RAJALAKSHMI ENGINEERING COLLEGE

Approved by AICTE | Affiliated to Anna University | Accredited by NAAC

Department of Computer Science and Engineering

CS23334 Fundamentals of Data Science Lab

III semester II Year (2023R)

Name of the Student : GOWTHAM D

Register Number : 2116240701154

EXPERIMENT NO: 1A

Analyze the trend of data science job postings over the last decade

Aim:

To visualize the **trend of Data Science job postings from 2014 to 2023** using a line graph in Python with the help of **Pandas**, **Matplotlib**, and **Seaborn** libraries.

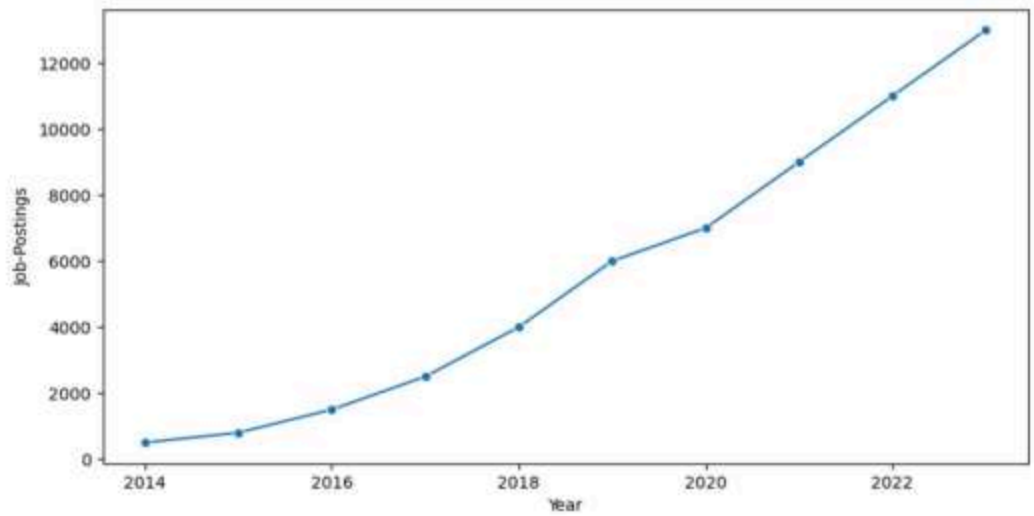
Algorithm:

- Import pandas, matplotlib.pyplot, and seaborn.
- Store data of *Years* and *Job Postings* in a dictionary.
- Use pd.DataFrame() to convert the dictionary into a DataFrame.
- Use sns.lineplot() to plot a line graph between Year (x-axis) and Job Postings (y-axis).
- Add the graph title, x-label, y-label, and grid.
- Use `plt.show()` to display the plotted graph.
-

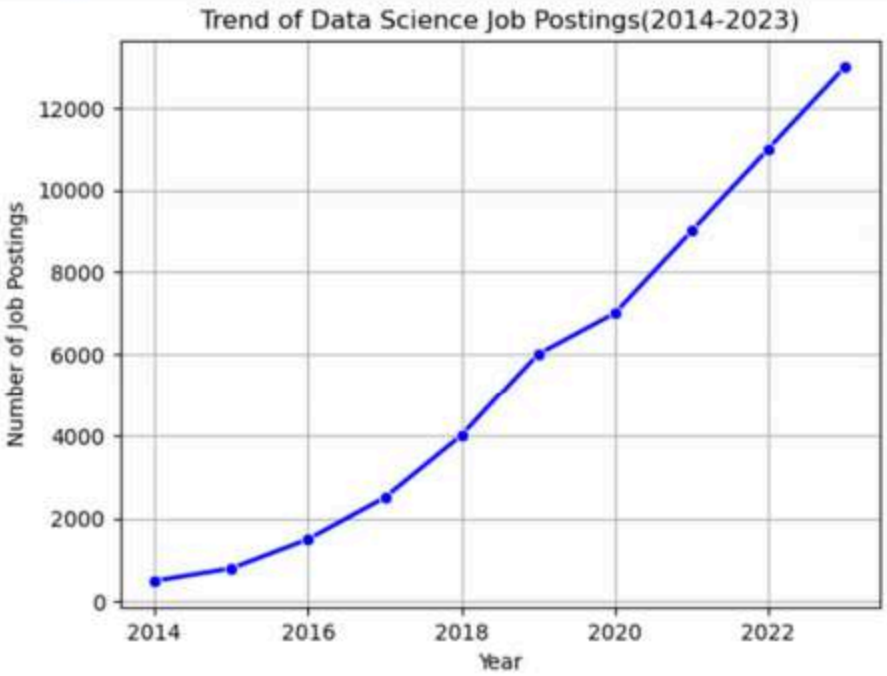
Program:

```
[15]: import pandas as pd
      import matplotlib.pyplot as plt
      import seaborn as sns
      data={
          'Year':[2014,2015,2016,2017,2018,2019,2020,2021,2022,2023],
          'Job-Postings':[500,800,1500,2500,4000,6000,7000,9000,11000,13000]
      }
      df=pd.DataFrame(data)
      plt.figure(figsize=(10,5))
      sns.lineplot(x='Year',y='Job-Postings',data=df,marker='o')
```

```
[15]: <Axes: xlabel='Year', ylabel='Job-Postings'>
```



```
[13]: sns.lineplot(x='Year', y='Job-Postings', data=df, marker='o', color='blue', linewidth=2)
      plt.title("Trend of Data Science Job Postings(2014-2023)")
      plt.xlabel('Year')
      plt.ylabel('Number of Job Postings')
      plt.grid(True)
      plt.show()
```



Result:

The program successfully displays a **line graph** showing a **steady increase in Data Science job postings** from **2014 (500)** to **2023 (13,000)**.

EXPERIMENT NO: 1B

Analyze and visualize the distribution of various data science roles

Aim:

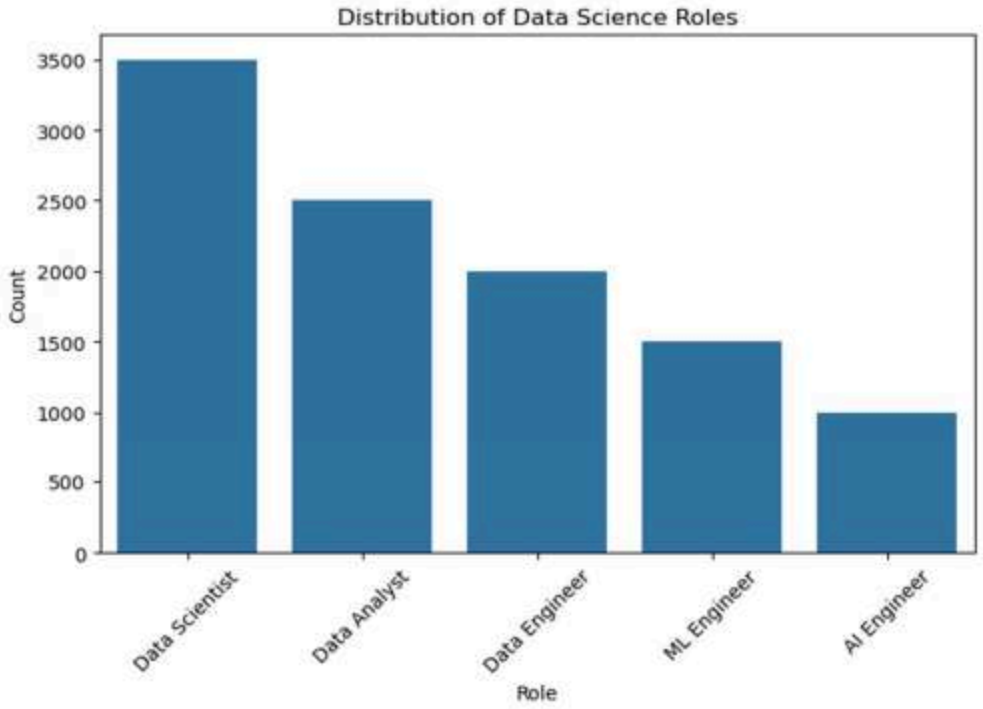
To visualize the distribution of different **Data Science roles** and their **counts** using bar and pie charts with Python libraries — **Pandas**, **Matplotlib**, and **Seaborn**.

Algorithm:

1. Import the necessary libraries: **pandas**, **matplotlib.pyplot**, and **seaborn**.
2. Create a dataset containing roles and their respective counts.
3. Convert the dataset into a **DataFrame** using pd.DataFrame().
4. Plot a **bar chart** using sns.barplot() to show role distribution.
5. Plot a **pie chart** using plt.pie() to show percentage distribution.
6. Add titles and rotate axis labels for better readability.
7. Display both charts using plt.show().

Program:

```
[19]: import pandas as pd
      import matplotlib.pyplot as plt
      import seaborn as sns
      data={
          'Role':['Data Scientist','Data Analyst','Data Engineer','ML Engineer','AI Engineer'],
          'Count':[3500,2500,2000,1500,1000]
      }
      df=pd.DataFrame(data)
      plt.figure(figsize=(8,5))
      sns.barplot(x='Role',y='Count',data=df)
      plt.title("Distribution of Data Science Roles")
      plt.xticks(rotation=45)
      plt.show()
```



Result:

The program successfully displays a bar chart and a pie chart showing that **Data Scientist** has the highest count and **AI Engineer** has the lowest count among the data science roles.

EXPERIMENT NO: 1C

Conduct an experiment to differentiate Structured , Un-structured and Semi structured data

Aim:

To demonstrate the three types of data — **structured, semi-structured, and unstructured** — and display them in Python.

Algorithm:

1. **Import pandas** for handling structured data.
2. **Create structured data** using pd.DataFrame with columns Emp_id, Name, Salary.
3. **Print structured data**.
4. **Create semi-structured data** as a list of dictionaries with Emp_id, Name, and nested Skills.
5. **Print semi-structured data**.
6. **Create unstructured data** as a list of plain text strings describing employees.
7. **Print unstructured data**.

Program:

```
[2]: import pandas as pd
structured_data=pd.DataFrame({
    'Emp_id':[1,2],
    'Name':['Alice','Bob'],
    'Salary':[50000,60000]
})
print('Structured Data: \n',structured_data)
```

```
Structured Data:
    Emp_id   Name  Salary
0       1  Alice   50000
1       2    Bob   60000
```

```
[5]: semi_structured_data=[
    {"Emp_id":1,"Name":"Alice","Skills":["Python","SQL"]},
    {"Emp_id":2,"Name":"Bob","Skills":["Java","Spark"]}
]
print("\nSemi Structured Data: \n",semi_structured_data)
```

```
Semi Structured Data:
 [{'Emp_id': 1, 'Name': 'Alice', 'Skills': ['Python', 'SQL']}, {'Emp_id': 2, 'Name': 'Bob', 'Skills': ['Java', 'Spark']}]
```

```
[3]: unstructured_data=[
    "Alice joined the company in 2020 and works on python and sql.",
    "Bob is skilled in java and spark."
]
print("\nUnstructured Data: \n",unstructured_data)
```

```
Unstructured Data:
 ['Alice joined the company in 2020 and works on python and sql.', 'Bob is skilled in java and spark.']
```

Result:

Data can be **structured**, **semi-structured**, or **unstructured**. Structured data is organized in tables, semi-structured data has some organization like JSON, and unstructured data is plain text without a fixed format.

EXPERIMENT NO: 1D

Conduct an experiment to encrypt and decrypt given sensitive data.

Aim:

To securely **encrypt and decrypt a sensitive message** (like a password) using symmetric encryption with Fernet from the cryptography library.

Algorithm:

1. **Generate a key using Fernet.generate_key().**
2. **Create a Fernet cipher with the generated key.**
3. **Convert the data (string) to bytes.**
4. **Encrypt the data using cipher.encrypt().**
5. **Decrypt the data using cipher.decrypt().**
6. **Convert bytes back to string to view the original message.**

Program:

```
[5]: from cryptography.fernet import Fernet
     key = Fernet.generate_key()
     print("Generated Key:", key.decode())

     Generated Key: FeC5LlsCxE6ccLsHYpbDYsujt1Rr5XxGFLd5aaheo4E=

[6]: cipher = Fernet(key)
     data = "My bank account password is 1234".encode()
     print("\nOriginal Data:", data.decode())

     Original Data: My bank account password is 1234

[7]: encrypted_data = cipher.encrypt(data)
     print("Encrypted Data:", encrypted_data.decode())

     Encrypted Data: gAAAAABoGkT8215UKLv27YBvLjNrir6qSKoq2b1AyepvZecs719vcpKzNarqcNf84fjUVMvjK9GgeVDH59fOpupGcgf4DUhBkbT6KYEmchQMM-F4H9MhTdYC4UAyvqc1pxsKTTrw7
     mQH

[8]: decrypted_data = cipher.decrypt(encrypted_data)
     print("Decrypted Data:", decrypted_data.decode())

     Decrypted Data: My bank account password is 1234
```

Result:

The program aims to securely encrypt and decrypt a message using Fernet symmetric encryption. It generates a key, converts the message into bytes, encrypts it into unreadable data, and then decrypts it back to the original message. The result shows that the original message is safely restored after encryption.

EXPERIMENT NO: 2

Upload and Analyze the data set given in csv format and perform data preprocessing and visualization

Aim:

To analyze and visualize sales data, clean missing values, summarize total sales and quantities per product, and examine correlations between numeric variables.

Algorithm:

1. **Import libraries: pandas, numpy, matplotlib, seaborn.**
2. **Load the CSV file into a DataFrame.**
3. **Clean data: convert columns to numeric, fill or drop missing values.**
4. **Group data by product to calculate total sales and quantity.**
5. **Plot a bar chart of total sales per product.**
6. **Create a pivot table showing sales by region and product.**
7. **Compute correlation matrix and plot a heatmap.**

Program:

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
file_path = 'C:/Users/vijay/Downloads/sales_data.csv'
df = pd.read_csv(file_path)
print(df.head())
```

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

[3]:
```
print(df.isnull().sum())
```

```
Date        0
Product     0
Sales       0
Quantity    0
Region      0
dtype: int64
```

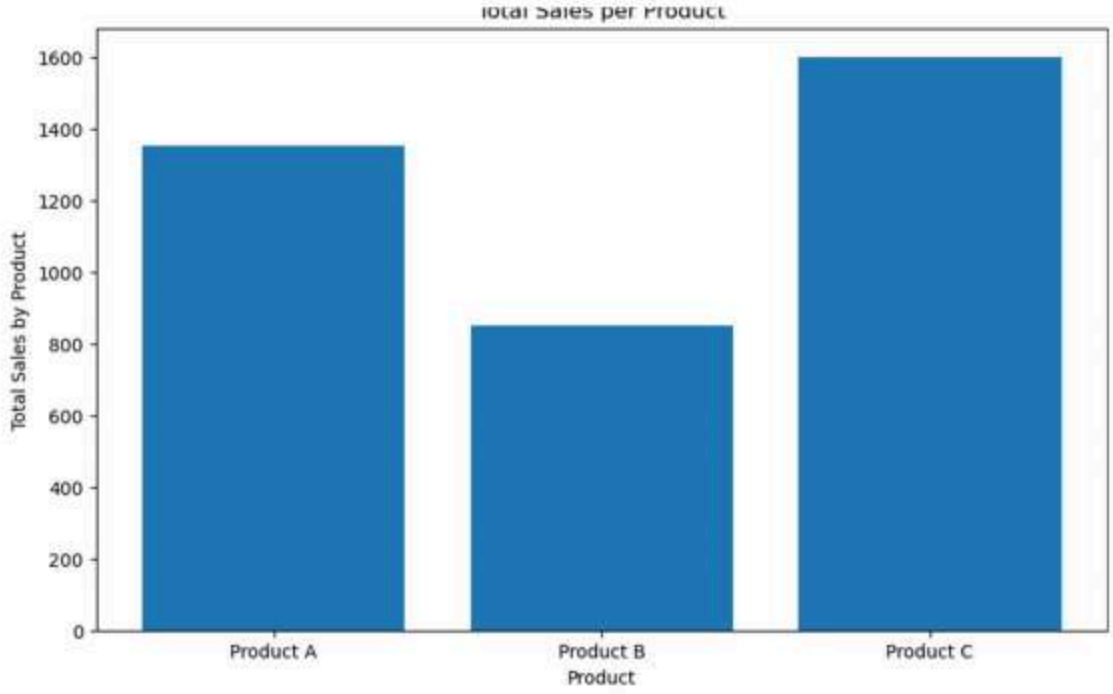
[4]:
```
df['Sales'] = pd.to_numeric(df['Sales'], errors='coerce')
df['Quantity'] = pd.to_numeric(df['Quantity'], errors='coerce')
df['Sales'].fillna(df['Sales'].mean(), inplace=True)
df.dropna(subset=['Product', 'Quantity', 'Region'], inplace=True)
print(df.describe())
```

```
            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
```

[5]:
```
product_summary = df.groupby('Product').agg({
    'Sales': 'sum',
    'Quantity': 'sum'
}).reset_index()
print(product_summary)
```

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

[8]:
```
plt.figure(figsize=(10, 6))
plt.bar(product_summary['Product'], product_summary['Sales'])
plt.xlabel('Product')
plt.ylabel('Total Sales by Product')
plt.title('Total Sales per Product')
plt.show()
```



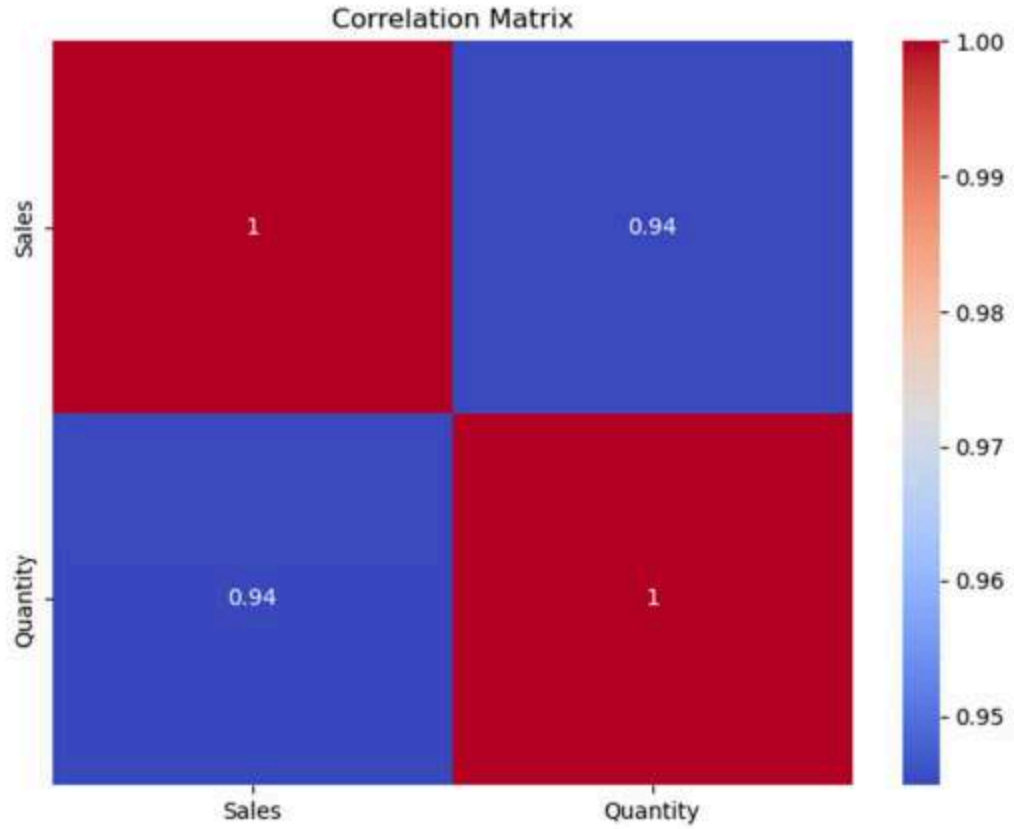
[10]:
```
pivot_table = df.pivot_table(
    values='Sales',
    index='Region',
    columns='Product',
    aggfunc=np.sum,
    fill_value=0
)
print(pivot_table)
```

Product	Product A	Product B	Product C
Region			
East	0	0	1600
North	1350	0	0
South	0	480	0
West	0	370	0

[7]:
```
correlation_matrix = df.corr(numeric_only=True)
print(correlation_matrix)
```

```
             Sales  Quantity
Sales     1.000000  0.944922
Quantity  0.944922  1.000000
```

[9]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



Result:

The dataset was cleaned, total sales and quantities per product were calculated, the bar chart highlighted top-selling products, the pivot table showed regional sales distribution, and the correlation matrix revealed a strong positive relationship between quantity and sales.

EXPERIMENT NO: 3A

Data Preprocessing and Encoding for Machine Learning

Aim:

To preprocess a dataset by handling missing values, encoding categorical variables, and preparing it for analysis or machine learning.

Algorithm:

1. **Load Data: Read CSV file into a DataFrame.**
2. **Handle Missing Values**
3. **Encode Categorical Data**
4. **Combine Data: Concatenate dummy variables with other relevant columns.**
5. **Output: Print the cleaned and encoded dataset.**

Program:

```
[3]: import numpy as np
     import pandas as pd
     df=pd.read_csv("C:/Users/vijay/Downloads/pre_process_datasample (1).csv")
     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)
     country_dummies=pd.get_dummies(df.Country)
     updated_dataset=pd.concat([country_dummies,df.iloc[:,[1,2,3]]],axis=1)
     updated_dataset.Purchased.replace(['No','Yes'],[0,1],inplace=True)
     print(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   False     True  False  50.0  83000.0          0
9    True    False  False  37.0  67000.0          1
```

Result:

A cleaned and transformed dataset where all missing values are replaced, categorical variables are converted into numeric form, and the Purchased column is ready for modeling. Each row now has numeric values only, suitable for machine learning or statistical analysis.

EXPERIMENT NO: 3B

Hotel Dataset Cleaning and Preprocessing

Aim:

To clean and preprocess the hotel dataset by removing duplicates, correcting invalid and inconsistent values, filling missing data, and standardizing text entries, making it ready for analysis or machine learning.

Algorithm:

1. **Load Data: Read the CSV file into a DataFrame.**
2. **Remove Duplicates: Drop duplicate rows and reset the index.**
3. **Drop Unnecessary Columns: Remove Age_Group.1.**
4. **Handle Invalid Values:**
    - **Replace negative CustomerID, Bill, and EstimatedSalary with NaN.**
    - **Replace NoOfPax values <1 or >20 with NaN.**
5. **Standardize Text Data:**
    - **Correct Hotel names (e.g., 'Ibys' → 'Ibis').**
    - **Normalize FoodPreference values ('Vegetarian'/'veg' → 'Veg', 'non-Veg' → 'Non-Veg').**
6. **Fill Missing Values:**
    - **EstimatedSalary → mean (rounded)**
    - **NoOfPax → median (rounded)**
    - **Rating(1-5) → median (rounded)**
7. **Output: Print the cleaned dataset.**

Program:

```
[3]: import numpy as np
     import pandas as pd
     df = pd.read_csv("C:/Users/vijay/Downloads/Hotel_Dataset.csv")
     df.drop_duplicates(inplace=True)
     df.reset_index(drop=True, inplace=True)
     df.drop(['Age_Group.1'], axis=1, inplace=True)
     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.loc[(df['NoOfPax'] < 1) | (df['NoOfPax'] > 20), 'NoOfPax'] = np.nan
     df.Hotel.replace(['Ibys'], 'Ibis', inplace=True)
     df.FoodPreference.replace(['Vegetarian', 'veg'], 'Veg', inplace=True)
     df.FoodPreference.replace(['non-Veg'], 'Non-Veg', inplace=True)
     df.EstimatedSalary.fillna(round(df.EstimatedSalary.mean()), inplace=True)
     df.NoOfPax.fillna(round(df.NoOfPax.median()), inplace=True)
     df['Rating(1-5)'].fillna(round(df['Rating(1-5)'].median()), inplace=True)
     print(df)
```

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

   NoOfPax  EstimatedSalary
0      2.0          40000.0
1      3.0          59000.0
2      2.0          30000.0
3      2.0         120000.0
4      2.0          45000.0
5      2.0         122220.0
6      2.0          21122.0
7      2.0         345673.0
8      3.0          96755.0
9      4.0          87777.0
```

Result:

A cleaned and consistent hotel dataset with no duplicates, corrected text values, invalid numerical entries replaced, and missing values filled, ready for analysis or modeling.

EXPERIMENT NO: 3C

Airline Flight Dataset Generation and CSV Export

Aim:

To generate a synthetic dataset of airline flights with random flight details (IDs, airlines, source and destination cities, departure/arrival times, durations, and fares) and export it as a CSV file for analysis or testing purposes.

Algorithm:

1) **Define Data Sources: Lists of airlines and cities.**
2) **Generate Records: For 15 flights:**
3) **Assign a unique Flight_ID.**
4) **Randomly select airline, source, and destination (different cities).**
5) **Generate random departure time.**
6) **Compute random flight duration and arrival time.**
7) **Generate random fare.**
8) **Store Data: Append flight details to a list.**
9) **Create DataFrame: Convert the list into a pandas DataFrame with appropriate column names.**
10) **Export CSV: Save the DataFrame to "airline.flights".**
11) **Output: Print confirmation and show the first few rows.**

Program:

```
[7]: import pandas as pd
     import random
     from datetime import datetime,timedelta
     airlines=["Air India","Indigo","Spicejet","Vistara","GoAir"]
     city=["Chennai","Delhi","Mumbai","Bangalore","Hyderabad","Kolkata","Pune"]
     data=[]
     for i in range(1,16):
         flight_id=f"FL{i:03d}"
         airline=random.choice(airlines)
         source,destination=random.sample(city,2)
         dep_time=datetime(2024,8,1,random.randint(0,23),random.choice([0,15,30,45]))
         duration_hours=random.randint(1,5)
         duration_min=random.choice([0,15,30,45])
         duration=timedelta(hours=duration_hours,minutes=duration_min)
         arr_time=dep_time+duration
         fare=random.randint(3000,16000)
         data.append([
             flight_id,airline,source,destination,
             dep_time.strftime("%Y-%m-%d %H:%M"),
             arr_time.strftime("%Y-%m-%d %H:%M"),
             f"{duration_hours}h{duration_min}m",
             fare
         ])

     df=pd.DataFrame(data,columns=[
         "Flight_ID","AirLine","Source","Destination","Departure_time","Arrival_time","Duration","Fare"
     ])
     df.to_csv("airline.flights",index=True)
     print("CSV file 'airline_flights.csv' created successfully!")
     print(df.head().to_string())
```

```
CSV file 'airline_flights.csv' created successfully!
  Flight_ID    AirLine   Source Destination    Departure_time      Arrival_time Duration   Fare
0     FL001  Air India    Delhi     Kolkata  2024-08-01 00:00  2024-08-01 02:45     2h45m   7339
1     FL002    Vistara  Kolkata       Delhi  2024-08-01 03:15  2024-08-01 06:45     3h30m   5333
2     FL003  Air India  Kolkata   Bangalore  2024-08-01 09:00  2024-08-01 11:30     2h30m   7069
3     FL004   Spicejet     Pune     Kolkata  2024-08-01 22:15  2024-08-02 02:00     3h45m   5142
4     FL005      GoAir    Delhi     Kolkata  2024-08-01 06:00  2024-08-01 08:00      2h0m  13052
```

Result:

A CSV file "airline.flights" containing 15 randomly generated flight records with columns: Flight_ID, AirLine, Source, Destination, Departure_time, Arrival_time, Duration, and Fare. The dataset can be used for analysis, testing, or simulations.

EXPERIMENT NO: 4

Outlier Detection and Visualization using IQR

Aim:

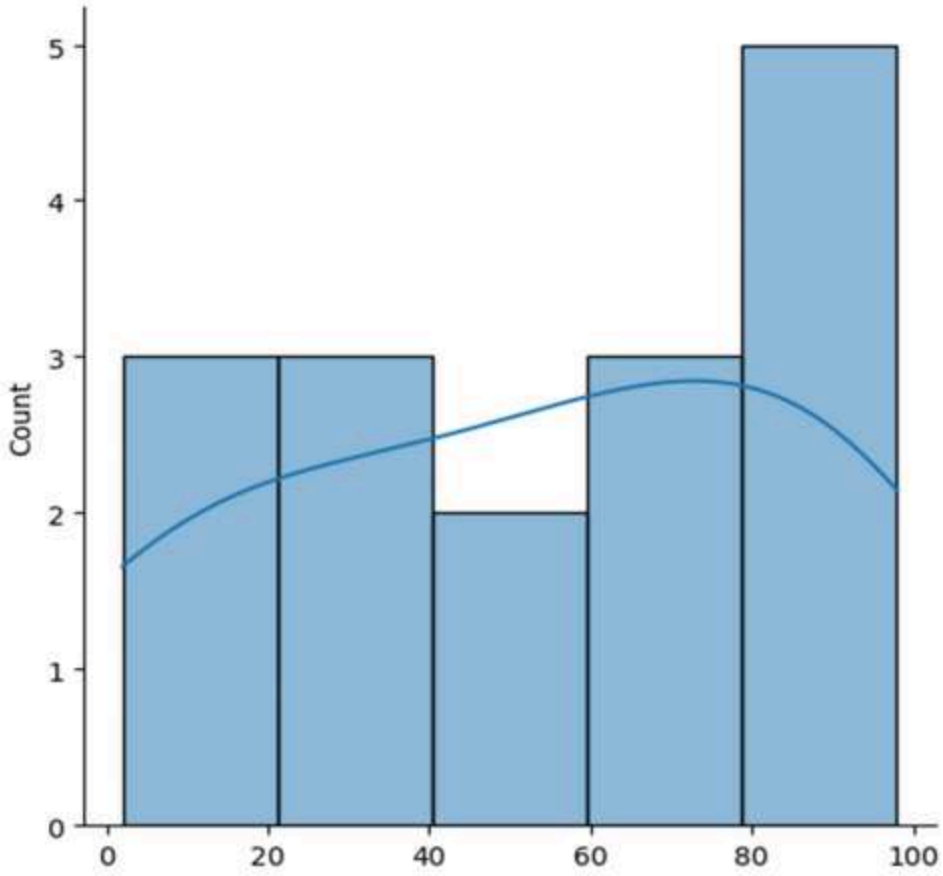
To detect and remove outliers from a dataset using the Interquartile Range (IQR) method and visualize the data distribution before and after outlier removal.

Algorithm:

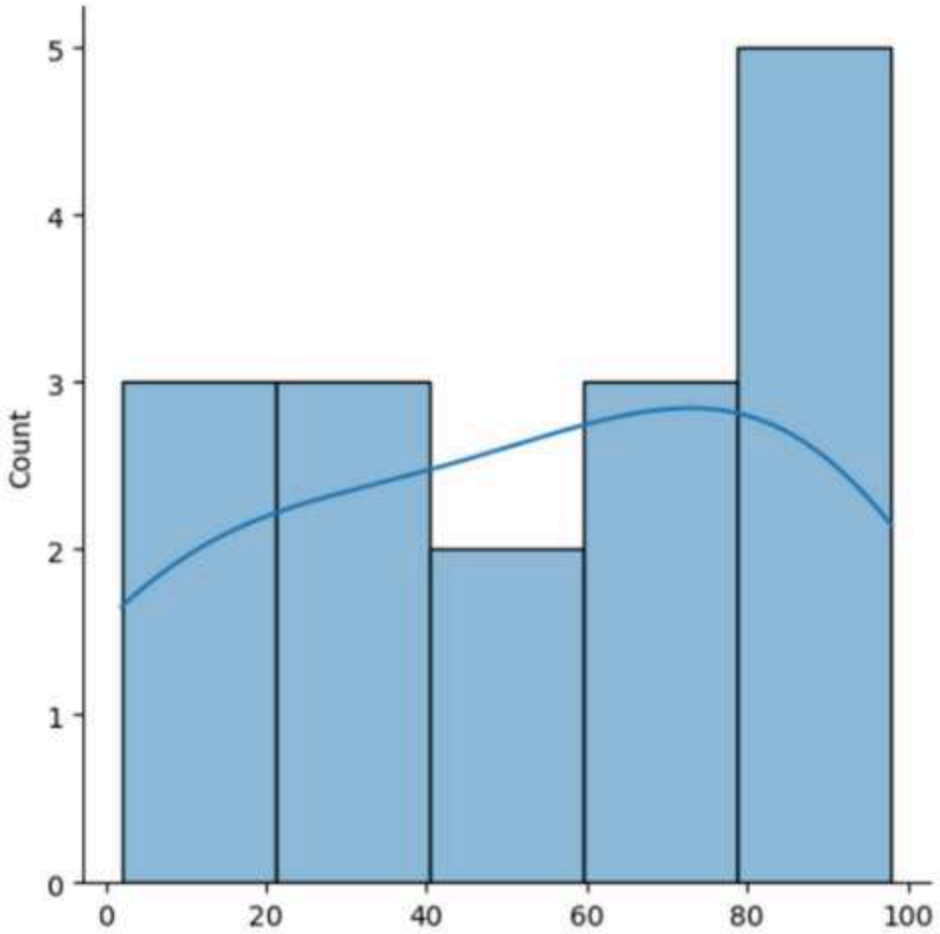
1) **Generate Data:** Create a random array of 16 integers between 1 and 100.
2) **Define Outlier Detection Function:**
    a. Compute Q1 (25th percentile) and Q3 (75th percentile).
    b. Calculate IQR = Q3 − Q1.
    c. Determine lower and upper bounds: `lr = Q1 - 1.5*IQR`, `ur = Q3 + 1.5*IQR`.
3) **Visualize Original Data:** Plot distribution with KDE.
4) **First Outlier Removal:** Filter values within `[lr, ur]` and plot the distribution.
5) **Second Outlier Removal:** Apply IQR again on filtered data and plot final distribution.

Program:

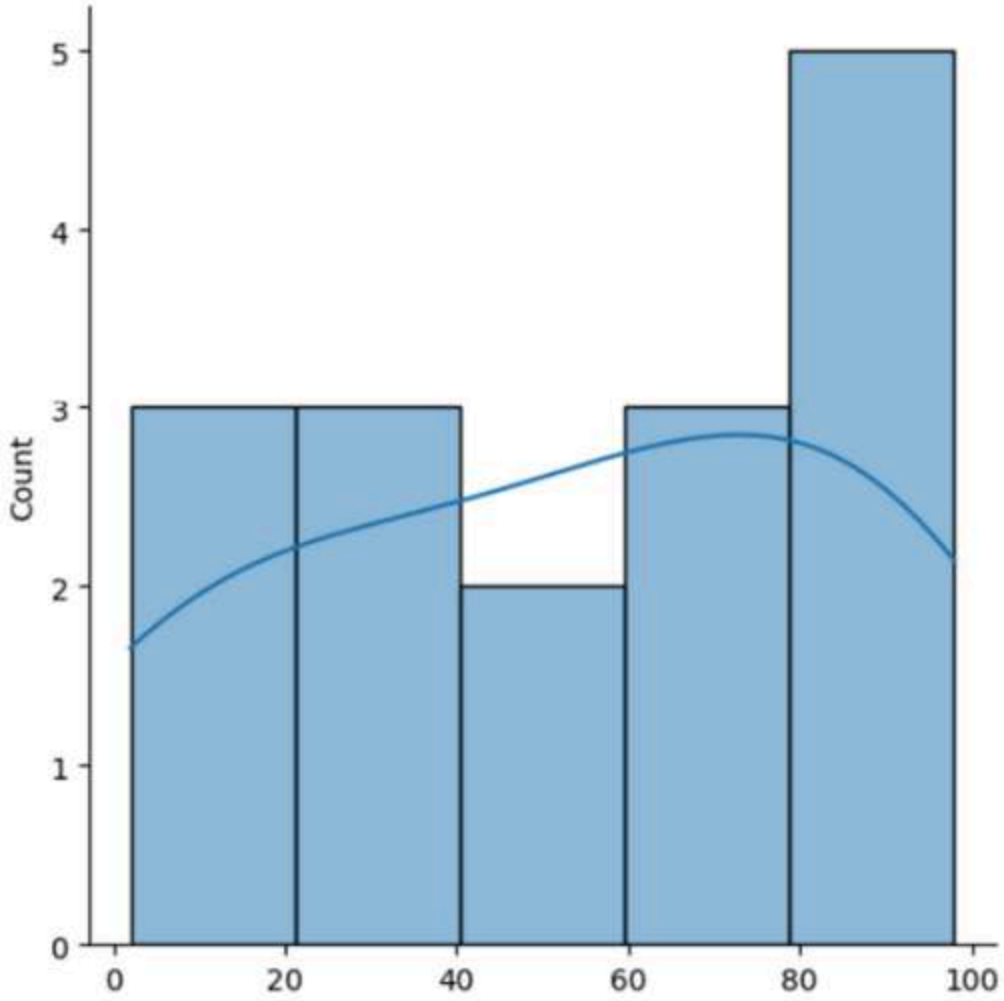
[2]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
arr=np.random.randint(1,100,16)
def outdetection(arr):
    sorted(arr)
    q1,q3=np.percentile(arr,[25,75])
    iqr=q3-q1
    lr=q1-(1.5*iqr)
    ur=q3+(1.5*iqr)
    return lr,ur
lr,ur=outdetection(arr)
sns.displot(arr,kde=True)
plt.show()
```



[3]:
```
new_arr=arr[(arr>lr)&(arr<ur)]
sns.displot(new_arr,kde=True)
plt.show()
```



[4]:
```
lr1,ur1=outdetection(new_arr)
final_arr=new_arr[(new_arr>lr1)&(new_arr<ur1)]
sns.displot(final_arr,kde=True)
plt.show()
```



Result:

The code removes extreme values from a random dataset using IQR, showing three plots: original data, after first outlier removal, and after final outlier removal, resulting in a cleaner, more representative dataset.

EXPERIMENT NO: 5

Experiment to understand feature scaling.

Aim:

To preprocess a dataset by handling missing values, encoding categorical variables, and scaling features using StandardScaler and MinMaxScaler.

Algorithm:

1) **Load Data: Read CSV file into a DataFrame.**
2) **Handle Missing Values: Fill missing Country values with mode; use mean imputation for Age and Salary.**
3) **Separate Features and Label: Extract features and target column.**
4) **Encode Categorical Data: Apply one-hot encoding to the Country column.**
5) **Combine Data: Concatenate encoded Country with numerical features (Age, Salary).**
6) **Scale Features:**
7) **Apply StandardScaler for mean-centered scaling.**
8) **Apply MinMaxScaler to scale features between 0 and 1.**
9) **Output: Print the final combined features and the scaled versions.**

Program:

[1]:
```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler
df = pd.read_csv("C:/Users/vijay/Downloads/pre_process_datasample (1).csv")
df['Country'].fillna(df['Country'].mode()[0], inplace=True)
features = df.iloc[:, :-1].values
label = df.iloc[:, -1].values
age_imputer = SimpleImputer(strategy="mean")
salary_imputer = SimpleImputer(strategy="mean")
features[:, [1]] = age_imputer.fit_transform(features[:, [1]])
features[:, [2]] = salary_imputer.fit_transform(features[:, [2]])
oh = OneHotEncoder(sparse_output=False)
Country = oh.fit_transform(features[:, [0]])
final_set = np.concatenate((Country, features[:, [1, 2]]), axis=1)
sc = StandardScaler()
feat_standard_scaler = sc.fit_transform(final_set)
mms = MinMaxScaler(feature_range=(0, 1))
feat_minmax_scaler = mms.fit_transform(final_set)
print("Final Set:\n", final_set)
```

```
Final Set:
 [[1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [0.0 1.0 0.0 30.0 54000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 35.0 58000.0]
 [0.0 0.0 1.0 38.77777777777778 52000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
```

```
[2]: print("\nStandard Scaled Features:\n", feat_standard_scaler)
```

```
Standard Scaled Features:
 [[ 1.22474487e+00 -6.54653671e-01 -6.54653671e-01  7.58874362e-01
   7.49473254e-01]
 [-8.16496581e-01 -6.54653671e-01  1.52752523e+00 -1.71150388e+00
  -1.43817841e+00]
 [-8.16496581e-01  1.52752523e+00 -6.54653671e-01 -1.27555478e+00
  -8.91265492e-01]
 [-8.16496581e-01 -6.54653671e-01  1.52752523e+00 -1.13023841e-01
  -2.53200424e-01]
 [-8.16496581e-01  1.52752523e+00 -6.54653671e-01  1.77608893e-01
   6.63219199e-16]
 [ 1.22474487e+00 -6.54653671e-01 -6.54653671e-01 -5.48972942e-01
  -5.26656882e-01]
 [-8.16496581e-01 -6.54653671e-01  1.52752523e+00  0.00000000e+00
  -1.07356980e+00]
 [ 1.22474487e+00 -6.54653671e-01 -6.54653671e-01  1.34013983e+00
   1.38753832e+00]
 [-8.16496581e-01  1.52752523e+00 -6.54653671e-01  1.63077256e+00
   1.75214693e+00]
 [ 1.22474487e+00 -6.54653671e-01 -6.54653671e-01 -2.58340208e-01
   2.93712492e-01]]
```

```
[3]: print("\nMin-Max Scaled Features:\n", feat_minmax_scaler)
```

```
Min-Max Scaled Features:
 [[1.         0.         0.         0.73913043 0.68571429]
 [0.         0.         1.         0.         0.        ]
 [0.         1.         0.         0.13043478 0.17142857]
 [0.         0.         1.         0.47826087 0.37142857]
 [0.         1.         0.         0.56521739 0.45079365]
 [1.         0.         0.         0.34782609 0.28571429]
 [0.         0.         1.         0.51207729 0.11428571]
 [1.         0.         0.         0.91304348 0.88571429]
 [0.         1.         0.         1.         1.        ]
 [1.         0.         0.         0.43478261 0.54285714]]
```

Result:

The dataset is cleaned and numeric features are combined with one-hot encoded Country. Standard scaling centers the data, while Min-Max scaling normalizes it between 0 and 1, making it ready for machine learning.

EXPERIMENT NO: 5

Data Visualization using Seaborn

Aim:

To explore and visualize the tips dataset using various plots to understand distributions, relationships, correlations, and categorical counts.

Algorithm:

1) **Load Data: Import the tips dataset from Seaborn.**
2) **Visualize Distributions:**
3) **Use displot for total_bill with and without KDE.**
4) **Visualize Relationships:**
5) **jointplot for tip vs total_bill (scatter, regression, and hex).**
6) **pairplot for all numerical features; add hue for time and day.**
7) **Correlation Analysis:**
8) **Use heatmap to display correlations between numeric features.**
9) **Outlier Detection:**
10) **Use boxplot for total_bill and tip.**
11) **Categorical Counts:**
12) **Use countplot for day and sex.**

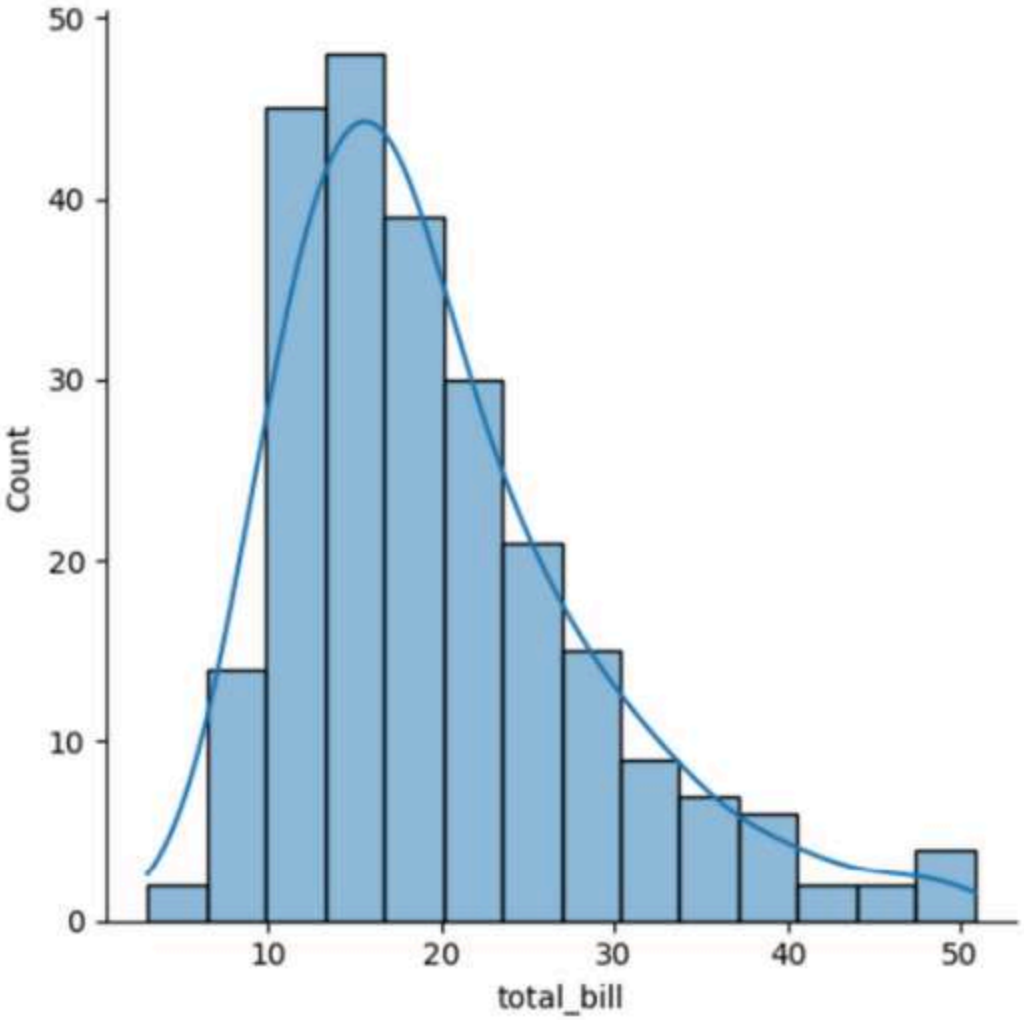
Program:

[1]:
```
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()
```

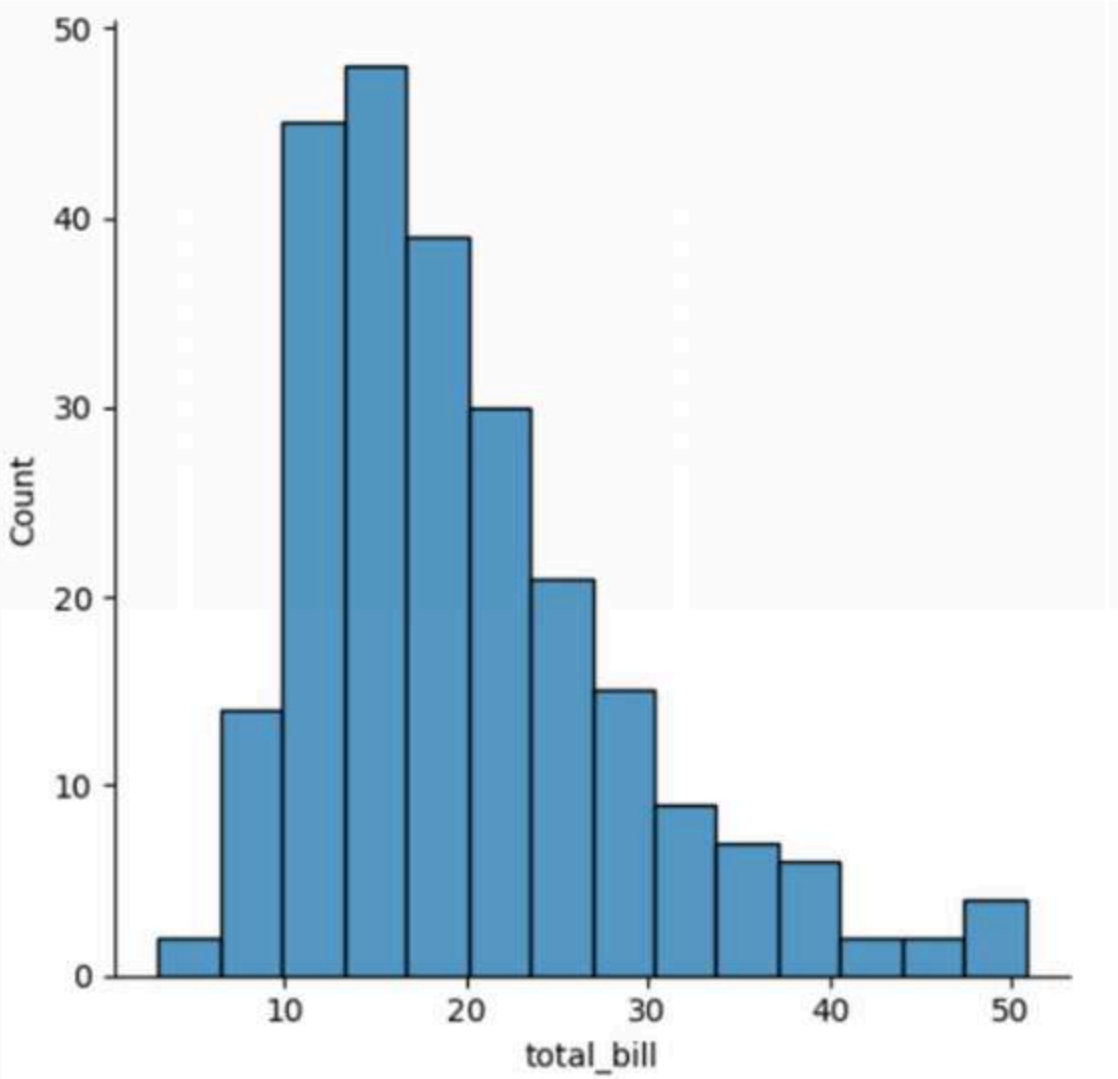
[1]:

	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

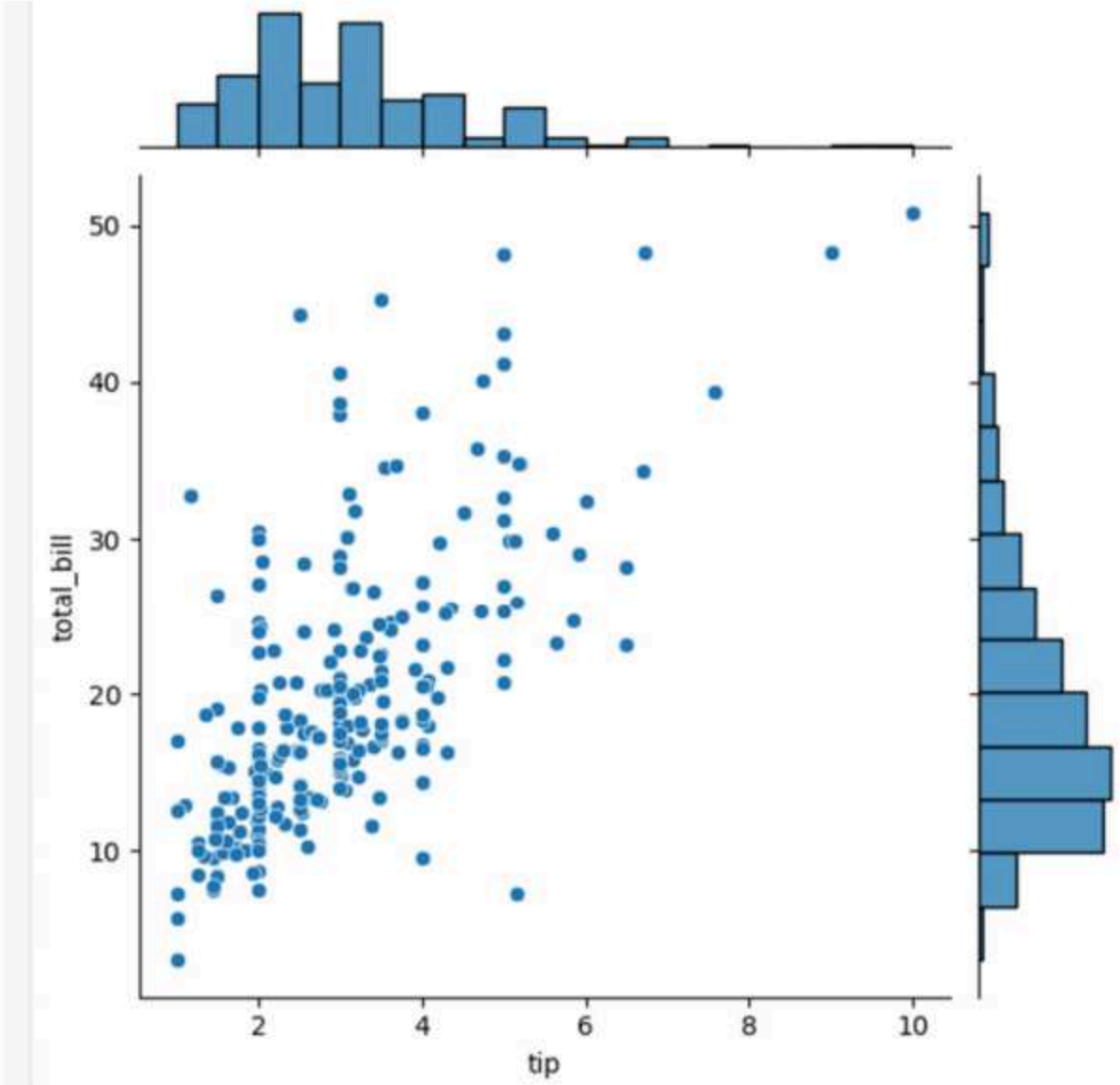
[2]:
```
m=sns.displot(tips.total_bill,kde=True)
plt.show()
```



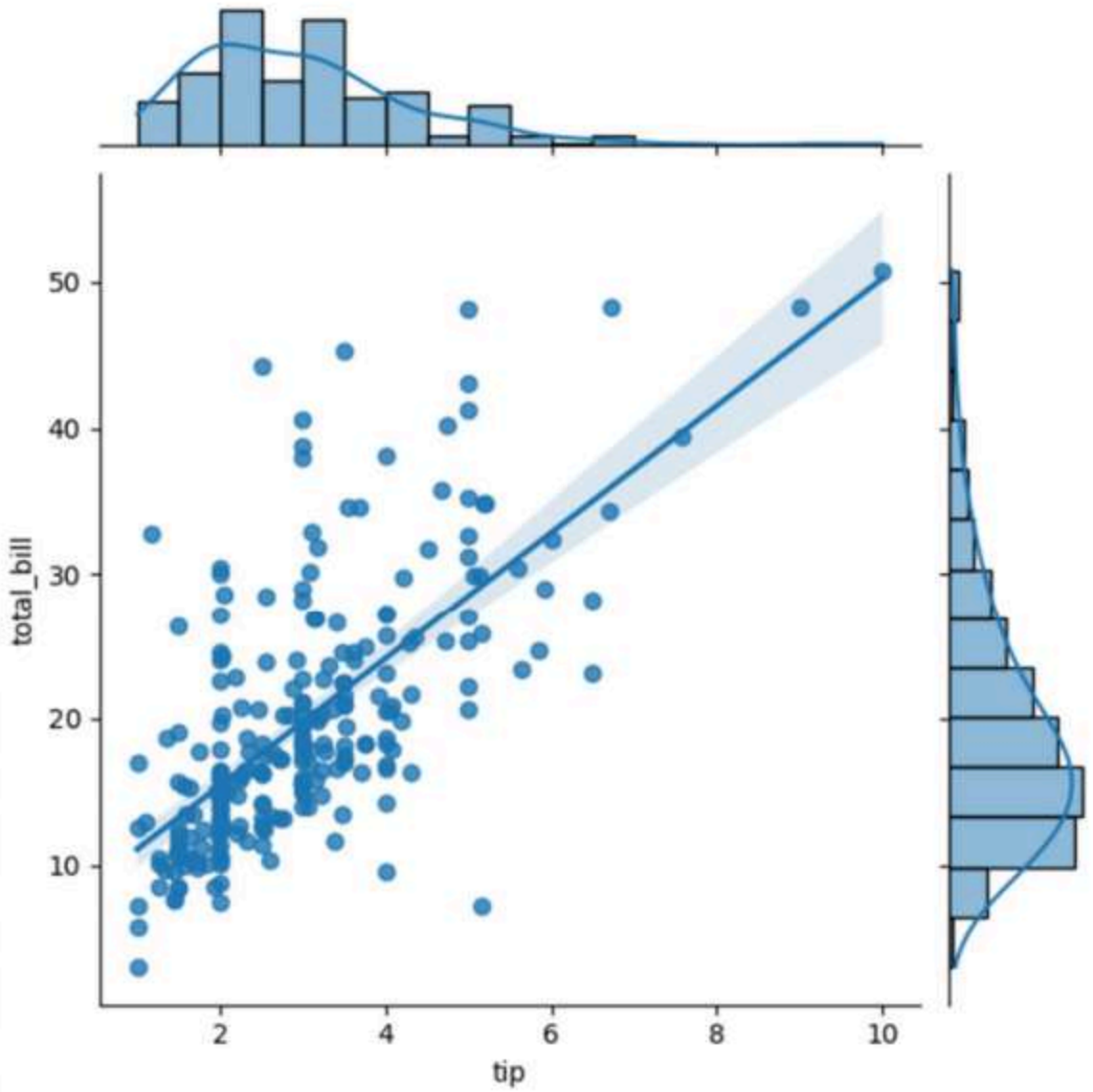
[3]:
```
n=sns.displot(tips.total_bill,kde=False)
plt.show()
```



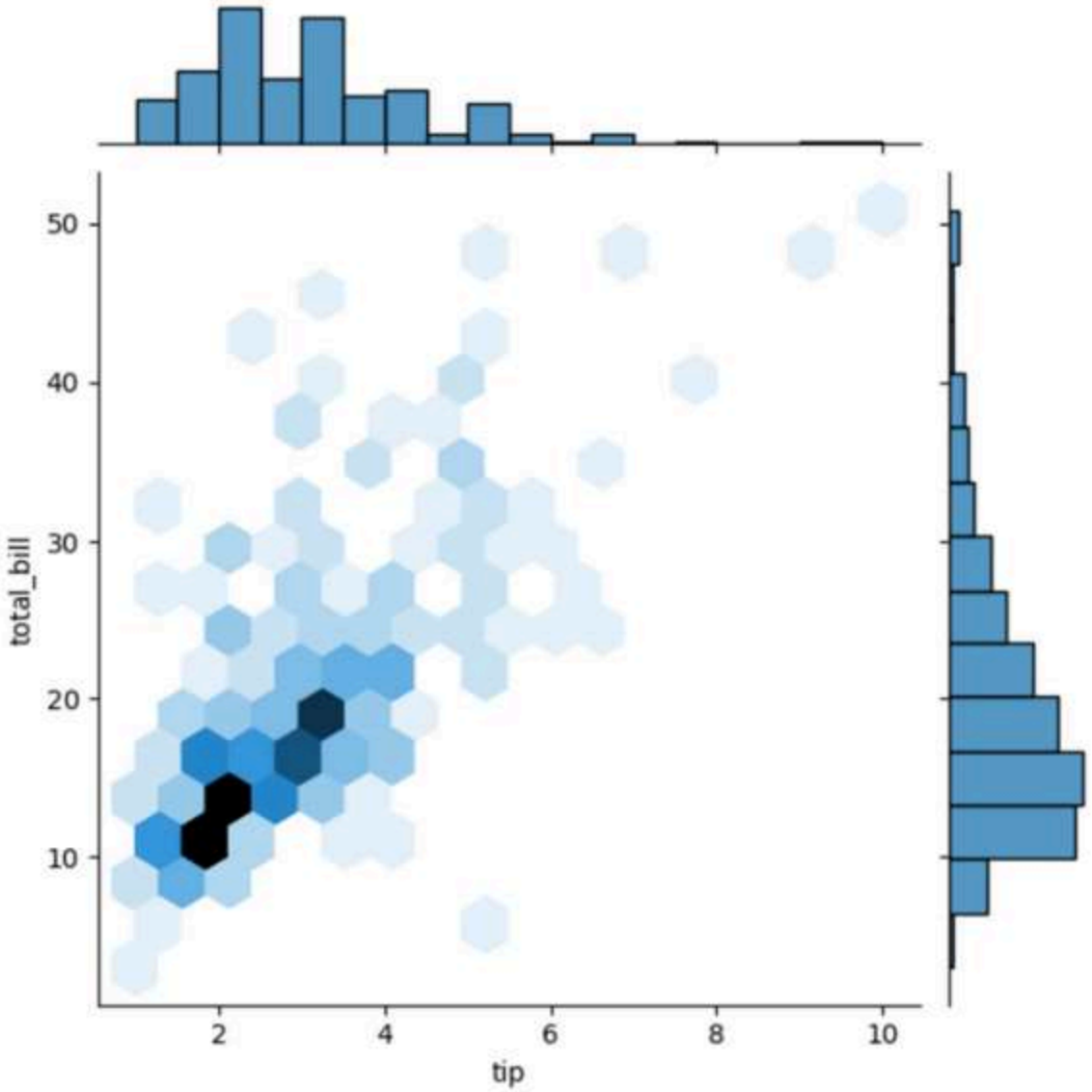

```
]: p=sns.jointplot(x=tips.tip,y=tips.total_bill)
   plt.show()
```



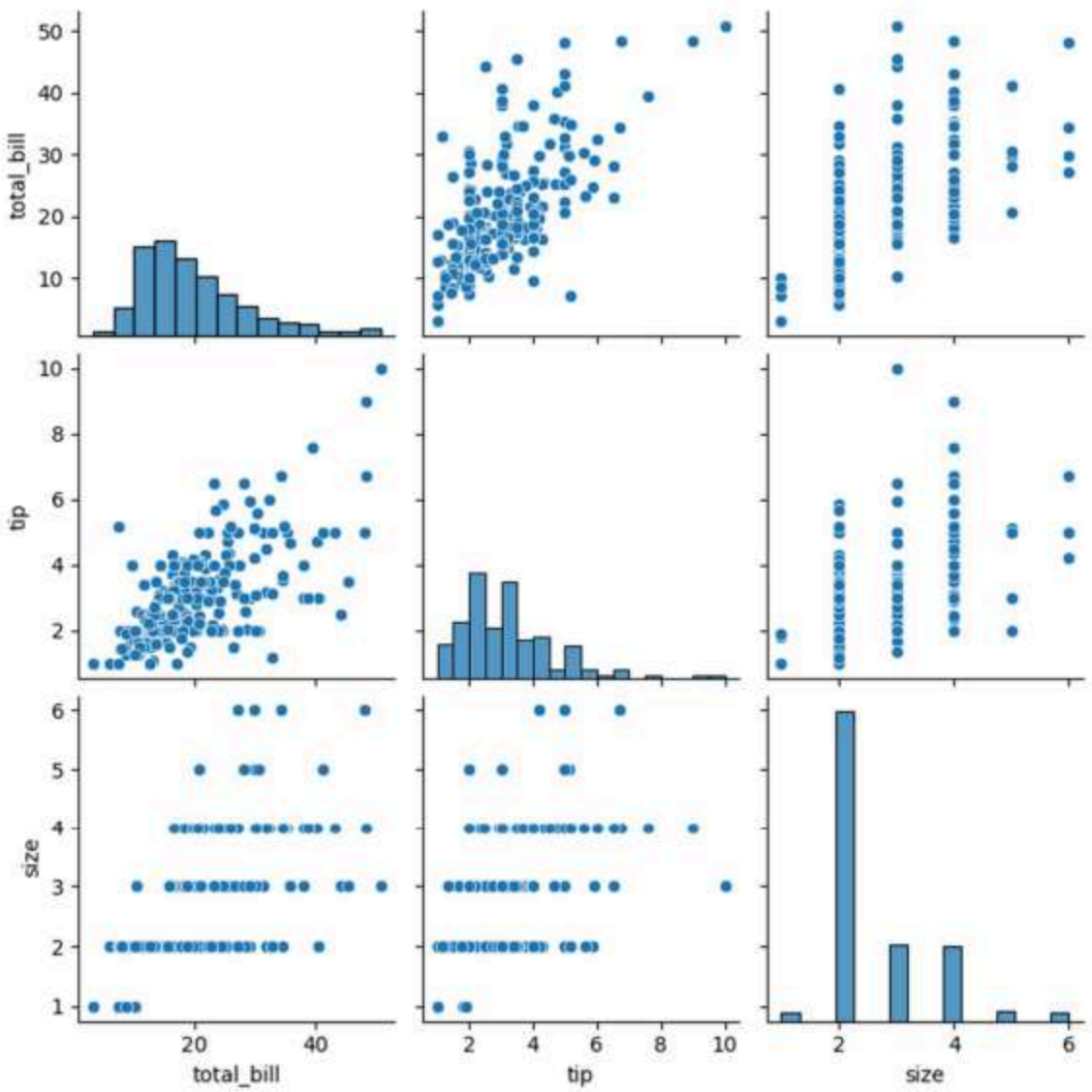

```
l=sns.jointplot(x=tips.tip,y=tips.total_bill,kind="reg")
plt.show()
```



]:
```
q=sns.jointplot(x=tips.tip,y=tips.total_bill,kind="hex")
plt.show()
```



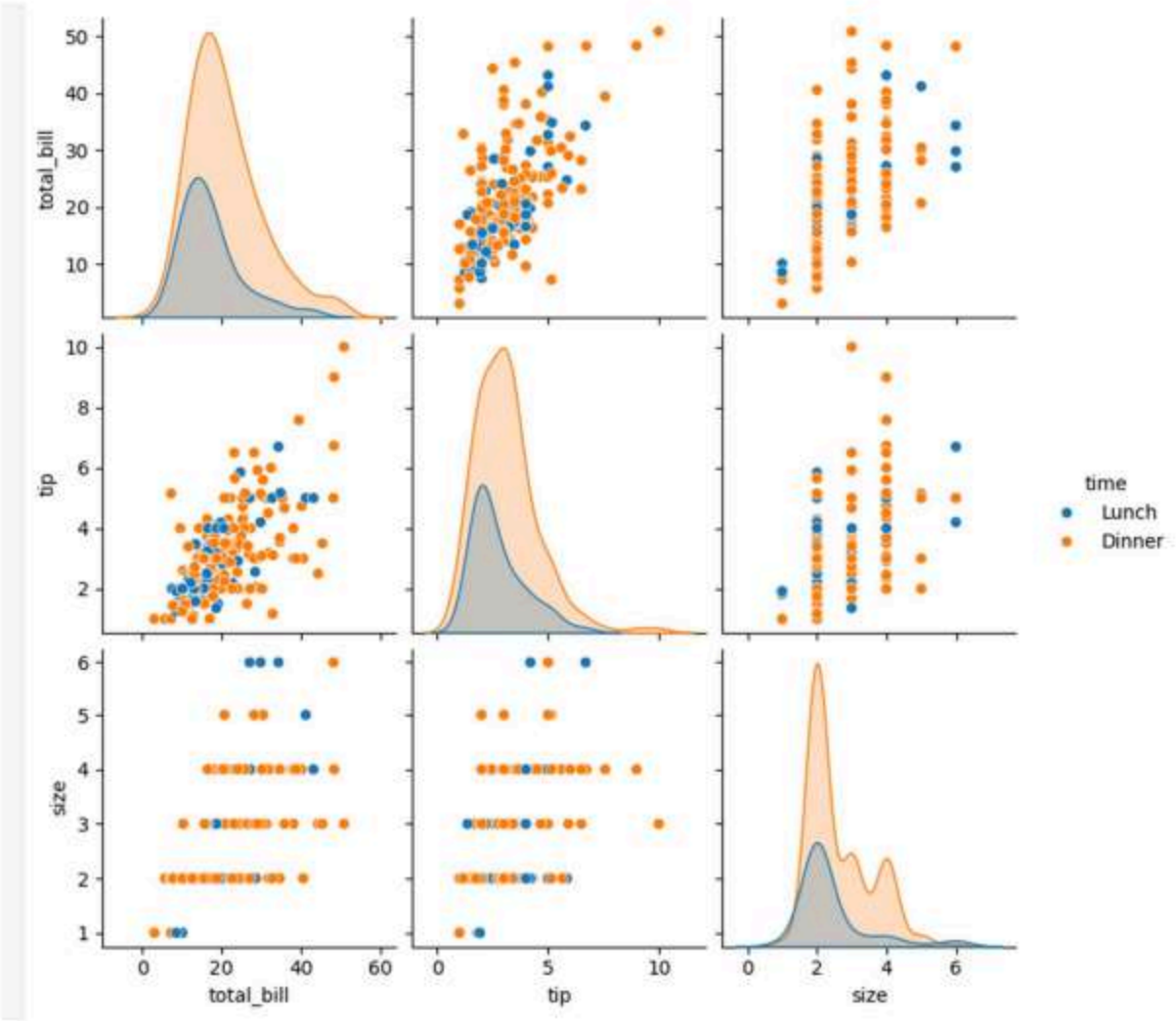

```
sns.pairplot(tips)
plt.show()
```



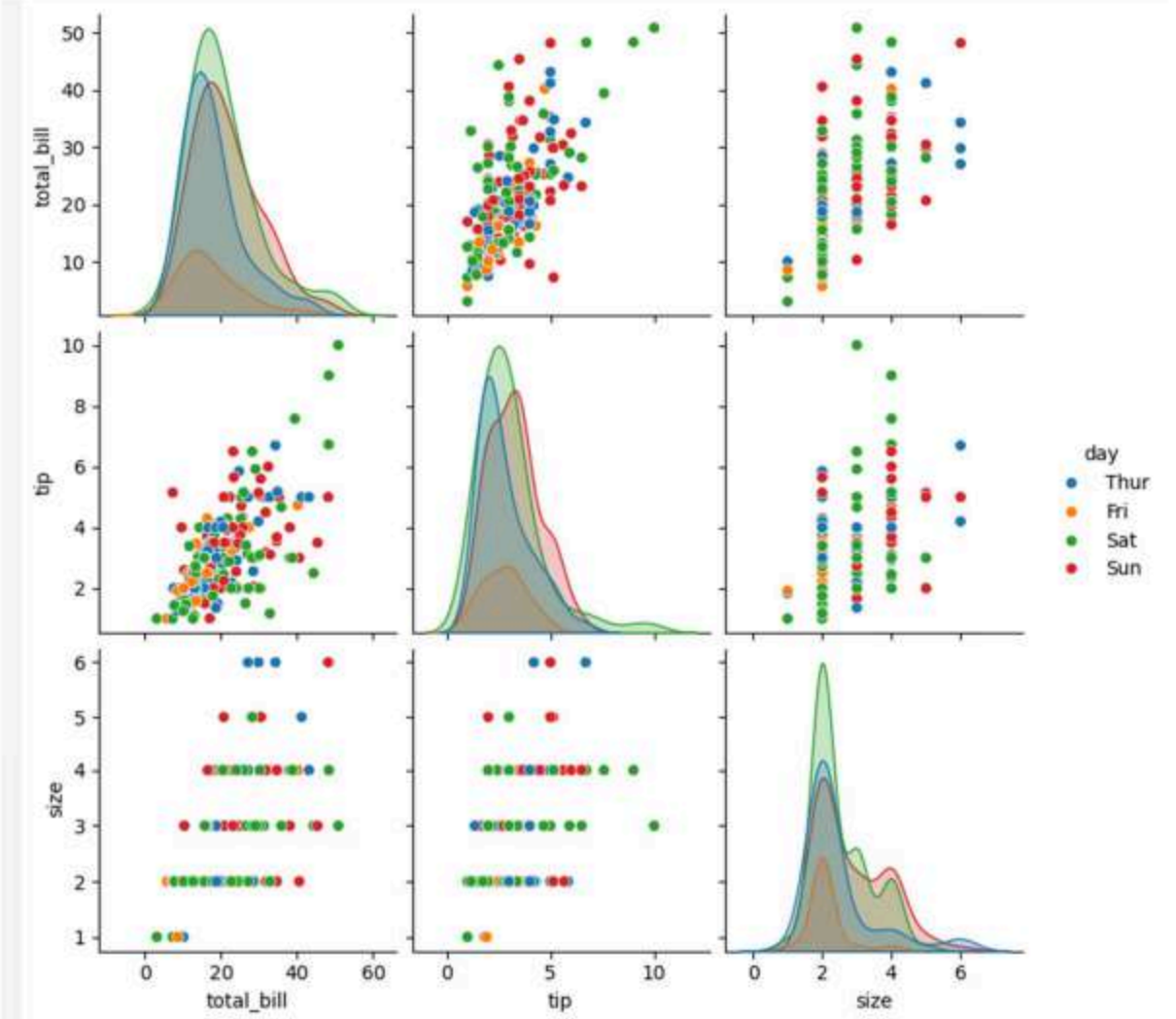
]:
```
tips.time.value_counts()
```

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

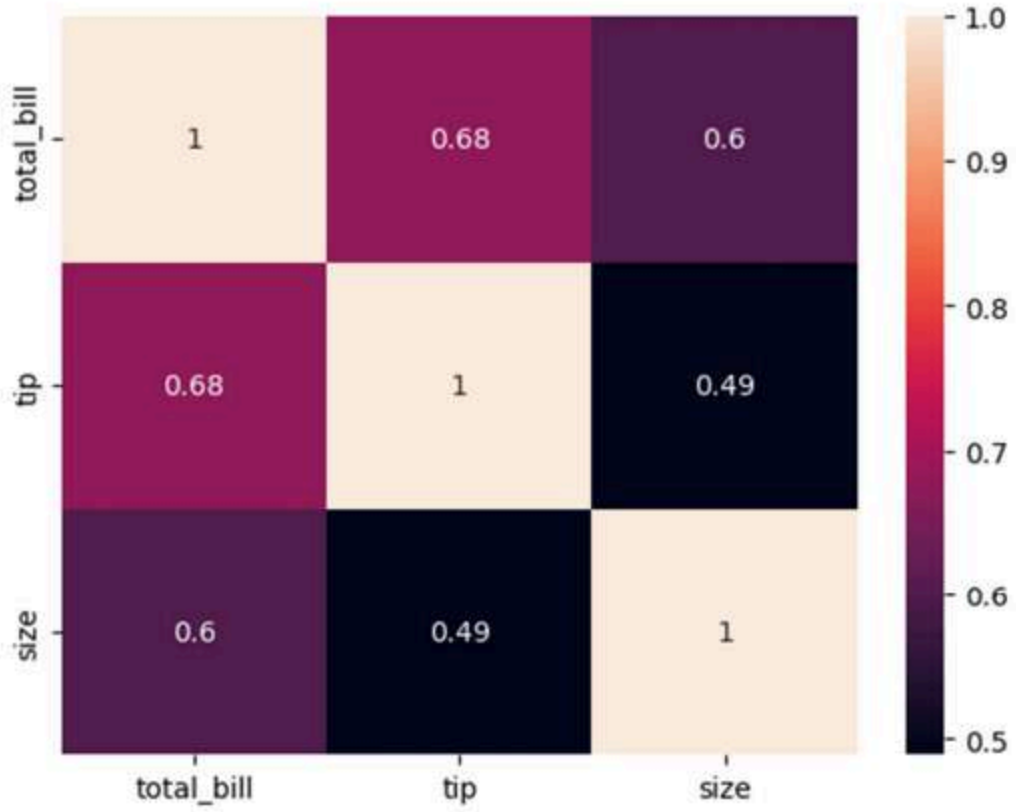
]:
```
sns.pairplot(tips,hue='time')
plt.show()
```


total_bill
tip
size
time
Lunch
Dinner

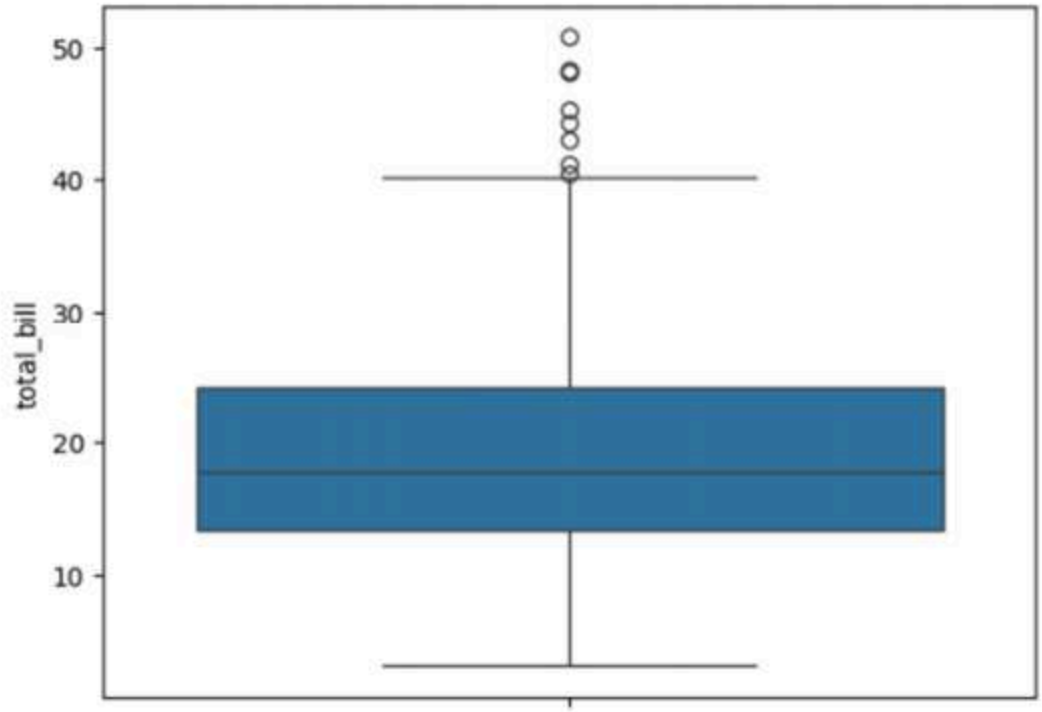
```
sns.pairplot(tips,hue='day')
plt.show()
```



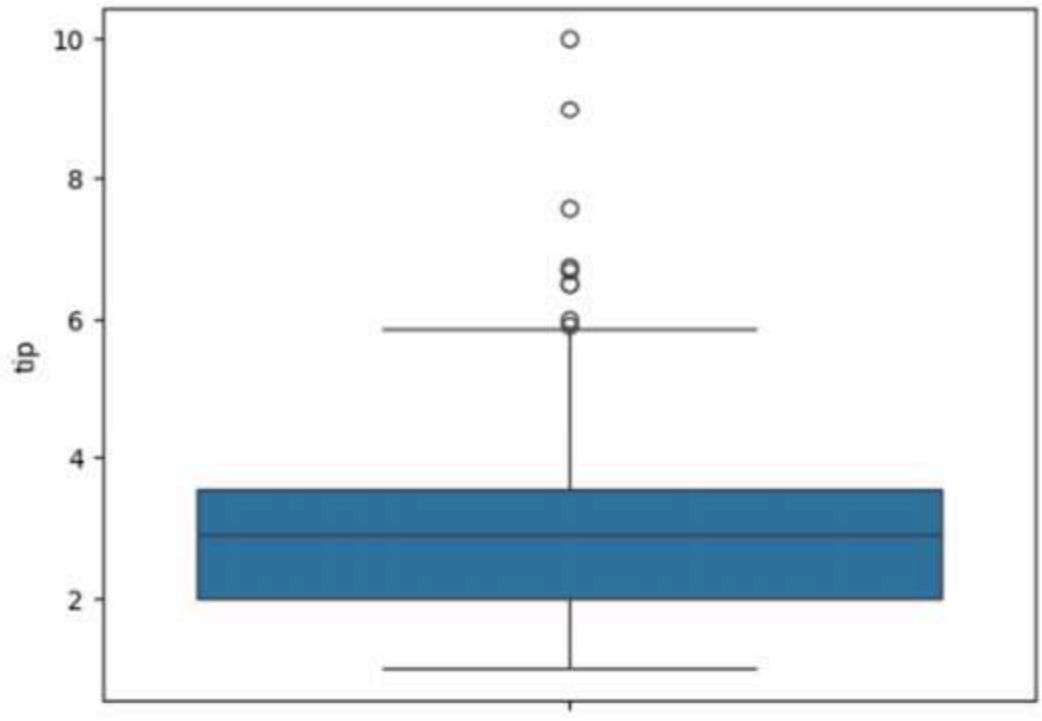
```
sns.heatmap(tips.corr(numeric_only=True),annot=True)
plt.show()
```



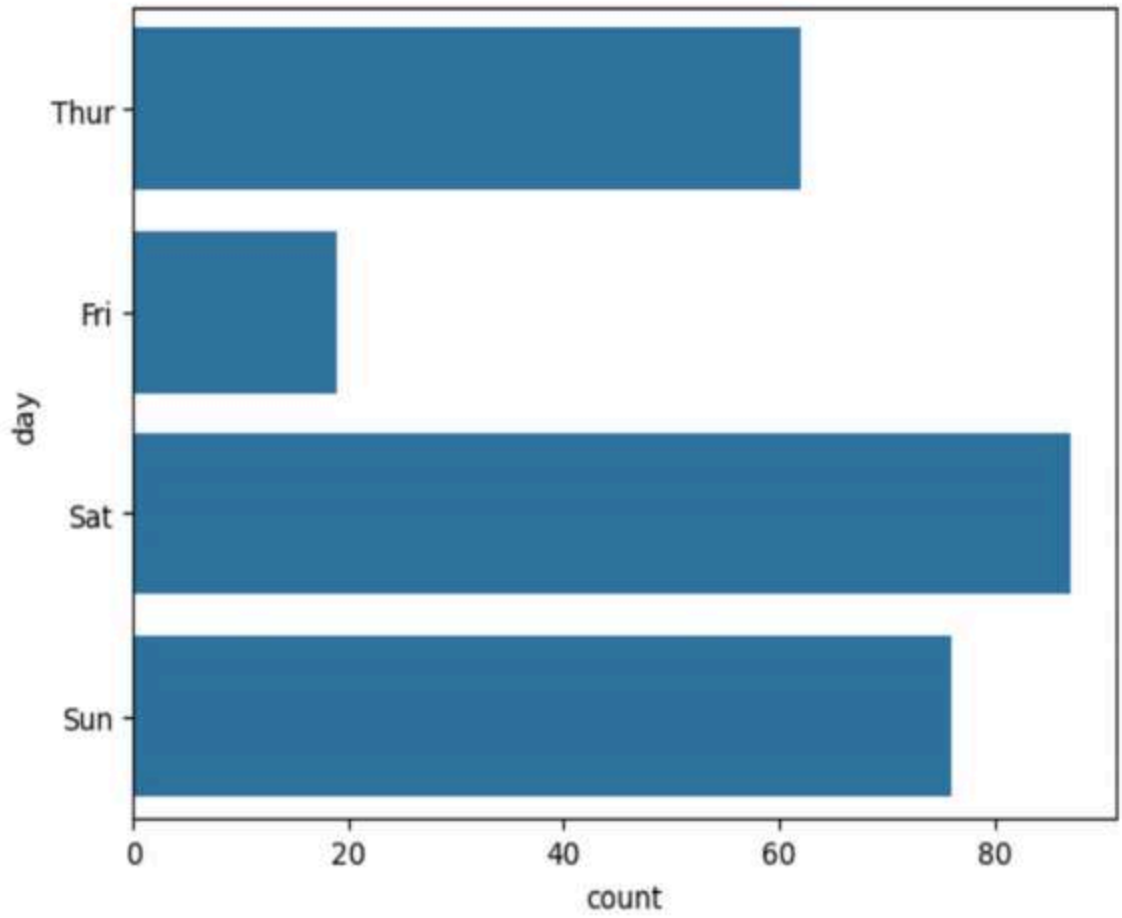

```
sns.boxplot(tips.total_bill)
plt.show()
```



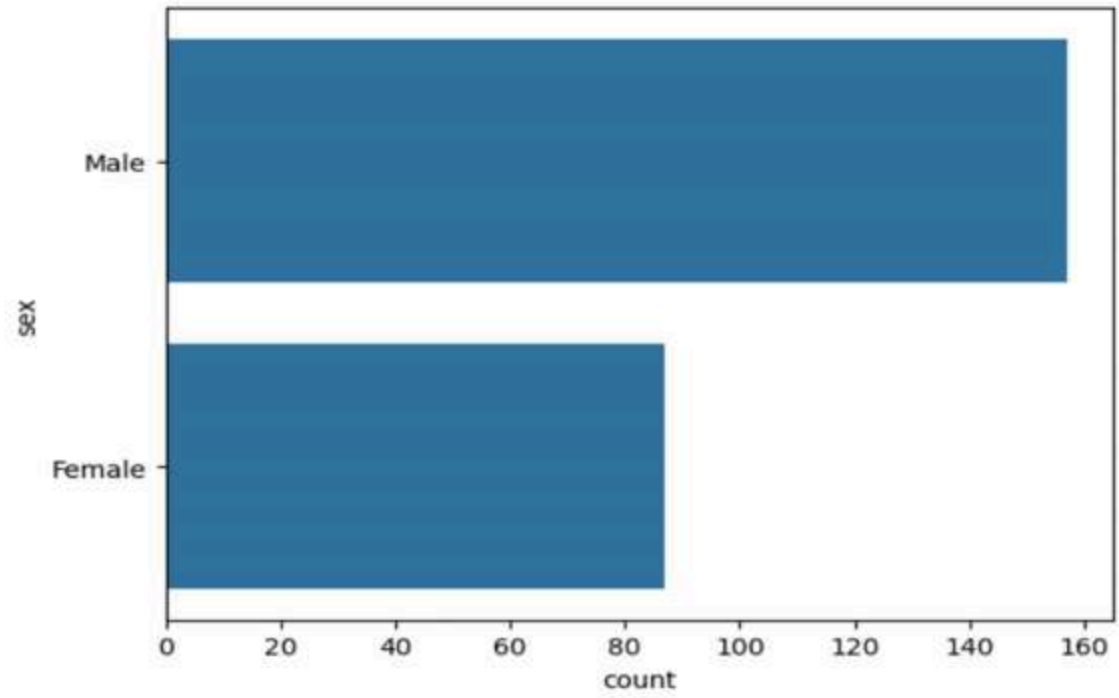

```
]: sns.boxplot(tips.tip)
   plt.show()
```



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




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



Result:

The visualizations show distributions of bills and tips, relationships between tip and total bill, correlations among numeric features, outliers, and counts by day and gender, providing a clear overview of the dataset.

EXPERIMENT NO: 7

Salary Prediction using Linear Regression

Aim:

To build and train a linear regression model that predicts an employee's salary based on their years of experience.

Algorithm:

1. **Load Data: Import the salary dataset using pandas.**
2. **Clean Data: Check for missing values and remove null entries.**
3. **Split Data: Separate features (YearsExperience) and labels (Salary), then divide into training and testing sets.**
4. **Train Model: Fit a LinearRegression model using the training data.**
5. **Evaluate Model: Calculate model accuracy using training and testing scores.**
6. **Model Parameters: Display slope (coef_) and intercept (intercept_).**
7. **Save & Load Model: Save the trained model using pickle and reload it for use.**
8. **Prediction: Take user input for years of experience and predict the corresponding salary.**

Program:

[22]:
```
import numpy as np
import pandas as pd
df=pd.read_csv("C:/Users/vijay/Downloads/Salary_data.csv")
df
```

[22]:

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891
5	2.9	56642
6	3.0	60150
7	3.2	54445
8	3.2	64445

[23]:
```
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
```

[24]:
```
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: 612.0 bytes
```

[25]:
```
df.describe()
```

[25]:

	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

[26]:
```
features=df.iloc[:,[0]].values
label=df.iloc[:,[1]].values
```

[27]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(features,label,test_size=0.2,random_state=42)
```

[28]:
```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
```

```
[29]: model.fit(x_train,y_train)
      model.score(x_train,y_train)
      model.score(x_test,y_test)

[29]: 0.9024461774180497

[30]: model.coef_

[30]: array([[9423.81532303]])

[31]: model.intercept_

[31]: array([25321.58301178])

[32]: import pickle
      pickle.dump(model,open('SalaryPred.model','wb'))
      model=pickle.load(open('SalaryPred.model','rb'))

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

     Enter Years of Experience:  44
```

Result:

The trained linear regression model accurately predicts salary based on years of experience. The model is saved for future use, and upon entering a number of years, it outputs the estimated salary value.

EXPERIMENT NO: 8

Iris Flower Classification using K-Nearest Neighbors (KNN)

Aim:

To classify Iris flower species based on their sepal and petal measurements using the K-Nearest Neighbors algorithm.

Algorithm:

1. **Load Data: Read the Iris dataset using pandas.**
2. **Explore Data: Check data info, count species, and display sample records.**
3. **Split Data: Separate features (measurements) and labels (species).**
4. **Train-Test Split: Divide data into 80% training and 20% testing sets.**
5. **Model Training: Initialize KNN with 5 neighbors and fit it on training data.**
6. **Evaluate Model: Display training and testing accuracy scores.**
7. **Performance Metrics: Generate and print confusion matrix and classification report.**

Program:

[1]:
```
import numpy as np
import pandas as pd
df=pd.read_csv("C:/Users/vijay/Downloads/Iris (1).csv")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal.length  150 non-null    float64
 1   sepal.width   150 non-null    float64
 2   petal.length  150 non-null    float64
 3   petal.width   150 non-null    float64
 4   variety       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

[2]:
```
df.variety.value_counts()
```

[2]:
```
variety
Setosa        50
Versicolor    50
Virginica     50
Name: count, dtype: int64
```

[3]:
```
df.head()
```

[3]:

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

[4]:
```
features=df.iloc[:,:-1].values
label=df.iloc[:,4].values
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

[5]:
```
xtrain,xtest,ytrain,ytest=train_test_split(features,label,test_size=0.2,random_state=42)
model_KNN=KNeighborsClassifier(n_neighbors=5)
model_KNN.fit(xtrain,ytrain)
```

[5]:
```
KNeighborsClassifier()
```

[6]:
```
print(model_KNN.score(xtrain,ytrain))
print(model_KNN.score(xtest,ytest))
```

```
0.9666666666666667
1.0
```

[7]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(label,model_KNN.predict(features))
```

[7]:
```
array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  1, 49]])
```

[8]:
```
from sklearn.metrics import classification_report
print(classification_report(label,model_KNN.predict(features)))
```

```
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        50
  Versicolor       0.98      0.94      0.96        50
   Virginica       0.94      0.98      0.96        50

    accuracy                           0.97       150
   macro avg       0.97      0.97      0.97       150
weighted avg       0.97      0.97      0.97       150
```

Result:

The KNN model successfully classifies Iris flowers into their respective species with high accuracy. The confusion matrix and classification report confirm that the model performs well with minimal misclassifications.

EXPERIMENT NO: 9

Social Network Ads Classification using Logistic Regression

Aim:

To build a Logistic Regression model that predicts whether a user purchases a product based on their age and estimated salary.

Algorithm:

1. **Load Data: Read the Social Network Ads dataset using pandas.**
2. **Select Features and Labels:**
3. **Features → Age and Estimated Salary**
4. **Label → Purchased (0 or 1)**
5. **Model Optimization:**
6. **Run multiple train-test splits with different random states (1–400).**
7. **Identify the state where the test accuracy exceeds the training accuracy.**
8. **Train Final Model:**
9. **Split data with the best random state (306).**
10. **Train the Logistic Regression model.**
11. **Evaluate Model:**
12. **Display training and testing accuracy.**
13. **Generate classification report to assess precision, recall, and F1-score.**

Program:

[1]:
```
import pandas as pd
import numpy as np
df=pd.read_csv("C:/Users/vijay/Downloads/Social_Network_Ads.csv")
df
```

[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

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

[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

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

[3]:
```
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]
```

[4]:
```
label
```

[4]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 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, 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, 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])
```

[5]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

[8]:
```
for i in range(1,401):
    xtrain,xtest,ytrain,ytest=train_test_split(features,label,test_size=0.2,random_state=i)
    model=LogisticRegression()
    model.fit(xtrain,ytrain)
    train_score=model.score(xtrain,ytrain)
    test_score=model.score(xtest,ytest)
    if test_score>train_score:
        print("Test: {:.3f}, Train: {:.3f}, Random State: {}".format(test_score,train_score,i))
```

```
Test: 0.900, Train: 0.841, Random State: 10
Test: 0.863, Train: 0.856, Random State: 14
Test: 0.850, Train: 0.844, Random State: 15
Test: 0.863, Train: 0.856, Random State: 16
Test: 0.875, Train: 0.834, Random State: 18
Test: 0.850, Train: 0.844, Random State: 19
Test: 0.875, Train: 0.844, Random State: 20
Test: 0.863, Train: 0.834, Random State: 21
Test: 0.875, Train: 0.841, Random State: 22
Test: 0.875, Train: 0.841, Random State: 24
Test: 0.850, Train: 0.834, Random State: 26
Test: 0.850, Train: 0.841, Random State: 27
Test: 0.863, Train: 0.834, Random State: 30
```

[9]:
```
xtrain,xtest,ytrain,ytest=train_test_split(features,label,test_size=0.2,random_state=306)
fmodel=LogisticRegression()
fmodel.fit(xtrain,ytrain)
```

[9]:
```
▾ LogisticRegression
LogisticRegression()
```

[10]:
```
print(fmodel.score(xtrain,ytrain))
print(fmodel.score(xtest,ytest))
```

```
0.8375
0.9125
```

[12]:
```
from sklearn.metrics import classification_report
print(classification_report(label,fmodel.predict(features)))
```

```
              precision    recall  f1-score   support

           0       0.86      0.91      0.89       257
           1       0.83      0.74      0.78       143

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

Result:

The Logistic Regression model effectively predicts whether a customer will purchase based on age and salary. The model shows balanced training and testing accuracy, and the classification report confirms good overall performance.

EXPERIMENT NO: 10

Customer Segmentation using K-Means Clustering

Aim:

To segment mall customers into distinct groups based on their annual income and spending score using the K-Means clustering algorithm.

Algorithm:

1. **Load Data: Import the Mall_Customers.csv dataset using pandas.**
2. **Visualize Data: Use pairplot() to observe relationships among features.**
3. **Select Features: Extract Annual Income (k$) and Spending Score (1-100) for clustering.**
4. **Apply K-Means:**
5. **Initialize K-Means with 5 clusters.**
6. **Fit the model and assign cluster labels to each customer.**
7. **Visualize Clusters:**
8. **Plot scatter points showing different clusters based on income and spending.**
9. **Find Optimal Clusters:**
10. **Calculate WCSS (Within-Cluster Sum of Squares) for cluster counts 1–9.**
11. **Plot the Elbow Curve to determine the best number of clusters.**

Program:

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[2]:
```
df=pd.read_csv("C:/Users/vijay/Downloads/Mall_Customers.csv")
df.info()
```

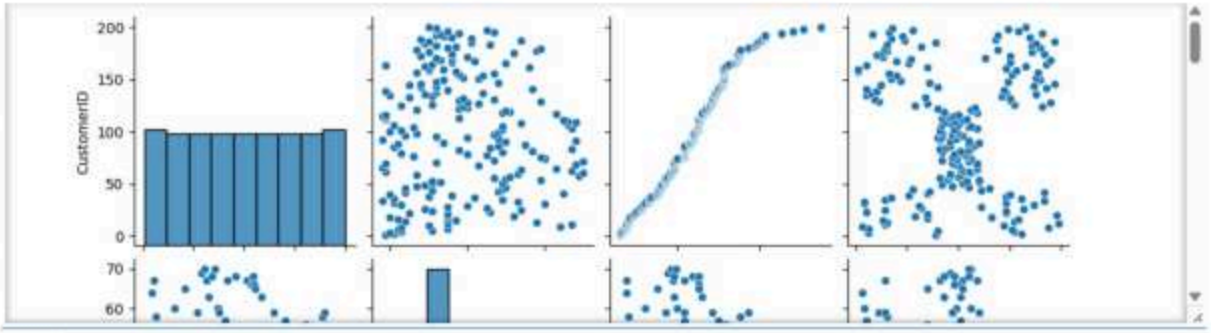
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

[3]:
```
df.head()
```

[3]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

[8]:
```
p=sns.pairplot(df)
plt.show()
```



[9]:
```
features=df.iloc[:,[3,4]].values
from sklearn.cluster import KMeans
model=KMeans(n_clusters=5)
model.fit(features)
KMeans(n_clusters=5)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

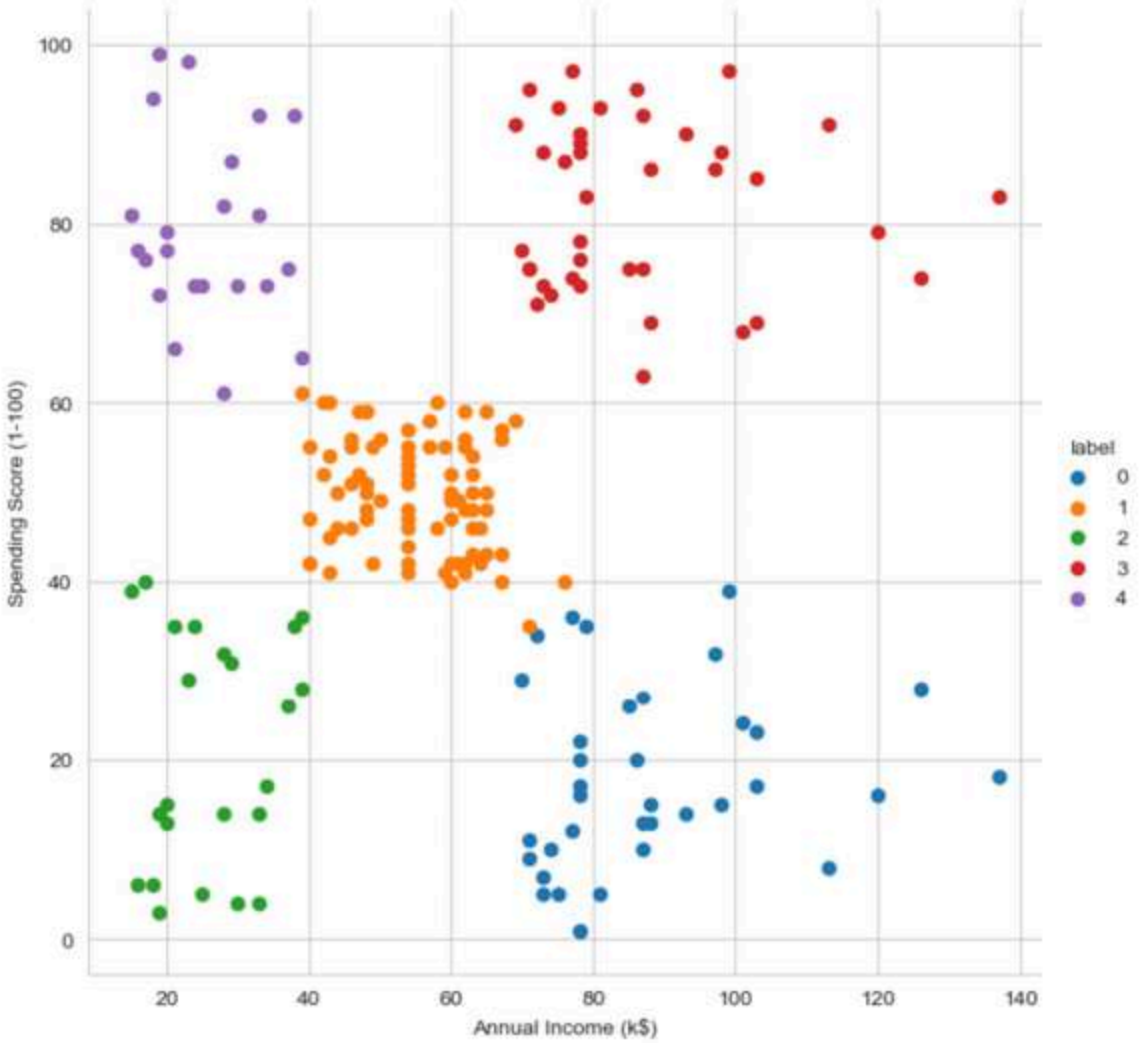
[9]:
```
KMeans
KMeans(n_clusters=5)
```

[10]:
```
Final=df.iloc[:,[3,4]]
Final['label']=model.predict(features)
Final.head()
```

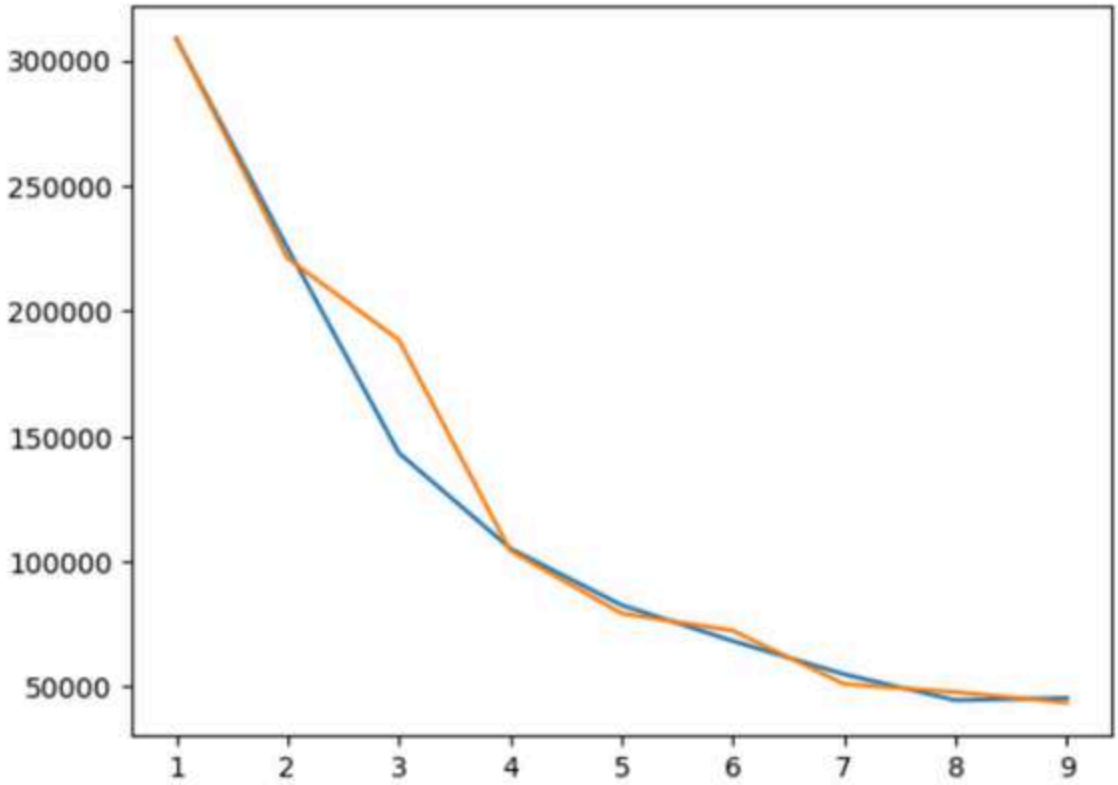
10]:

	Annual Income (k$)	Spending Score (1-100)	label
0	15	39	2
1	15	81	4
2	16	6	2
3	16	77	4
4	17	40	2

16]:
```
sns.set_style("whitegrid")
sns.FacetGrid(Final,hue="label",height=7) \
.map(plt.scatter,"Annual Income (k$)", "Spending Score (1-100)") \
.add_legend();
plt.show()
```




```
features_el=df.iloc[:,[2,3,4]].values
from sklearn.cluster import KMeans
wcss=[]
for i in range(1,10):
    model=KMeans(n_clusters=i)
    model.fit(features_el)
    wcss.append(model.inertia_)
plt.plot(range(1,10),wcss)
plt.show()
```



Result:

The K-Means algorithm groups customers into 5 clusters based on their spending habits and income. The Elbow Method helps confirm that 5 clusters provide the most balanced segmentation, clearly showing different customer behavior groups.

EXPERIMENT NO: 11

Random Sampling and Sampling Distribution

Aim:

To explore random sampling from a population and understand the concept of sampling distribution using Python in Jupyter Notebook.

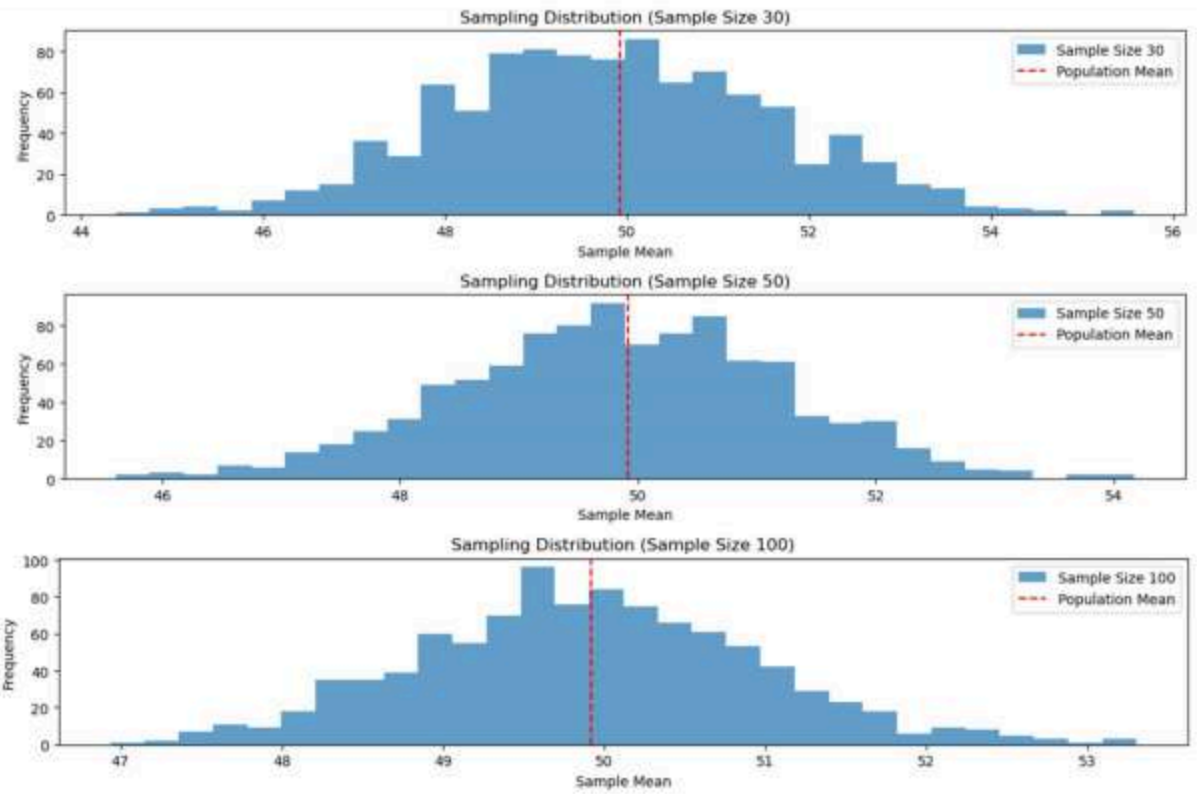
Algorithm:

- **Initialize population parameters**
- **Generate the population**
- **Select sample sizes**
- **Repeat sampling**
- **Compute sample means**
- **Form sampling distributions**
- **Plot histograms**
- **Mark population mean**
- **Analyze results**

Code:

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


population_mean = 50
population_std = 10
population_size = 100000
population = np.random.normal(population_mean, population_std, population_size)


sample_sizes = [30, 50, 100]
num_samples = 1000
sample_means = {}

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

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


plt.figure(figsize=(12, 8))

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

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



Result:

All sampling distributions are centered around 50, and as the sample size increases, the spread decreases, giving more accurate estimates of the population mean.

EXPERIMENT NO: 12

Hypothetical using Z-Test

Aim:

To test whether the sample mean IQ differs significantly from the population mean (100) using a one-sample t-test.

Algorithm:

- Generate a random sample from a normal distribution (mean = 102, std = 15).
- Calculate the **sample mean** and **sample standard deviation**.
- Perform a **one-sample t-test** comparing the sample mean to the population mean (100).
- Compare the **p-value** with the significance level ($\alpha = 0.05$).
- If $p < \alpha$, reject the null hypothesis; otherwise, fail to reject it.

Code:

```
import numpy as np
import scipy.stats as stats
np.random.seed(42)
sample_size = 25
sample_data = np.random.normal(loc=102, scale=15, size=sample_size)
population_mean = 100
sample_mean = np.mean(sample_data)
sample_std = np.std(sample_data, ddof=1)
n = len(sample_data)
t_statistic, p_value = stats.ttest_1samp(sample_data,
population_mean)
print(f"Sample Mean: {sample_mean:.2f}")
```

```
Sample Mean: 99.55
```

```
print(f"T-Statistic: {t_statistic:.4f}")
```

```
T-Statistic: -0.1577
```

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

```
P-Value: 0.8760
```

```
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.")
```

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

Result:

The test shows that the sample's average IQ score is not significantly different from the population mean of 100. Hence, we fail to reject the null hypothesis.

EXPERIMENT NO : 13

Hypothetical using T-Test

Aim:

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

Algorithm:

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

Code:

```
import numpy as np
import scipy.stats as stats

np.random.seed(42)

sample_size = 25
sample_data = np.random.normal(loc=102, scale=15, size=sample_size)

population_mean = 100

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

n = len(sample_data)

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

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

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.
```

Result:

The test shows **no significant difference** between the sample mean IQ and the population mean of 100.

Hence, we **fail to reject the null hypothesis**.

EXPERIMENT NO : 14

Hypothetical using ANOVA-Test

Aim:

To test whether there is a significant difference in the **mean growth rates** of plants under three different treatments (A, B, and C) using the **One-Way ANOVA** method.

Algorithm:

- Import required libraries: numpy, scipy.stats, and statsmodels.
- Generate three random samples (growth rates) for treatments A, B, and C.
- Perform **One-way ANOVA** using stats.f_oneway().
- Compare the obtained p-value with significance level $\alpha = 0.05$.
- If significant, perform **Tukey's HSD post-hoc test** to find which groups differ.

Code:

```
']: import numpy as np
    import scipy.stats as stats
    from statsmodels.stats.multicomp import pairwise_tukeyhsd

    np.random.seed(42)
    n_plants = 25

    # Generate sample data
    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
    all_data = np.concatenate([growth_A, growth_B, growth_C])
    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}")

    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.")

    # Perform Tukey's HSD post-hoc test
    tukey_results = pairwise_tukeyhsd(all_data, treatment_labels, alpha=0.05)
    print("\nTukey's HSD Post-hoc Test:")
    print(tukey_results)
else:
    print("Fail to reject the null hypothesis: There is no significant difference in mean growth rates among the three treatments.")
```

Result:

The p-value (< 0.05) indicates that there is a **significant difference** in the mean growth rates among treatments A, B, and C.
Tukey's HSD test confirms that **each treatment differs significantly** from the others