
```

```
lr,ur
```


```
(-38.5, 133.5)
```

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

```
<seaborn.axisgrid.FacetGrid at 0x78f3291c2710>
```



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

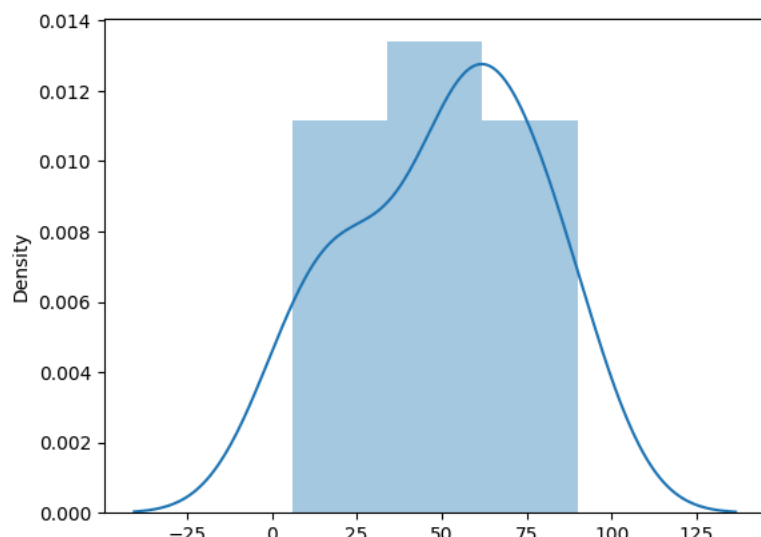
 <ipython-input-19-d72101983c40>:1: UserWarning:

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


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

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

```
sns.distplot(array)
<Axes: ylabel='Density'>
```

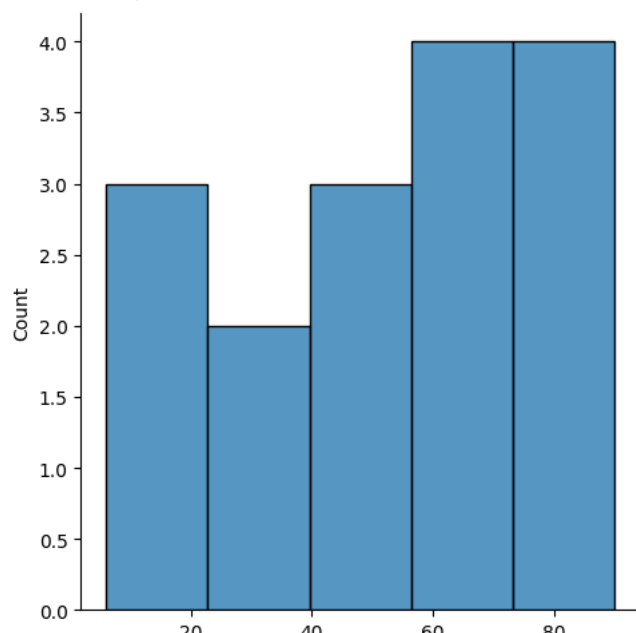


```
new_array=array[(array>lr) & (array<ur)]
new_array
```


 array([27, 50, 44, 6, 58, 61, 23, 86, 67, 20, 75, 7, 79, 61, 90, 54])

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


 <seaborn.axisgrid.FacetGrid at 0x78f2e09bb580>



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

 (-38.5, 133.5)

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

 array([27, 50, 44, 6, 58, 61, 23, 86, 67, 20, 75, 7, 79, 61, 90, 54])

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



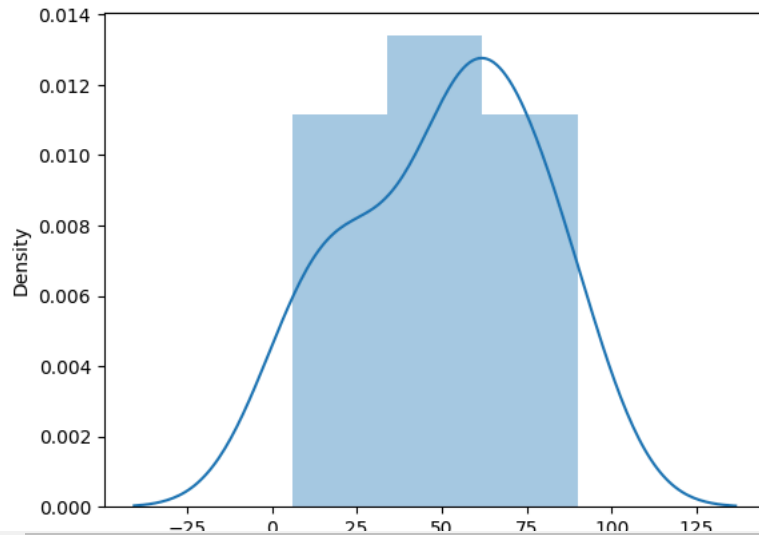
<ipython-input-18-7ba96ada5b76>:1: UserWarning:

`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: ylabel='Density'>
```



Exp:6 Handling Missing and Inappropriate Data in a Dataset

Aim: Demonstrate an experiment to handle missing data and inappropriate data in a Data set using Python Pandas Library for Data Preprocessing.

Dataset Given:

Hotel.csv

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

About Dataset:

No.of Columns =9 (called as series – CustomerID, Age_Group, Rating(1-5),Hotel, FoodPreference, Bill, NoOfPax, EstimatedSalary)

CustomerID: Numerical Continuous data

Age: Categorical Data

Rating (1-5): Numerical Discrete Data

Hotel: Categorical Data

Food: Categorical Data

Bill: Numerical Continuous data

NoOfPax: Numerical Discrete

EstimatedSalary: Numerical Continuous data

Python Code:

```
# Upload Hotel.csv and convert it into DataFrame
```

```
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

```
#From the dataframe identify the duplicate row(i.e row 9)
```

```
# The duplicated() method returns a Series with True and False values that describe which rows in the DataFrame are duplicated and not.
```

```
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
```

```
# The info() method prints information about the DataFrame. The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values).
```

```
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: 924.0+ bytes

```

The drop_duplicates() method removes duplicate rows.

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
10	10	30-35	5	RedFox	non-Veg	-6755	4	87777	30-35

#While removing duplicate record row index also removed

The len() function to return the length of an object. With a dataframe, the function returns the number of rows.

len(df)

10

#Reset the index

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])
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

Axis refers to the dimensions of a DataFrame (index and columns) or Series (index only) Use axis=0 to apply functions row-wise along the index. Use axis=1 to apply functions column-wise across columns.

```
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

The function . loc is typically used for label indexing and can access multiple columns.

```
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\Ayyadurai\AppData\Local\Temp\ipykernel_5300\2580639570.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy
    df.CustomerID.loc[df.CustomerID<0]=np.nan
df.Bill.loc[df.Bill<0]=np.nan
C:\Users\Ayyadurai\AppData\Local\Temp\ipykernel_5300\2580639570.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
df.EstimatedSalary.loc[df.EstimatedSalary<0]=np.nan
```

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

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

df

```
C:\Users\Ayyadurai\AppData\Local\Temp\ipykernel_5300\2129877948.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy
    df['NoOfPax'].loc[(df['NoOfPax']<1) | (df['NoOfPax']>20)]=np.nan
```

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

Using the inplace=True keyword in a pandas method changes the default behaviour such that the operation on the dataframe doesn't return anything, it instead 'modifies the underlying data'

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)

Fillna is a Pandas function to fill the NA/NaN values with the specified method.

If column or feature is numerical continuous data then replace the missing(NaN) value by taking mean value.

If column or feature is numerical discrete data then replace the missing(NaN) value by taking median value.

If column or feature is non-numerical i.e Categorical data then replace the missing(NaN) value by taking mode value.

```
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	4.0	Ibis	Veg	1300.0	2.0	40000.0
1	2.0	30-35	5.0	LemonTree	Non-Veg	2000.0	3.0	59000.0
2	3.0	25-30	4.0	RedFox	Veg	1322.0	2.0	30000.0
3	4.0	20-25	4.0	LemonTree	Veg	1234.0	2.0	120000.0
4	5.0	35+	3.0	Ibis	Veg	989.0	2.0	45000.0
5	6.0	35+	3.0	Ibis	Non-Veg	1909.0	2.0	122220.0
6	7.0	35+	4.0	RedFox	Veg	1000.0	2.0	21122.0
7	8.0	20-25	4.0	LemonTree	Veg	2999.0	2.0	345673.0
8	9.0	25-30	2.0	Ibis	Non-Veg	3456.0	3.0	96755.0
9	10.0	30-35	5.0	RedFox	Non-Veg	1801.0	4.0	87777.0


```
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()
```




Country
0 France

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



--

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



```
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 is not None, the resulting Series is reindexed 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
    The name of the 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	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.

```
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()
```

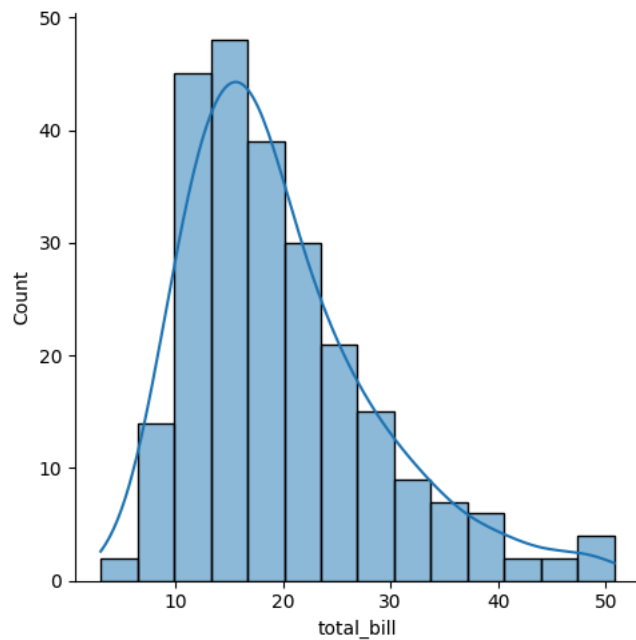
```
↗
```

	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

[+ Code](#)[+ Text](#)

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

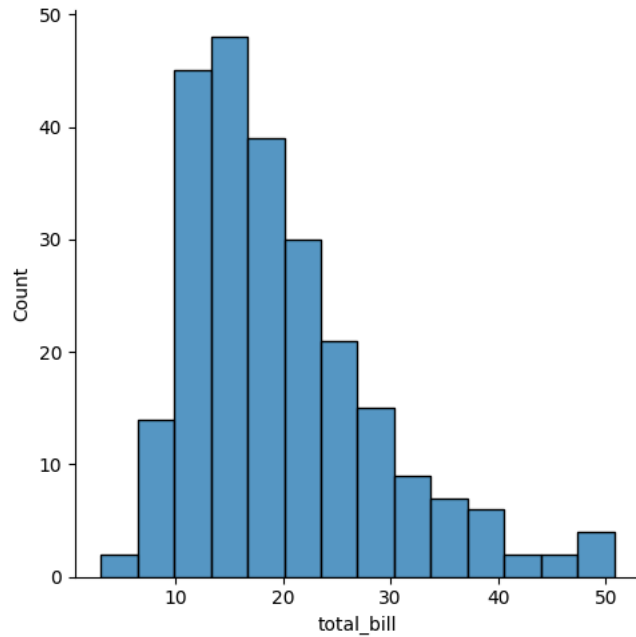
```
↗ <seaborn.axisgrid.FacetGrid at 0x79bb4c7ea680>
```



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

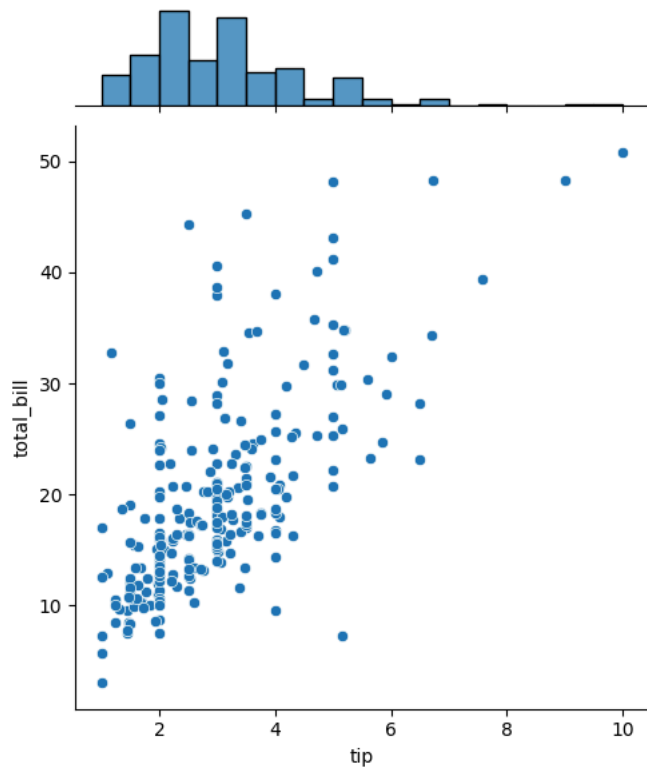


```
<seaborn.axisgrid.FacetGrid at 0x79bb0b0af580>
```




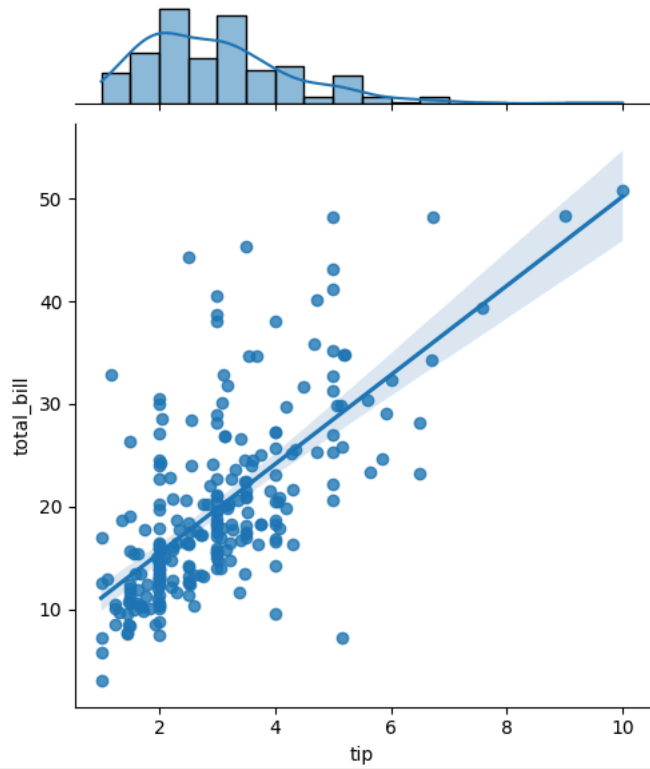
```
sns.jointplot(x=tips.tip,y=tips.total_bill)
```

```
<seaborn.axisgrid.JointGrid at 0x79bb08fc96c0>
```




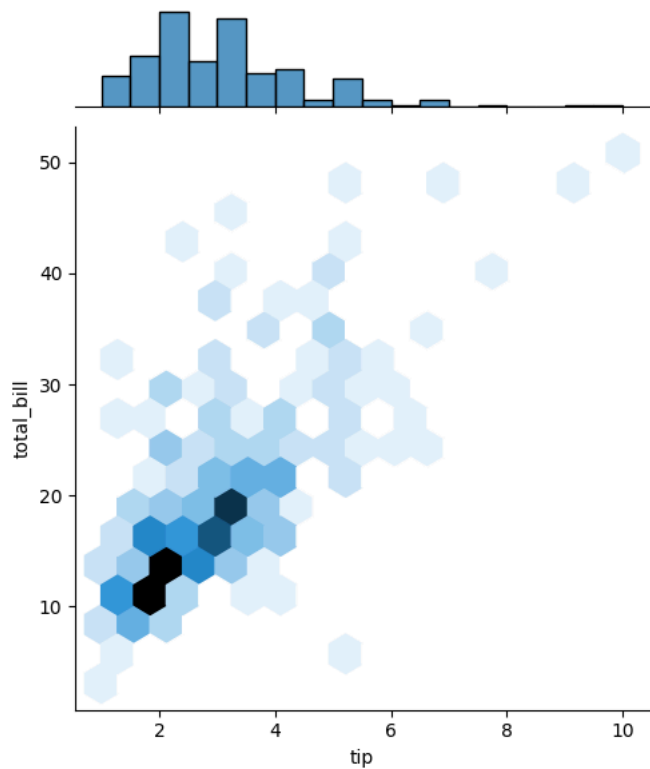
```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="reg")
```

 <seaborn.axisgrid.JointGrid at 0x79bb08fc9cf0>



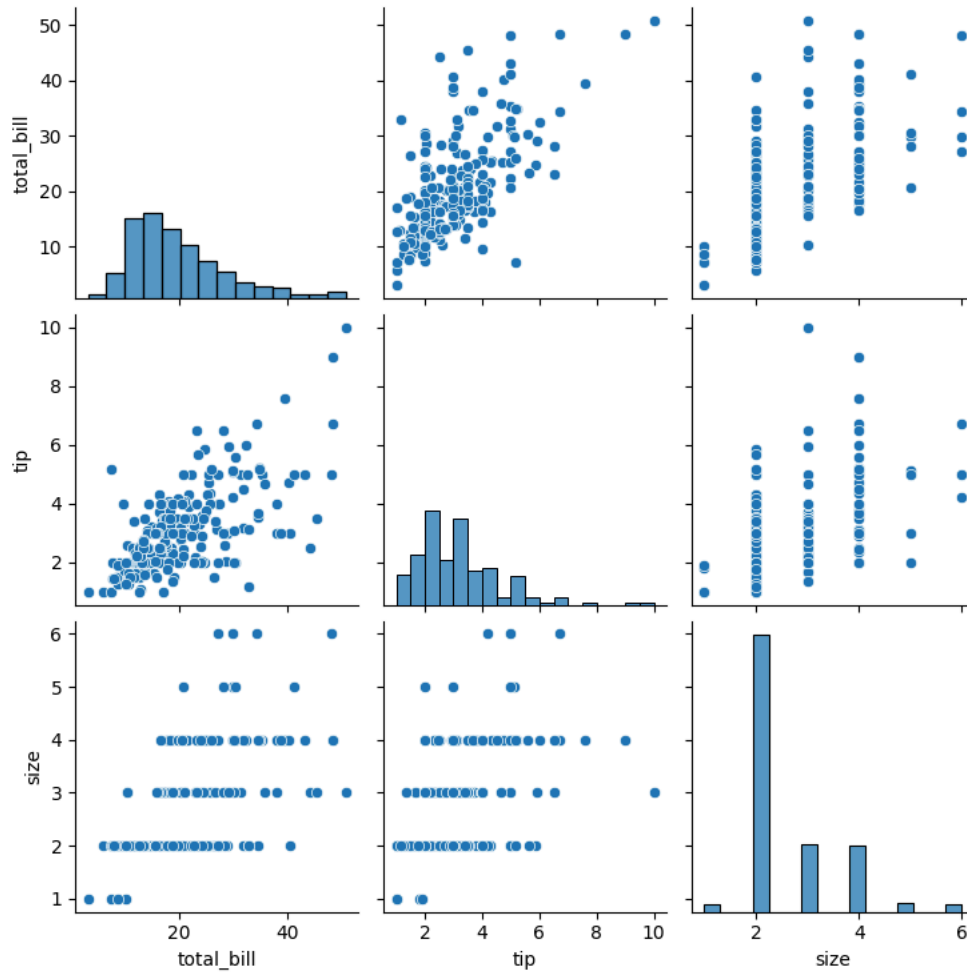
```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="hex")
```

 <seaborn.axisgrid.JointGrid at 0x79bb088f4730>



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

```
<seaborn.axisgrid.PairGrid at 0x79bb06fc3d30>
```



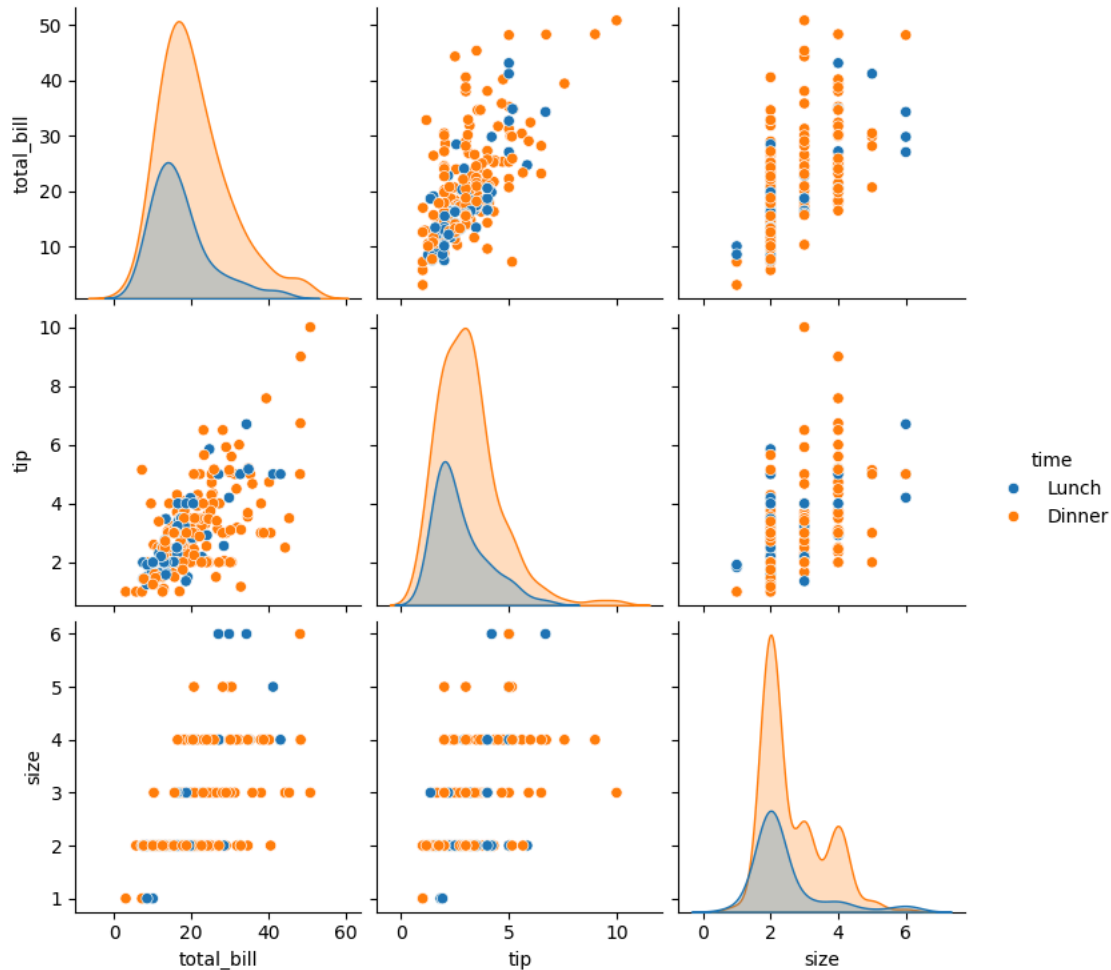
```
tips.time.value_counts()
```

```
count
time
Dinner    176
Lunch      68
```

```
dtype: int64
```

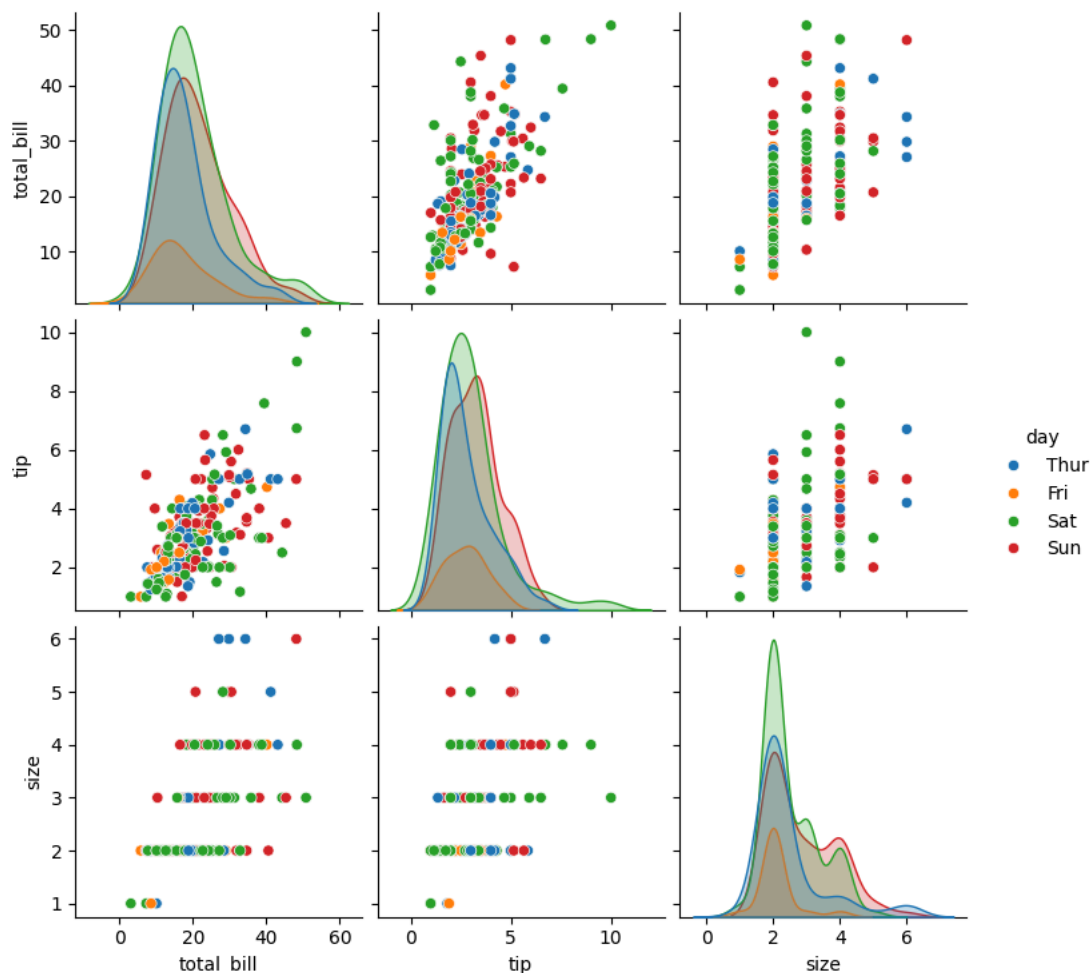
```
sns.pairplot(tips,hue='time')
```

<seaborn.axisgrid.PairGrid at 0x79bb088f4670>



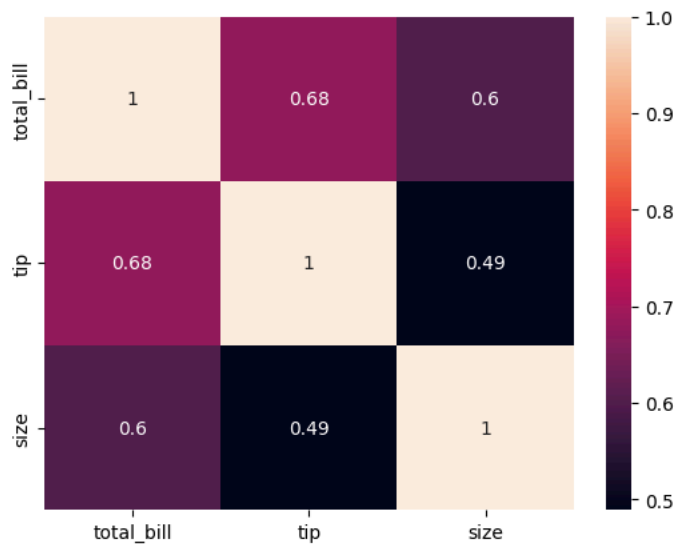
```
sns.pairplot(tips,hue='day')
```

```
<seaborn.axisgrid.PairGrid at 0x79bb08f1f6a0>
```



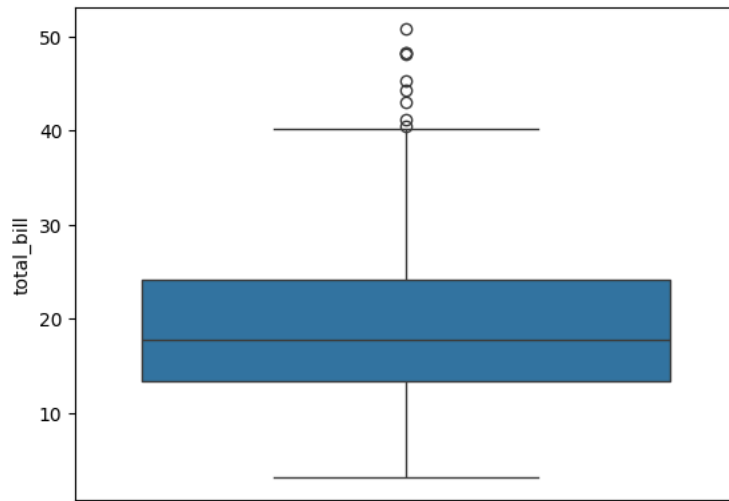
```
sns.heatmap(tips.corr(numeric_only=True),annot=True)
```

```
<Axes: >
```



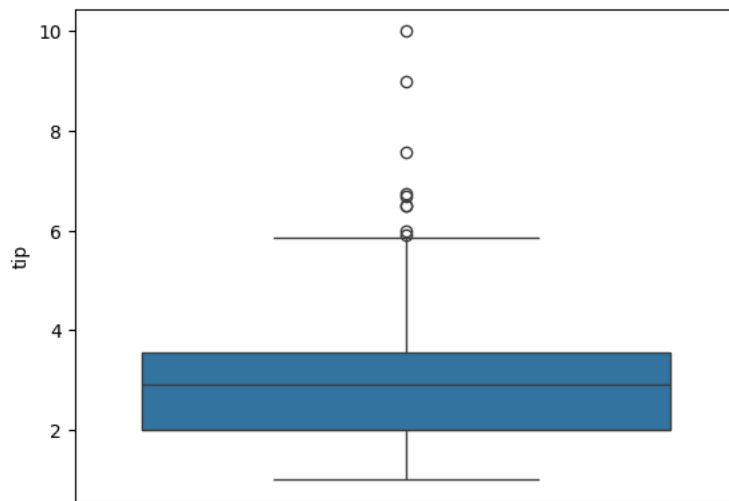
```
sns.boxplot(tips.total_bill)
```

```
<Axes: ylabel='total_bill'>
```



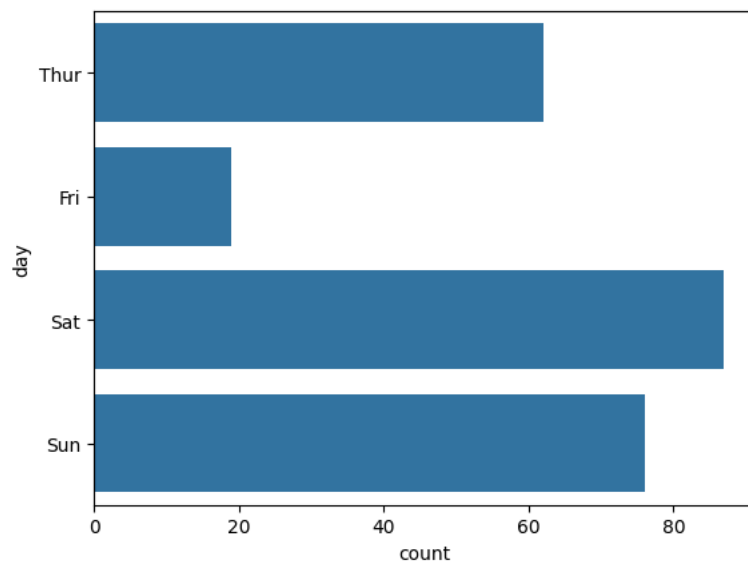
```
sns.boxplot(tips.tip)
```

```
<Axes: ylabel='tip'>
```




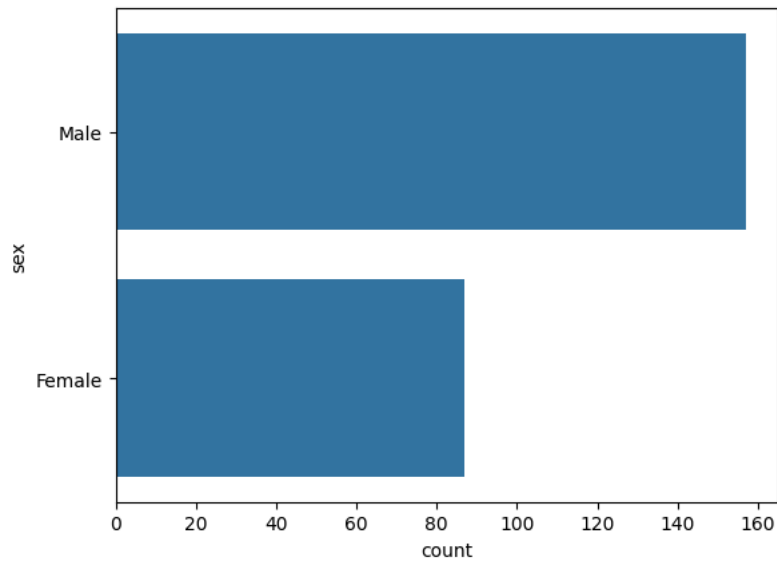
```
sns.countplot(tips.day)
```

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




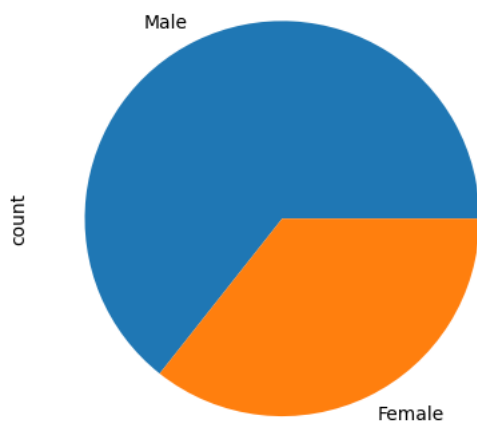
```
sns.countplot(tips.sex)
```

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




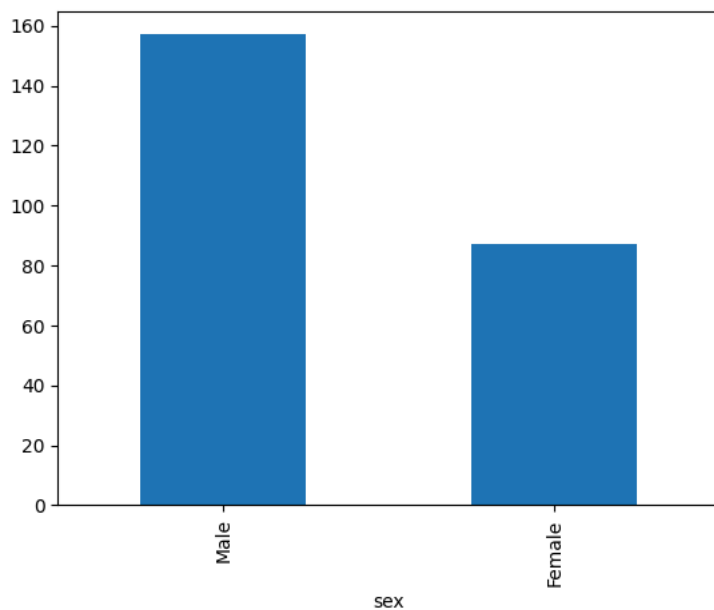
```
tips.sex.value_counts().plot(kind='pie')
```

 <Axes: ylabel='count'>




```
tips.sex.value_counts().plot(kind='bar')
```

 <Axes: xlabel='sex'>



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

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



Exp No:12 Hypothetical using Z-Test

Objective: To test whether the average weight of a species of birds differs from 150 grams.

Procedure:

1. **Null Hypothesis (H_0):** The average weight of the birds is 150 grams.
2. **Alternative Hypothesis (H_1):** The average weight of the birds is not 150 grams.
3. **Sample:** Measure the weights of 30 birds randomly selected from the population.
4. **Z-Test:** Conduct a Z-test to compare the sample mean to 150 grams.
5. **Decision Rule:** Use a significance level of $\alpha = 0.05$.

Python Code Example:

python

Copy code

```
import numpy as np

import scipy.stats as stats

# Define the sample data (hypothetical weights in grams)

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

# Population mean under the null hypothesis

population_mean = 150

# Calculate sample statistics

sample_mean = np.mean(sample_data)

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


# Number of observations

n = len(sample_data)

# Calculate the Z-statistic

z_statistic = (sample_mean - population_mean) / (sample_std /
np.sqrt(n))


# Calculate the p-value
```

```
p_value = 2 * (1 - stats.norm.cdf(np.abs(z_statistic))) # Two-tailed test
```

```
# Print results
```

```
print(f"Sample Mean: {sample_mean:.2f}")
```

```
print(f"Z-Statistic: {z_statistic:.4f}")
```

```
print(f"P-Value: {p_value:.4f}")
```

```
# Decision based on the significance level
```

```
alpha = 0.05
```

```
if p_value < alpha:
```

```
    print("Reject the null hypothesis: The average weight is significantly different from 150 grams.")
```

```
else:
```

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

OUTPUT:

Sample Mean: 150.20

Z-Statistic: 0.6406

P-Value: 0.5218

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

Exp No:13 Hypothetical using T-Test

Objective: To test whether the average IQ score of a sample of students differs significantly from a population mean IQ score of 100.

Procedure:

1. **Null Hypothesis (H_0):** The average IQ score of the sample is 100.
2. **Alternative Hypothesis (H_1):** The average IQ score of the sample is not 100.
3. **Sample:** Measure the IQ scores of 25 randomly selected students.
4. **T-Test:** Conduct a one-sample T-test to compare the sample mean to 100.
5. **Decision Rule:** Use a significance level of $\alpha = 0.05$.

Python Code Example:

python

Copy code

```
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.")
```

OUTPUT:

```
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.
```

RESULT

Exp No:13 Hypothetical using ANOVA-Test

Objective: To compare the growth rates of plants under three different fertilizer treatments (Treatment A, B, and C) to determine if there is a significant difference in their mean growth.

Procedure:

1. **Null Hypothesis (H_0):** The mean growth rates of plants under all three fertilizer treatments are equal.
2. **Alternative Hypothesis (H_1):** At least one pair of mean growth rates of plants under different fertilizer treatments are not equal.
3. **Samples:**
 - Measure the growth (in centimeters) of 25 plants under Treatment A.
 - Measure the growth (in centimeters) of 25 plants under Treatment B.
 - Measure the growth (in centimeters) of 25 plants under Treatment C.
4. **ANOVA:** Conduct a one-way ANOVA to compare the mean growth rates of plants across the three fertilizer treatments.
5. **Decision Rule:** Use a significance level of $\alpha = 0.05$.

Python Code Example:

```
python
Copy code
import numpy as np
import scipy.stats as stats

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

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

```

# Growth data (in cm) for Treatment A, B, and C
growth_A = np.random.normal(loc=10, scale=2, size=n_plants)
growth_B = np.random.normal(loc=12, scale=3, size=n_plants)
growth_C = np.random.normal(loc=15, scale=2.5, size=n_plants)

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

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

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

# Print results
print("Treatment A Mean Growth:", np.mean(growth_A))
print("Treatment B Mean Growth:", np.mean(growth_B))
print("Treatment C Mean Growth:", np.mean(growth_C))
print()
print(f"F-Statistic: {f_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")

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

# Additional: Post-hoc analysis (Tukey's HSD) if ANOVA is
significant
if p_value < alpha:
    from statsmodels.stats.multicomp import pairwise_tukeyhsd

    tukey_results = pairwise_tukeyhsd(all_data, treatment_labels,
alpha=0.05)
    print("\nTukey's HSD Post-hoc Test:")
    print(tukey_results)

```

OUTPUT:

```

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

```

F-Statistic: 36.1214

P-Value: 0.0000

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


```
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



Next steps:

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

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



Next steps:

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

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

 <ipython-input-5-20665a0bbaa1>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame c
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate ob
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inpla

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



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

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

```
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
```

```
SimpleImputer()
```

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



```
SimpleImputer
```

```
SimpleImputer()
```

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

```
from sklearn.preprocessing import OneHotEncoder
```

```
oh = OneHotEncoder(sparse_output=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.]])
```



```
[1., 0., 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, 63777.777777777778],
        [1.0, 0.0, 0.0, 35.0, 58000.0],
        [0.0, 0.0, 1.0, 38.77777777777778, 52000.0],
        [1.0, 0.0, 0.0, 48.0, 79000.0],
        [1.0, 0.0, 0.0, 50.0, 83000.0],
        [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

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

```
feat_standard_scaler
```

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

```
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],
        [1.         , 0.         , 0.         , 1.         , 1.         ],
        [1.         , 0.         , 0.         , 0.43478261, 0.54285714]])
```

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


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

```
In [19]: 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: 612.0 bytes
```

```
In [3]: df.dropna(inplace=True)
```

```
In [4]: 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: 612.0 bytes
```

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

```
Out[5]:
```

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

```
In [6]: features=df.iloc[:,[0]].values
label=df.iloc[:,[1]].values
```

```
In [7]: 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_st
```

```
In [20]: from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
```

```
Out[20]:
```

LinearRegression

LinearRegression()

```
In [21]: model.score(x_train,y_train)
```

```
Out[21]: 0.9603182547438908
```

```
In [23]: model.score(x_test,y_test)
```

```
Out[23]: 0.9184170849214232
```

```
In [24]: model.coef_
```

```
Out[24]: array([[9281.30847068]])
```

```
In [25]: model.intercept_
```

```
Out[25]: array([27166.73682891])
```

```
In [26]: import pickle  
pickle.dump(model,open('SalaryPred.model','wb'))
```

```
In [27]: model=pickle.load(open('SalaryPred.model','rb'))
```

```
In [28]: 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

```
In [ ]:
```

```
In [29]: print("Estimated Salary for {} years of experience is {}: " .format(yr_of_exp,Salary))
```

Estimated Salary for 44.0 years of experience is [[435544.30953887]]:

```
In [ ]:
```

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

```
Out[1]:
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

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

```
Out[2]:
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

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

```
Out[4]: 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],
 [ 12, 20000]])
```

```
In [5]: label
```

```
Out[5]: array([0, 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, 1, 0, 0, 0, 1, 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, 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, 0, 1, 0, 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, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
               0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
               1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
               1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
               0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
               1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
               0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
               1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
               0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
               1, 1, 0, 1], dtype=int64)
```

```
In [6]: from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LogisticRegression
```

```
In [7]: for i in range(1,401):
        x_train,x_test,y_train,y_test=train_test_split(features,label,test_size=0.2,
        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 32
Test 0.6625 Train0.6375 Random State 33
Test 0.6625 Train0.6375 Random State 34
Test 0.6625 Train0.6375 Random State 35
Test 0.6625 Train0.6375 Random State 36
Test 0.6625 Train0.6375 Random State 37
Test 0.6625 Train0.6375 Random State 38
Test 0.6625 Train0.6375 Random State 39
Test 0.6625 Train0.6375 Random State 40
```

```
In [8]: x_train,x_test,y_train,y_test=train_test_split(features,label,test_size=0.2,
        finalModel=LogisticRegression()
        finalModel.fit(x_train,y_train)
```

Out[8]: LogisticRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [9]: print(finalModel.score(x_train,y_train))
        print(finalModel.score(x_test,y_test))
```

```
0.834375
0.9125
```

```
In [10]: 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	0.84	0.71	0.77	143
accuracy			0.85	400
macro avg	0.85	0.82	0.83	400
weighted avg	0.85	0.85	0.85	400

In []: