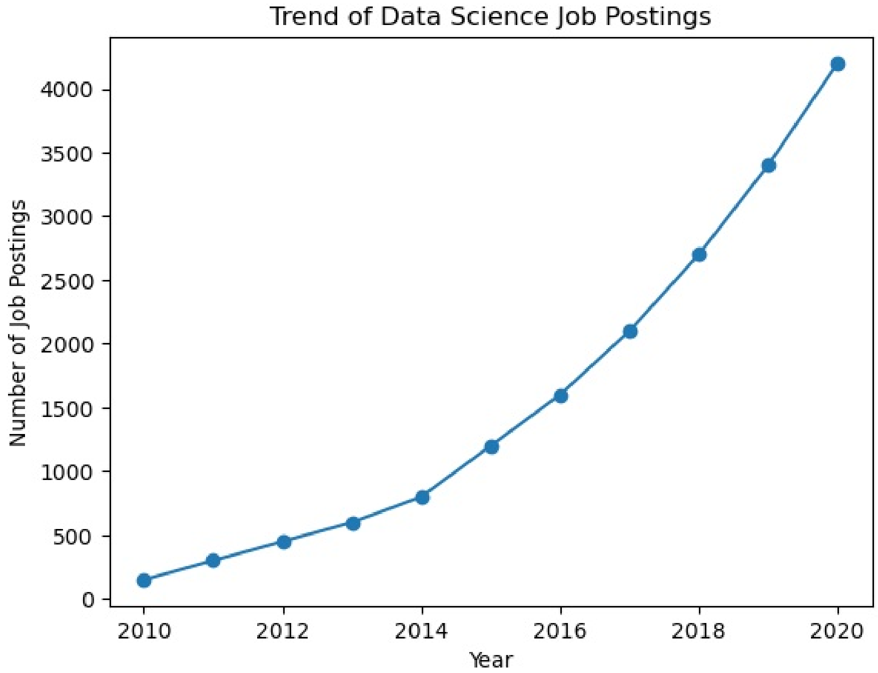
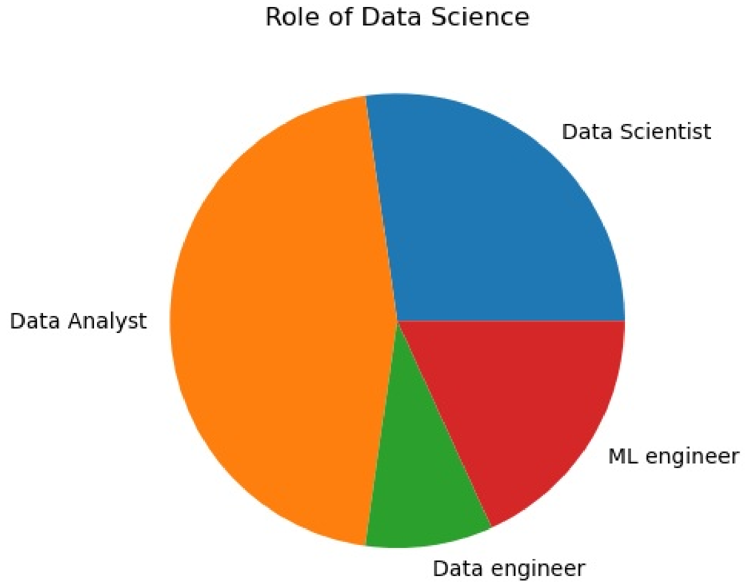
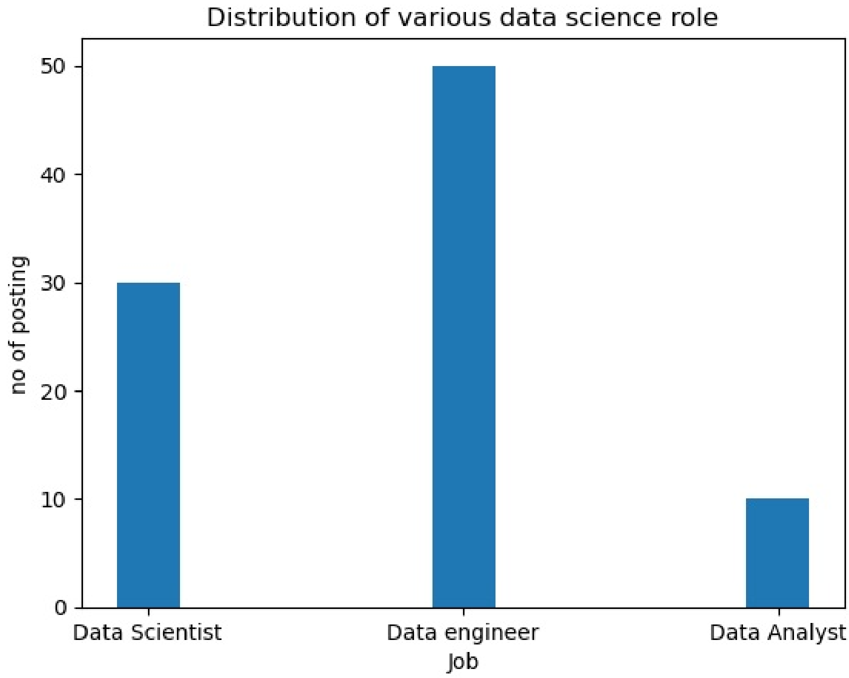
In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
data ={'Year':list(range(2010,2021)),'Job Postings':[150,300,450,600,800,1200,1600,2100,2700,3400,4200]}
df=pd.DataFrame(data)
plt.plot(df['Year'],df['Job Postings'],marker='o')
plt.title('Trend of Data Science Job Postings')
plt.xlabel('Year')
plt.ylabel('Number of Job Postings')
plt.show()
```



In [4]:
```
import pandas as pd
import matplotlib.pyplot as plt
name=['Data Scientist','Data Analyst','Data engineer','ML engineer']
data=[30,50,10,20]
plt.pie(data, labels=name)
plt.title('Role of Data Science')
plt.show()
```



In [6]:
```
import matplotlib.pyplot as plt
roles=['Data Scientist','Data engineer','Data Analyst']
posting=[30,50,10]
plt.bar(roles,posting,width=0.2)
plt.title('Distribution of various data science role')
plt.xlabel('Job')
plt.ylabel('no of posting')
plt.show()
```



In [8]:

```
import pandas as pd
sd=pd.DataFrame(
        {
            "ID":[1,2,3],
            "Name":['Raj','Riya','Ram'],
            "Age":[23,18,25]
        })
print('Structured Data\n',sd)
print('')
unsd='This is a unstructured data which contain audio, video,combination of data type.'
print('Unstructured Data\n',unsd,'\n')
semisd='{"Name":"Raju","Age":40,"Job":"Software engineer","Hobby":"Gamming"}'
print('Semistructured Data\n',semisd)
```

```
Structured Data
    ID  Name  Age
0   1   Raj   23
1   2  Riya   18
2   3   Ram   25

Unstructured Data
 This is a unstructured data which contain audio, video,combination of data type.

Semistructured Data
 {"Name":"Raju","Age":40,"Job":"Software engineer","Hobby":"Gamming"}
```

In [10]:

```
from cryptography.fernet import Fernet
key=Fernet.generate_key()
f=Fernet(key)
token=f.encrypt(b'This is computer science department.')
token
b'...'
f.decrypt(token)
b'This is computer science department'
key=Fernet.generate_key()
cipher_suite=Fernet(key)
plain_text=b'This is computer science department.'
cipher_text=cipher_suite.encrypt(plain_text)
decrypted_text=cipher_suite.decrypt(cipher_text)
print('Original data:',plain_text)
print('Encrypted data:',cipher_text)
print('Decrypted data:',decrypted_text)
```

```
Original data: b'This is computer science department.'
Encrypted data: b'gAAAAABnPyN-hn0B7prAdWoWjZnlGs1VBtQQd0doZMV4J2SvwrfeXHLTgOrIzaOo83QrjHK5sdDXxZq6UIuZ0LNb3iuO3o
LOcQxb0GQPPLrRrjdA7pSpML9LD-HErYOdbAGooGb7BBn9'
Decrypted data: b'This is computer science department.'
```

In [ ]:

In [27]:
```
import numpy as np
import pandas as pd
list=[[1,'Smith',50000],[2,'Jones',60000]]
df=pd.DataFrame(list)
df
```

Out[27]:

	0	1	2
0	1	Smith	50000
1	2	Jones	60000

In [29]:
```
df=pd.read_csv("melb_data.csv")
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18396 entries, 0 to 18395
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Unnamed: 0     18396 non-null  int64
 1   Suburb         18396 non-null  object
 2   Address        18396 non-null  object
 3   Rooms          18396 non-null  int64
 4   Type           18396 non-null  object
 5   Price          18396 non-null  float64
 6   Method         18396 non-null  object
 7   SellerG        18396 non-null  object
 8   Date           18396 non-null  object
 9   Distance       18395 non-null  float64
 10  Postcode       18395 non-null  float64
 11  Bedroom2       14927 non-null  float64
 12  Bathroom       14925 non-null  float64
 13  Car            14820 non-null  float64
 14  Landsize       13603 non-null  float64
 15  BuildingArea   7762 non-null   float64
 16  YearBuilt      8958 non-null   float64
 17  CouncilArea    12233 non-null  object
 18  Lattitude      15064 non-null  float64
 19  Longtitude     15064 non-null  float64
 20  Regionname     18395 non-null  object
 21  Propertycount  18395 non-null  float64
dtypes: float64(12), int64(2), object(8)
memory usage: 3.1+ MB
```

Out[29]:

	Unnamed: 0	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	La
count	18396.000000	18396.000000	1.839600e+04	18395.000000	18395.000000	14927.000000	14925.000000	14820.000000	13603.0
mean	11826.787073	2.935040	1.056697e+06	10.389986	3107.140147	2.913043	1.538492	1.615520	558.1
std	6800.710448	0.958202	6.419217e+05	6.009050	95.000995	0.964641	0.689311	0.955916	3987.3
min	1.000000	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.0
25%	5936.750000	2.000000	6.330000e+05	6.300000	3046.000000	2.000000	1.000000	1.000000	176.5
50%	11820.500000	3.000000	8.800000e+05	9.700000	3085.000000	3.000000	1.000000	2.000000	440.0
75%	17734.250000	3.000000	1.302000e+06	13.300000	3149.000000	3.000000	2.000000	2.000000	651.0
max	23546.000000	12.000000	9.000000e+06	48.100000	3978.000000	20.000000	8.000000	10.000000	433014.0

In [9]:
```
df.head()
df.tail()
```

Out[9]:

	Unnamed: 0	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	...	Bathroom	Ca
18391	23540	Williamstown	8/2 Thompson St	2	t	622500.0	SP	Greg	26/08/2017	6.8	...	2.0	1.(
18392	23541	Williamstown	96 Verdon St	4	h	2500000.0	PI	Sweeney	26/08/2017	6.8	...	1.0	5.(
18393	23544	Yallambie	17 Amaroo Wy	4	h	1100000.0	S	Buckingham	26/08/2017	12.7	...	3.0	2.(
18394	23545	Yarraville	6 Agnes St	4	h	1285000.0	SP	Village	26/08/2017	6.3	...	1.0	1.(
18395	23546	Yarraville	33 Freeman St	4	h	1050000.0	VB	Village	26/08/2017	6.3	...	2.0	2.(

5 rows × 22 columns

In [21]:

```
df.Price.mean()
df.Price.median()
df.Price.mode()
```

Out[21]:

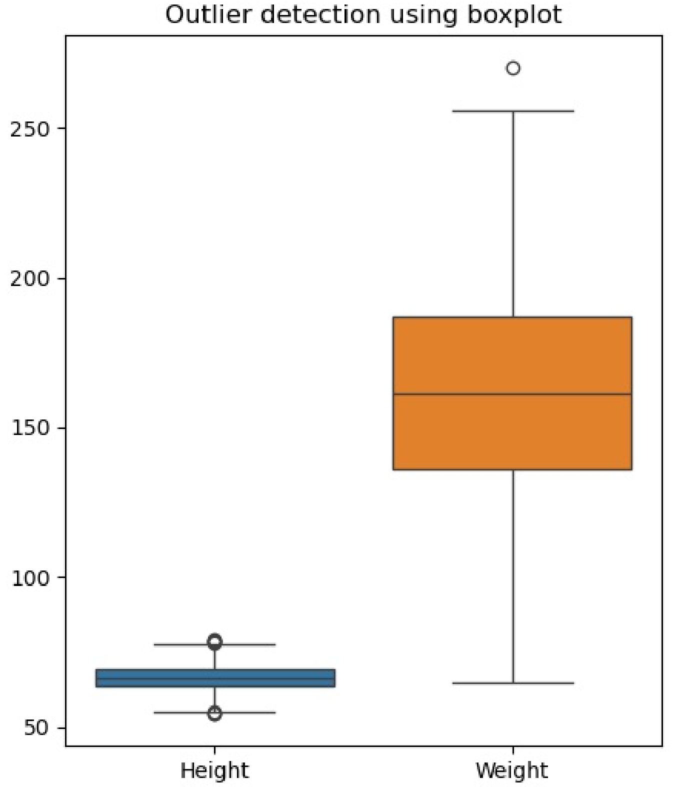
```
0    600000.0
Name: Price, dtype: float64
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [4]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
#1.Load the data in DataFrame
df=pd.read_csv(r"D:\230701164fds\weight-height.csv")
df=pd.DataFrame(df)
print(df)

#2.Detection of outliers using boxplot
plt.figure(figsize=(5,6))
sns.boxplot(data=df)
plt.title('Outlier detection using boxplot')
plt.show()
```

```
      Gender     Height      Weight
0       Male  73.847017  241.893563
1       Male  68.781904  162.310473
2       Male  74.110105  212.740856
3       Male  71.730978  220.042470
4       Male  69.881796  206.349801
...      ...        ...         ...
9995  Female  66.172652  136.777454
9996  Female  67.067155  170.867906
9997  Female  63.867992  128.475319
9998  Female  69.034243  163.852461
9999  Female  61.944246  113.649103

[10000 rows x 3 columns]
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [5]:
```
import numpy as np
import pandas as pd
df=pd.read_csv(r"C:\Users\HP\Downloads\Hotel_Dataset.csv")
df
```

Out[5]:

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

In [7]:
```
df.duplicated()
```

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

In [9]:
```
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
```

In [11]:
```
df.drop_duplicates(inplace=True)
df
```

Out[11]:

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

In [13]:
```
len(df)
```

Out[13]:
```
10
```

In [15]:
```
index=np.array(list(range(0,len(df))))
df.set_index(index,inplace=True)
index
```

Out[15]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [17]:
```
df.drop(['Age_Group.1'],axis=1,inplace=True)
df
```

Out[17]:

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

In [23]:
```
df.Age_Group.unique()
df
```

Out[23]:

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

In [25]:
```
df.Hotel.unique()
```

Out[25]:
```
array(['Ibis', 'LemonTree', 'RedFox', 'Ibys'], dtype=object)
```

In [31]:

In [ ]:

In [3]:
```
import numpy as np
import pandas as pd
df=pd.read_csv(r"C:\Users\HP\Downloads\archive (9)\Data.csv")
df
```

Out[3]:

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

In [5]:
```
df.Country.mode()
```

Out[5]:
```
0    France
Name: Country, dtype: object
```

In [7]:
```
df.Country.mode()[0]
```

Out[7]:
```
'France'
```

In [9]:
```
type(df.Country.mode())
```

Out[9]:
```
pandas.core.series.Series
```

In [13]:
```
pd.get_dummies(df.Country)
```

Out[13]:

	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	False	True	False
9	True	False	False

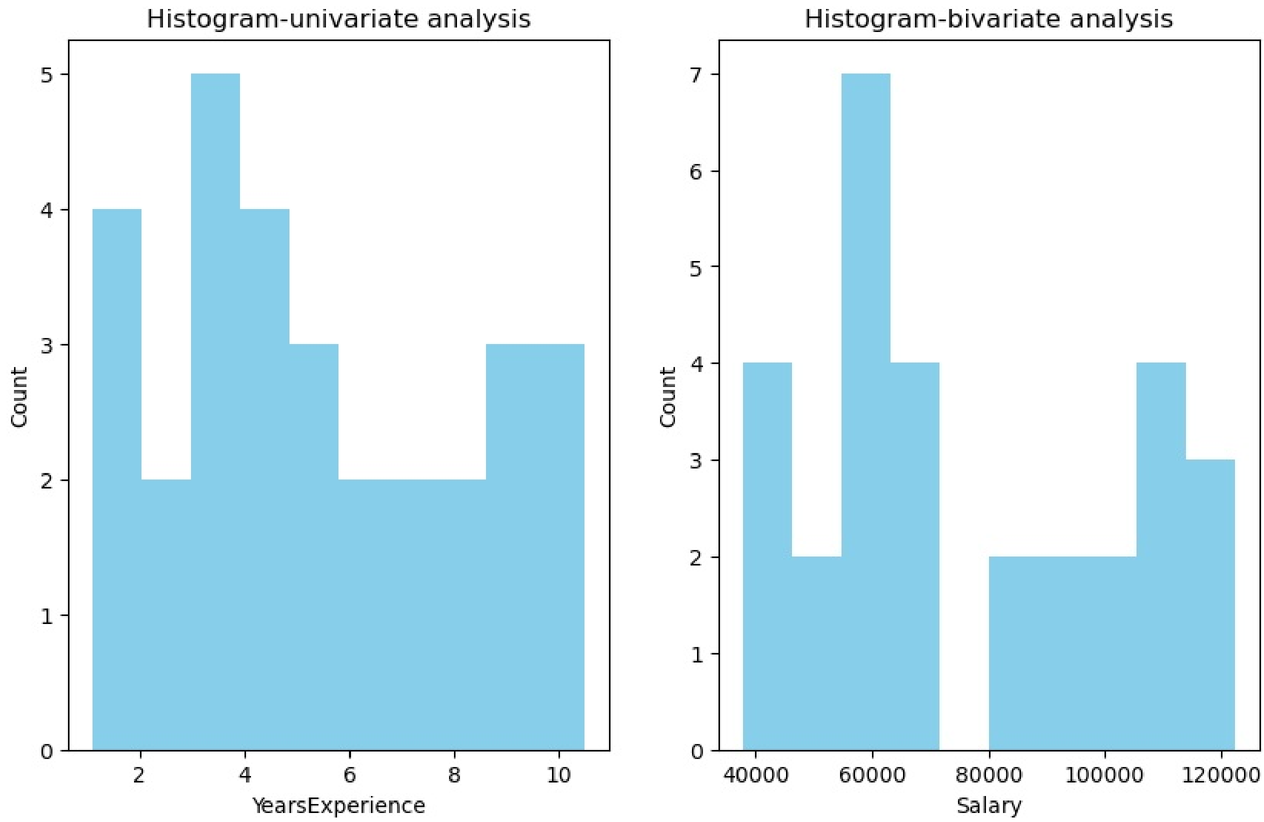
In [17]:
```
updated_dataset=pd.concat([pd.get_dummies(df.Country),df.iloc[:,[1,2,3]]],axis=1)
df.info()
```

```
<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: 452.0+ bytes
```

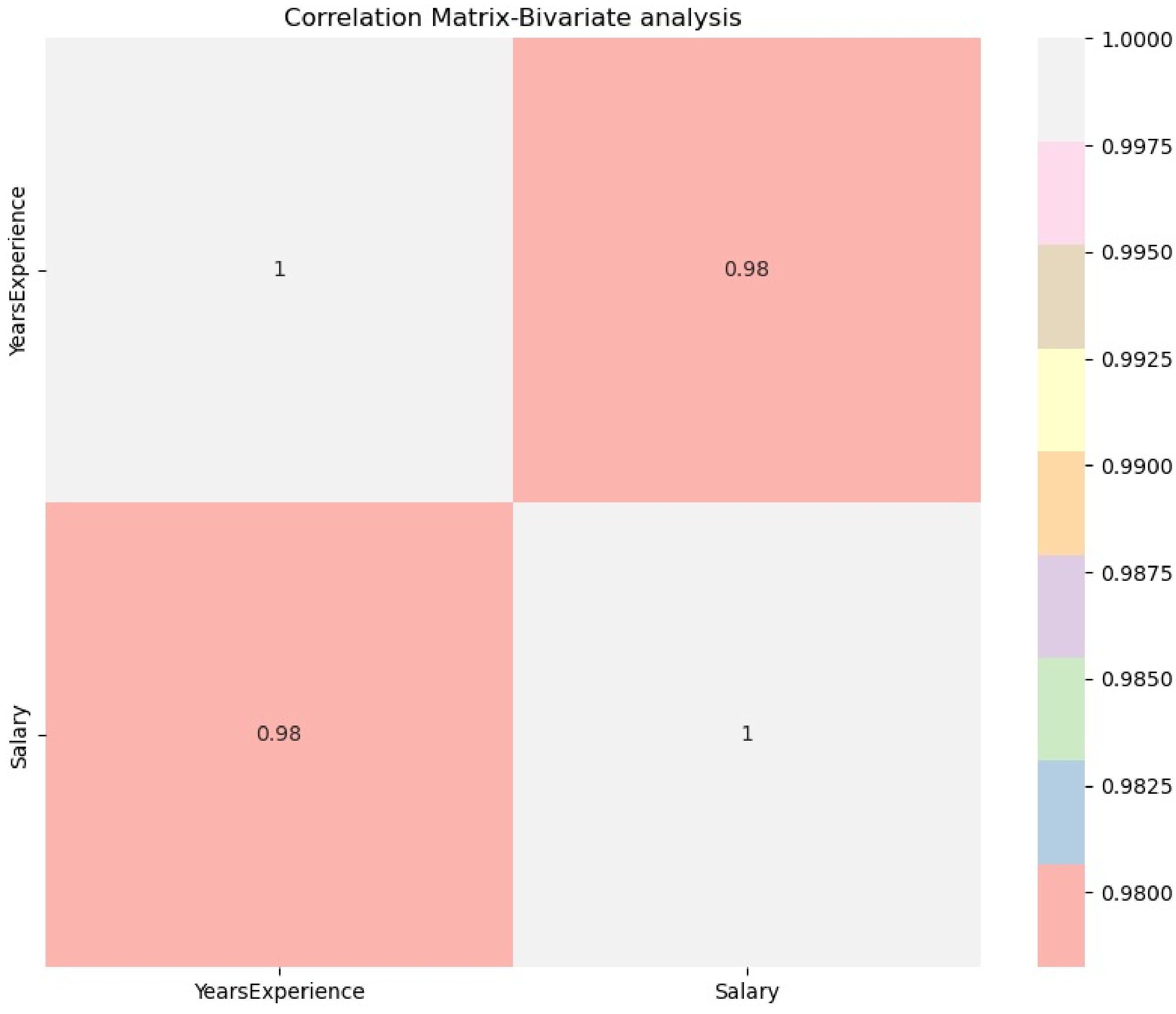
In [ ]:

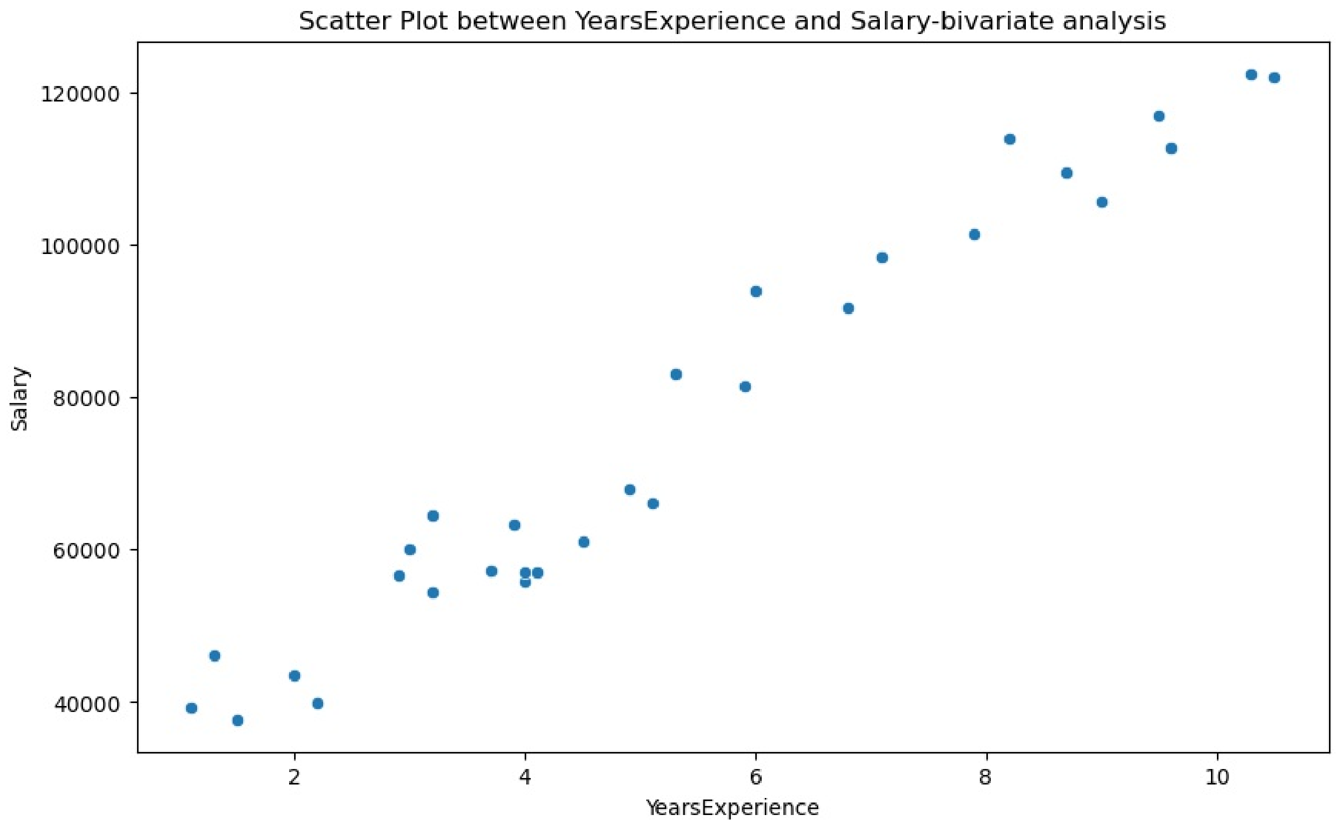
In [21]:
```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
df=pd.read_csv(r"D:\230701164fds\salary_data.csv")
df=pd.DataFrame(df)
print(df)


plt.figure(figsize=(10,6))
plt.subplot(1, 2, 1)
plt.hist(df['YearsExperience'],bins=10,color='skyblue')
plt.title('Histogram-univariate analysis')
plt.xlabel('YearsExperience')
plt.ylabel('Count')

plt.subplot(1, 2, 2)
plt.hist(df['Salary'],bins=10,color='skyblue')
plt.title('Histogram-bivariate analysis')
plt.xlabel('Salary')
plt.ylabel('Count')


plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='Pastel1')
plt.title('Correlation Matrix-Bivariate analysis')
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='YearsExperience', y='Salary')  # Replace with your numerical columns
plt.title('Scatter Plot between YearsExperience and Salary-bivariate analysis')
plt.show()
```

```
    YearsExperience    Salary
0               1.1   39343.0
1               1.3   46205.0
2               1.5   37731.0
3               2.0   43525.0
4               2.2   39891.0
5               2.9   56642.0
6               3.0   60150.0
7               3.2   54445.0
8               3.2   64445.0
9               3.7   57189.0
10              3.9   63218.0
11              4.0   55794.0
12              4.0   56957.0
13              4.1   57081.0
14              4.5   61111.0
15              4.9   67938.0
16              5.1   66029.0
17              5.3   83088.0
18              5.9   81363.0
19              6.0   93940.0
20              6.8   91738.0
21              7.1   98273.0
22              7.9  101302.0
23              8.2  113812.0
24              8.7  109431.0
25              9.0  105582.0
26              9.5  116969.0
27              9.6  112635.0
28             10.3  122391.0
29             10.5  121872.0
```



Histogram-univariate analysis
Count
YearsExperience
Histogram-bivariate analysis
Count
Salary

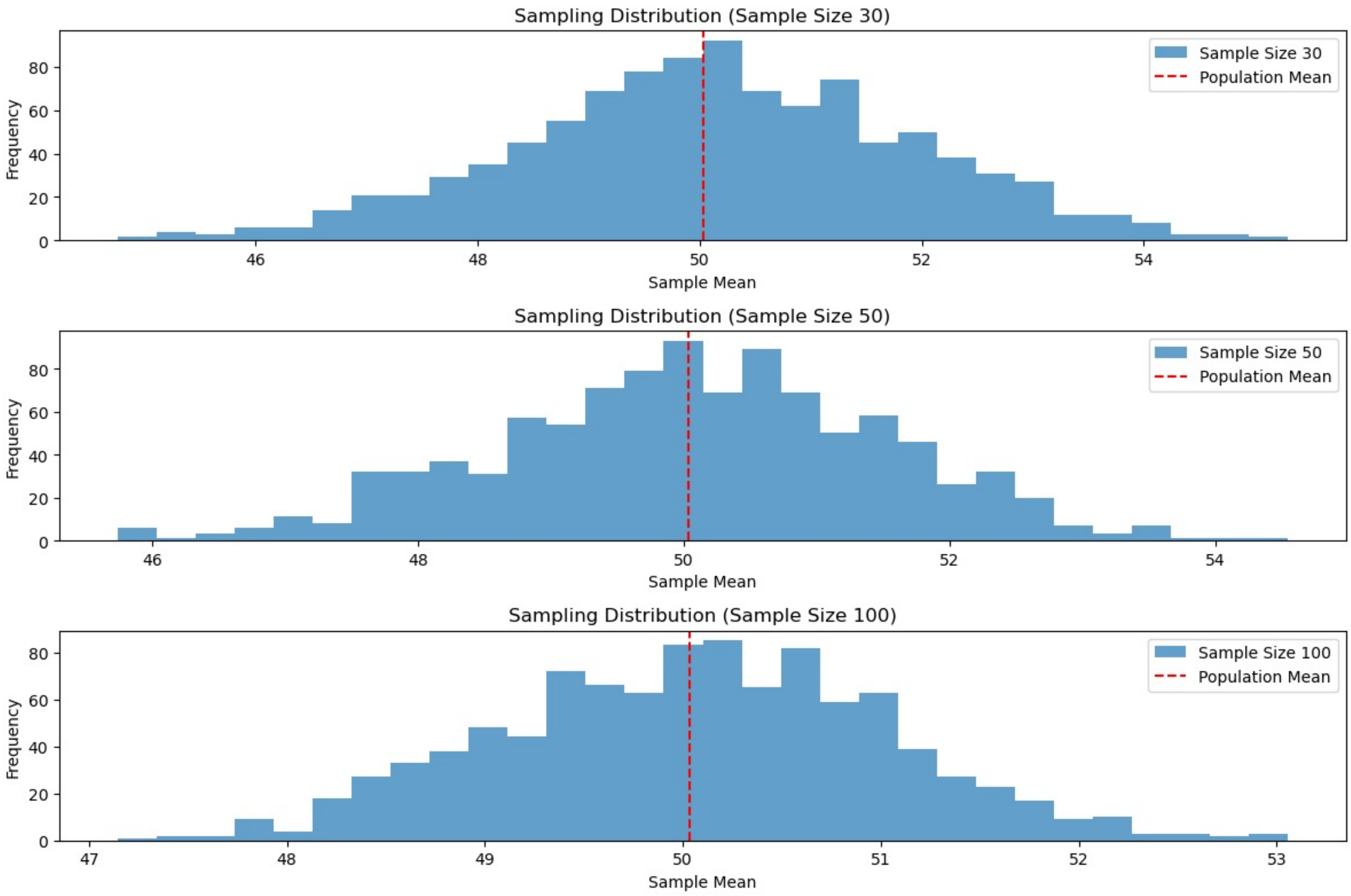

Correlation Matrix-Bivariate analysis
YearsExperience
Salary
1
0.98
0.98
1
YearsExperience
Salary
1.0000
0.9975
0.9950
0.9925
0.9900
0.9875
0.9850
0.9825
0.9800



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define population parameters
population_mean = 50
population_std = 10
population_size = 100000

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

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

# Initialize dictionary for sample means
for size in sample_sizes:
    sample_means[size] = []

# Generate samples and calculate sample means for each sample size
for size in sample_sizes:  # Corrected this loop
    for _ in range(num_samples):
        sample = np.random.choice(population, size=size, replace=False)
        sample_means[size].append(np.mean(sample))

# Plot the sampling distributions
plt.figure(figsize=(12, 8))
for i, size in enumerate(sample_sizes):
    plt.subplot(len(sample_sizes), 1, i + 1)
    plt.hist(sample_means[size], bins=30, alpha=0.7, label=f'Sample Size {size}')
    plt.axvline(np.mean(population), color='red', linestyle='dashed', linewidth=1.5,
                label='Population Mean')
    plt.title(f'Sampling Distribution (Sample Size {size})')
    plt.xlabel('Sample Mean')
    plt.ylabel('Frequency')
    plt.legend()

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



In [ ]:

In [9]:
```python
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 standarddeviation

# 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 issignificantly different from 150 grams.")
else:
    print("Fail to reject the null hypothesis: There is nosignificant difference in average weight from 150 gra
```

```
Sample Mean: 150.20
Z-Statistic: 0.6406
P-Value: 0.5218
Fail to reject the null hypothesis: There is nosignificant difference in average weight from 150 grams.
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [1]:
```
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 standarddeviation
# Number of observations
n = len(sample_data)
# Calculate the T-statistic and p-value
t_statistic, p_value = stats.ttest_1samp(sample_data,population_mean)
# Print results
print(f"Sample Mean: {sample_mean:.2f}")
print(f"T-Statistic: {t_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")
# Decision based on the significance level
alpha = 0.05
if p_value < alpha:
  print("Reject the null hypothesis: The average IQ score is significantly different from 100.")
else:
  print("Fail to reject the null hypothesis: There is no significant difference in average IQ score from 100.")
```

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

In [ ]:

In [7]:

```python
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 tre
else:
   print("Fail to reject the null hypothesis: There is no significant difference in mean growth rates among the
# Additional: Post-hoc analysis (Tukey's HSD) if ANOVA is

if p_value < alpha:
   from statsmodels.stats.multicomp import pairwise_tukeyhsd
   tukey_results = pairwise_tukeyhsd(all_data, treatment_labels,alpha=0.05)
   print("\nTukey's HSD Post-hoc Test:")
   print(tukey_results)
```

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

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

Tukey's HSD Post-hoc Test:
Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
group1 group2 meandiff p-adj   lower  upper  reject
----------------------------------------------------
     A      B   1.4647 0.0877 -0.1683 3.0977  False
     A      C   5.5923    0.0  3.9593 7.2252   True
     B      C   4.1276    0.0  2.4946 5.7605   True
----------------------------------------------------
```

In [ ]:

In [3]:
```python
import numpy as np
import pandas as pd
df=pd.read_csv(r"C:\Users\HP\Downloads\archive (9)\Data.csv")
df.head()
```

Out[3]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes

In [5]:
```python
df.Country.fillna(df.Country.mode()[0],inplace=True)
features=df.iloc[:,:-1].values
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_24408\3424832005.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

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


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

In [7]:
```python
label=df.iloc[:,-1].values


from sklearn.impute import SimpleImputer
age=SimpleImputer(strategy="mean",missing_values=np.nan)
Salary=SimpleImputer(strategy="mean",missing_values=np.nan)
age.fit(features[:,[1]])
```

Out[7]:
```
▾ SimpleImputer
SimpleImputer()
```

In [23]:
```python
Salary.fit(features[:,[2]])
```

Out[23]:
```
▾ SimpleImputer
SimpleImputer()
```

In [13]:
```python
features[:,[1]]=age.transform(features[:,[1]])
features[:,[2]]=Salary.transform(features[:,[2]])
features
```

Out[13]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 63777.77777777778],
       ['France', 35.0, 58000.0],
       ['Spain', 38.77777777777778, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

In [15]:
```python
from sklearn.preprocessing import OneHotEncoder
oh = OneHotEncoder(sparse_output=False)
Country=oh.fit_transform(features[:,[0]])
Country
```

Out[15]:
```
array([[1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.]])
```

In [17]:
```
final_set=np.concatenate((Country,features[:,[1,2]]),axis=1)
final_set
```

Out[17]:
```
array([[1.0, 0.0, 0.0, 44.0, 72000.0],
       [0.0, 0.0, 1.0, 27.0, 48000.0],
       [0.0, 1.0, 0.0, 30.0, 54000.0],
       [0.0, 0.0, 1.0, 38.0, 61000.0],
       [0.0, 1.0, 0.0, 40.0, 63777.77777777778],
       [1.0, 0.0, 0.0, 35.0, 58000.0],
       [0.0, 0.0, 1.0, 38.77777777777778, 52000.0],
       [1.0, 0.0, 0.0, 48.0, 79000.0],
       [0.0, 1.0, 0.0, 50.0, 83000.0],
       [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

In [19]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit(final_set)
feat_standard_scaler=sc.transform(final_set)
feat_standard_scaler
```

Out[19]:
```
array([[ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
         7.58874362e-01,  7.49473254e-01],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
        -1.71150388e+00, -1.43817841e+00],
       [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
        -1.27555478e+00, -8.91265492e-01],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
        -1.13023841e-01, -2.53200424e-01],
       [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
         1.77608893e-01,  6.63219199e-16],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
        -5.48972942e-01, -5.26656882e-01],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
         0.00000000e+00, -1.07356980e+00],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
         1.34013983e+00,  1.38753832e+00],
       [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
         1.63077256e+00,  1.75214693e+00],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
        -2.58340208e-01,  2.93712492e-01]])
```

In [21]:
```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler(feature_range=(0,1))
mms.fit(final_set)
feat_minmax_scaler=mms.transform(final_set)
feat_minmax_scaler
```

Out[21]:
```
array([[1.        , 0.        , 0.        , 0.73913043, 0.68571429],
       [0.        , 0.        , 1.        , 0.        , 0.        ],
       [0.        , 1.        , 0.        , 0.13043478, 0.17142857],
       [0.        , 0.        , 1.        , 0.47826087, 0.37142857],
       [0.        , 1.        , 0.        , 0.56521739, 0.45079365],
       [1.        , 0.        , 0.        , 0.34782609, 0.28571429],
       [0.        , 0.        , 1.        , 0.51207729, 0.11428571],
       [1.        , 0.        , 0.        , 0.91304348, 0.88571429],
       [0.        , 1.        , 0.        , 1.        , 1.        ],
       [1.        , 0.        , 0.        , 0.43478261, 0.54285714]])
```

In [ ]:

In [3]:
```python
import numpy as np
import pandas as pd
df=pd.read_csv(r"C:\Users\HP\Downloads\Salary_data.csv")
df
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]:
```python
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
```

In [7]:
```python
df.describe()
```

Out[7]:

	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 [17]:
```python
features=df.iloc[:,[0]].values
label=df.iloc[:,[1]].values
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

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

# Initialize the Linear Regression model
model = LinearRegression()

# Train the model using the training data
model.fit(x_train, y_train)
```

Out[17]:

```
▾ LinearRegression
LinearRegression()
```

In [19]:
```python
model.score(x_train,y_train)

model.score(x_test,y_test)

model.coef_

model.intercept_

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)
```

In [23]:
```
print("Estimated Salary for {} years of experience is {}: " .format(yr_of_exp,Salary))
```

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

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [1]:
```python
import numpy as np
import pandas as pd
df=pd.read_csv(r"C:\Users\HP\Downloads\Social_Network_Ads (1).csv")
df

df.head()
```

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

In [7]:
```python
features=df.iloc[:,[2,3]].values
label=df.iloc[:,4].values
features
```

Out[7]:
```
array([[     19,  19000],
       [     35,  20000],
       [     26,  43000],
       [     27,  57000],
       [     19,  76000],
       [     27,  58000],
       [     27,  84000],
       [     32, 150000],
       [     25,  33000],
       [     35,  65000],
       [     26,  80000],
       [     26,  52000],
       [     20,  86000],
       [     32,  18000],
       [     18,  82000],
       [     29,  80000],
       [     47,  25000],
       [     45,  26000],
       [     46,  28000],
       [     48,  29000],
       [     45,  22000],
       [     47,  49000],
       [     48,  41000],
       [     45,  22000],
       [     46,  23000],
       [     47,  20000],
       [     49,  28000],
       [     47,  30000],
       [     29,  43000],
       [     31,  18000],
       [     31,  74000],
       [     27, 137000],
       [     21,  16000],
       [     28,  44000],
       [     27,  90000],
       [     35,  27000],
       [     33,  28000],
       [     30,  49000],
       [     26,  72000],
       [     27,  31000],
       [     27,  17000],
       [     33,  51000],
       [     35, 108000],
       [     30,  15000],
       [     28,  84000],
       [     23,  20000],
       [     25,  79000],
       [     27,  54000],
       [     30, 135000],
       [     31,  89000],
       [     24,  32000],
       [     18,  44000],
       [     29,  83000],
       [     35,  23000],
       [     27,  58000],
       [     24,  55000],
       [     23,  48000],
       [     28,  79000],
       [     22,  18000],
```

```
       [    32, 117000],
       [    27,  20000],
       [    25,  87000],
       [    23,  66000],
       [    32, 120000],
       [    59,  83000],
       [    24,  58000],
       [    24,  19000],
       [    23,  82000],
       [    22,  63000],
       [    31,  68000],
       [    25,  80000],
       [    24,  27000],
       [    20,  23000],
       [    33, 113000],
       [    32,  18000],
       [    34, 112000],
       [    18,  52000],
       [    22,  27000],
       [    28,  87000],
       [    26,  17000],
       [    30,  80000],
       [    39,  42000],
       [    20,  49000],
       [    35,  88000],
       [    30,  62000],
       [    31, 118000],
       [    24,  55000],
       [    28,  85000],
       [    26,  81000],
       [    35,  50000],
       [    22,  81000],
       [    30, 116000],
       [    26,  15000],
       [    29,  28000],
       [    29,  83000],
       [    35,  44000],
       [    35,  25000],
       [    28, 123000],
       [    35,  73000],
       [    28,  37000],
       [    27,  88000],
       [    28,  59000],
       [    32,  86000],
       [    33, 149000],
       [    19,  21000],
       [    21,  72000],
       [    26,  35000],
       [    27,  89000],
       [    26,  86000],
       [    38,  80000],
       [    39,  71000],
       [    37,  71000],
       [    38,  61000],
       [    37,  55000],
       [    42,  80000],
       [    40,  57000],
       [    35,  75000],
       [    36,  52000],
       [    40,  59000],
       [    41,  59000],
       [    36,  75000],
       [    37,  72000],
       [    40,  75000],
       [    35,  53000],
       [    41,  51000],
       [    39,  61000],
       [    42,  65000],
       [    26,  32000],
       [    30,  17000],
       [    26,  84000],
       [    31,  58000],
       [    33,  31000],
       [    30,  87000],
       [    21,  68000],
       [    28,  55000],
       [    23,  63000],
       [    20,  82000],
       [    30, 107000],
       [    28,  59000],
       [    19,  25000],
       [    19,  85000],
       [    18,  68000],
```

```
[    35,  59000],
[    30,  89000],
[    34,  25000],
[    24,  89000],
[    27,  96000],
[    41,  30000],
[    29,  61000],
[    20,  74000],
[    26,  15000],
[    41,  45000],
[    31,  76000],
[    36,  50000],
[    40,  47000],
[    31,  15000],
[    46,  59000],
[    29,  75000],
[    26,  30000],
[    32, 135000],
[    32, 100000],
[    25,  90000],
[    37,  33000],
[    35,  38000],
[    33,  69000],
[    18,  86000],
[    22,  55000],
[    35,  71000],
[    29, 148000],
[    29,  47000],
[    21,  88000],
[    34, 115000],
[    26, 118000],
[    34,  43000],
[    34,  72000],
[    23,  28000],
[    35,  47000],
[    25,  22000],
[    24,  23000],
[    31,  34000],
[    26,  16000],
[    31,  71000],
[    32, 117000],
[    33,  43000],
[    33,  60000],
[    31,  66000],
[    20,  82000],
[    33,  41000],
[    35,  72000],
[    28,  32000],
[    24,  84000],
[    19,  26000],
[    29,  43000],
[    19,  70000],
[    28,  89000],
[    34,  43000],
[    30,  79000],
[    20,  36000],
[    26,  80000],
[    35,  22000],
[    35,  39000],
[    49,  74000],
[    39, 134000],
[    41,  71000],
[    58, 101000],
[    47,  47000],
[    55, 130000],
[    52, 114000],
[    40, 142000],
[    46,  22000],
[    48,  96000],
[    52, 150000],
[    59,  42000],
[    35,  58000],
[    47,  43000],
[    60, 108000],
[    49,  65000],
[    40,  78000],
[    46,  96000],
[    59, 143000],
[    41,  80000],
[    35,  91000],
[    37, 144000],
[    60, 102000],
[    35,  60000],
```

```
[    37,  53000],
[    36, 126000],
[    56, 133000],
[    40,  72000],
[    42,  80000],
[    35, 147000],
[    39,  42000],
[    40, 107000],
[    49,  86000],
[    38, 112000],
[    46,  79000],
[    40,  57000],
[    37,  80000],
[    46,  82000],
[    53, 143000],
[    42, 149000],
[    38,  59000],
[    50,  88000],
[    56, 104000],
[    41,  72000],
[    51, 146000],
[    35,  50000],
[    57, 122000],
[    41,  52000],
[    35,  97000],
[    44,  39000],
[    37,  52000],
[    48, 134000],
[    37, 146000],
[    50,  44000],
[    52,  90000],
[    41,  72000],
[    40,  57000],
[    58,  95000],
[    45, 131000],
[    35,  77000],
[    36, 144000],
[    55, 125000],
[    35,  72000],
[    48,  90000],
[    42, 108000],
[    40,  75000],
[    37,  74000],
[    47, 144000],
[    40,  61000],
[    43, 133000],
[    59,  76000],
[    60,  42000],
[    39, 106000],
[    57,  26000],
[    57,  74000],
[    38,  71000],
[    49,  88000],
[    52,  38000],
[    50,  36000],
[    59,  88000],
[    35,  61000],
[    37,  70000],
[    52,  21000],
[    48, 141000],
[    37,  93000],
[    37,  62000],
[    48, 138000],
[    41,  79000],
[    37,  78000],
[    39, 134000],
[    49,  89000],
[    55,  39000],
[    37,  77000],
[    35,  57000],
[    36,  63000],
[    42,  73000],
[    43, 112000],
[    45,  79000],
[    46, 117000],
[    58,  38000],
[    48,  74000],
[    37, 137000],
[    37,  79000],
[    40,  60000],
[    42,  54000],
[    51, 134000],
[    47, 113000],
```

```
[    36, 125000],
[    38,  50000],
[    42,  70000],
[    39,  96000],
[    38,  50000],
[    49, 141000],
[    39,  79000],
[    39,  75000],
[    54, 104000],
[    35,  55000],
[    45,  32000],
[    36,  60000],
[    52, 138000],
[    53,  82000],
[    41,  52000],
[    48,  30000],
[    48, 131000],
[    41,  60000],
[    41,  72000],
[    42,  75000],
[    36, 118000],
[    47, 107000],
[    38,  51000],
[    48, 119000],
[    42,  65000],
[    40,  65000],
[    57,  60000],
[    36,  54000],
[    58, 144000],
[    35,  79000],
[    38,  55000],
[    39, 122000],
[    53, 104000],
[    35,  75000],
[    38,  65000],
[    47,  51000],
[    47, 105000],
[    41,  63000],
[    53,  72000],
[    54, 108000],
[    39,  77000],
[    38,  61000],
[    38, 113000],
[    37,  75000],
[    42,  90000],
[    37,  57000],
[    36,  99000],
[    60,  34000],
[    54,  70000],
[    41,  72000],
[    40,  71000],
[    42,  54000],
[    43, 129000],
[    53,  34000],
[    47,  50000],
[    42,  79000],
[    42, 104000],
[    59,  29000],
[    58,  47000],
[    46,  88000],
[    38,  71000],
[    54,  26000],
[    60,  46000],
[    60,  83000],
[    39,  73000],
[    59, 130000],
[    37,  80000],
[    46,  32000],
[    46,  74000],
[    42,  53000],
[    41,  87000],
[    58,  23000],
[    42,  64000],
[    48,  33000],
[    44, 139000],
[    49,  28000],
[    57,  33000],
[    56,  60000],
[    49,  39000],
[    39,  71000],
[    47,  34000],
[    48,  35000],
[    48,  33000],
```

```
       [    47,  23000],
       [    45,  45000],
       [    60,  42000],
       [    39,  59000],
       [    46,  41000],
       [    51,  23000],
       [    50,  20000],
       [    36,  33000],
       [    49,  36000]], dtype=int64)
```

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

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

In [13]:
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Loop to try different random states for train-test split
for i in range(1, 401):
    # Split the dataset with a random state
    x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.5, random_state=i)

    # Initialize the Logistic Regression model
    model = LogisticRegression()

    # Train the model
    model.fit(x_train, y_train)

    # Calculate scores for train and test sets
    train_score = model.score(x_train, y_train)
    test_score = model.score(x_test, y_test)

    # Check if test score is greater than train score
    if test_score > train_score:
        print("Test Score: {:.4f}, Train Score: {:.4f}, Random State: {}".format(test_score, train_score, i))
```

```
Test Score: 0.8450, Train Score: 0.8400, Random State: 3
Test Score: 0.8550, Train Score: 0.8300, Random State: 5
Test Score: 0.8550, Train Score: 0.8050, Random State: 6
Test Score: 0.8650, Train Score: 0.8150, Random State: 7
Test Score: 0.8500, Train Score: 0.8350, Random State: 8
Test Score: 0.8550, Train Score: 0.8300, Random State: 10
Test Score: 0.8400, Train Score: 0.8350, Random State: 13
Test Score: 0.8700, Train Score: 0.8650, Random State: 15
Test Score: 0.8750, Train Score: 0.8450, Random State: 17
Test Score: 0.8700, Train Score: 0.8350, Random State: 18
Test Score: 0.8250, Train Score: 0.8200, Random State: 20
Test Score: 0.8550, Train Score: 0.8200, Random State: 21
Test Score: 0.8650, Train Score: 0.8400, Random State: 22
Test Score: 0.8700, Train Score: 0.8250, Random State: 27
Test Score: 0.8500, Train Score: 0.8400, Random State: 29
Test Score: 0.8500, Train Score: 0.8350, Random State: 37
Test Score: 0.8500, Train Score: 0.8350, Random State: 42
Test Score: 0.8950, Train Score: 0.8250, Random State: 46
Test Score: 0.8500, Train Score: 0.8150, Random State: 47
Test Score: 0.8550, Train Score: 0.8350, Random State: 48
Test Score: 0.8600, Train Score: 0.8450, Random State: 51
Test Score: 0.8450, Train Score: 0.8400, Random State: 52
Test Score: 0.8700, Train Score: 0.8100, Random State: 54
Test Score: 0.8550, Train Score: 0.7950, Random State: 56
Test Score: 0.8500, Train Score: 0.8400, Random State: 59
Test Score: 0.8750, Train Score: 0.8550, Random State: 61
Test Score: 0.8750, Train Score: 0.8550, Random State: 64
Test Score: 0.8550, Train Score: 0.8250, Random State: 65
Test Score: 0.8700, Train Score: 0.8150, Random State: 68
Test Score: 0.8350, Train Score: 0.8300, Random State: 72
```

```
Test Score: 0.8350, Train Score: 0.8200, Random State: 73
Test Score: 0.8750, Train Score: 0.8400, Random State: 74
Test Score: 0.8750, Train Score: 0.8500, Random State: 75
Test Score: 0.8650, Train Score: 0.8400, Random State: 76
Test Score: 0.8550, Train Score: 0.8300, Random State: 77
Test Score: 0.8550, Train Score: 0.8250, Random State: 79
Test Score: 0.8600, Train Score: 0.8250, Random State: 82
Test Score: 0.8650, Train Score: 0.8350, Random State: 84
Test Score: 0.8800, Train Score: 0.8300, Random State: 85
Test Score: 0.8550, Train Score: 0.8350, Random State: 86
Test Score: 0.8700, Train Score: 0.8150, Random State: 88
Test Score: 0.9050, Train Score: 0.8050, Random State: 90
Test Score: 0.8450, Train Score: 0.8400, Random State: 91
Test Score: 0.8700, Train Score: 0.8400, Random State: 95
Test Score: 0.8600, Train Score: 0.8400, Random State: 98
Test Score: 0.8700, Train Score: 0.8450, Random State: 99
Test Score: 0.8500, Train Score: 0.8400, Random State: 100
Test Score: 0.8700, Train Score: 0.8450, Random State: 104
Test Score: 0.8550, Train Score: 0.8350, Random State: 105
Test Score: 0.8650, Train Score: 0.8300, Random State: 106
Test Score: 0.8600, Train Score: 0.8300, Random State: 109
Test Score: 0.8700, Train Score: 0.8150, Random State: 112
Test Score: 0.8550, Train Score: 0.8250, Random State: 115
Test Score: 0.8600, Train Score: 0.8400, Random State: 116
Test Score: 0.8500, Train Score: 0.8400, Random State: 119
Test Score: 0.8800, Train Score: 0.8300, Random State: 120
Test Score: 0.8500, Train Score: 0.8350, Random State: 123
Test Score: 0.8650, Train Score: 0.8500, Random State: 125
Test Score: 0.8550, Train Score: 0.8450, Random State: 127
Test Score: 0.8650, Train Score: 0.8500, Random State: 128
Test Score: 0.8700, Train Score: 0.8500, Random State: 130
Test Score: 0.8650, Train Score: 0.8250, Random State: 133
Test Score: 0.8550, Train Score: 0.8400, Random State: 134
Test Score: 0.8700, Train Score: 0.8500, Random State: 136
Test Score: 0.8500, Train Score: 0.8250, Random State: 141
Test Score: 0.8450, Train Score: 0.8300, Random State: 143
Test Score: 0.8350, Train Score: 0.8300, Random State: 146
Test Score: 0.8500, Train Score: 0.8450, Random State: 147
Test Score: 0.8600, Train Score: 0.8300, Random State: 148
Test Score: 0.8800, Train Score: 0.8250, Random State: 150
Test Score: 0.8950, Train Score: 0.8350, Random State: 152
Test Score: 0.8600, Train Score: 0.8550, Random State: 154
Test Score: 0.8600, Train Score: 0.8200, Random State: 155
Test Score: 0.8700, Train Score: 0.8600, Random State: 156
Test Score: 0.8600, Train Score: 0.8500, Random State: 159
Test Score: 0.8650, Train Score: 0.8450, Random State: 162
Test Score: 0.8500, Train Score: 0.8000, Random State: 163
Test Score: 0.8700, Train Score: 0.8350, Random State: 164
Test Score: 0.8450, Train Score: 0.8400, Random State: 173
Test Score: 0.8550, Train Score: 0.8450, Random State: 174
Test Score: 0.8600, Train Score: 0.8300, Random State: 178
Test Score: 0.8600, Train Score: 0.8400, Random State: 179
Test Score: 0.8600, Train Score: 0.8250, Random State: 180
Test Score: 0.8750, Train Score: 0.8400, Random State: 184
Test Score: 0.8450, Train Score: 0.8400, Random State: 185
Test Score: 0.8600, Train Score: 0.8500, Random State: 186
Test Score: 0.8750, Train Score: 0.8250, Random State: 187
Test Score: 0.8400, Train Score: 0.8350, Random State: 189
Test Score: 0.8600, Train Score: 0.8400, Random State: 192
Test Score: 0.8450, Train Score: 0.8300, Random State: 194
Test Score: 0.8350, Train Score: 0.8100, Random State: 196
Test Score: 0.8500, Train Score: 0.8350, Random State: 200
Test Score: 0.8800, Train Score: 0.8050, Random State: 202
Test Score: 0.8850, Train Score: 0.8350, Random State: 203
Test Score: 0.8650, Train Score: 0.8450, Random State: 206
Test Score: 0.8600, Train Score: 0.8200, Random State: 209
Test Score: 0.8550, Train Score: 0.8300, Random State: 211
Test Score: 0.8550, Train Score: 0.8350, Random State: 212
Test Score: 0.8900, Train Score: 0.8300, Random State: 213
Test Score: 0.8650, Train Score: 0.8500, Random State: 214
Test Score: 0.8650, Train Score: 0.8300, Random State: 217
Test Score: 0.8950, Train Score: 0.8200, Random State: 220
Test Score: 0.8750, Train Score: 0.8050, Random State: 223
Test Score: 0.8550, Train Score: 0.8450, Random State: 228
Test Score: 0.8600, Train Score: 0.8500, Random State: 229
Test Score: 0.8650, Train Score: 0.8400, Random State: 232
Test Score: 0.8450, Train Score: 0.8400, Random State: 235
Test Score: 0.8550, Train Score: 0.8250, Random State: 242
Test Score: 0.8400, Train Score: 0.8350, Random State: 243
Test Score: 0.8700, Train Score: 0.8350, Random State: 245
Test Score: 0.8750, Train Score: 0.8450, Random State: 247
Test Score: 0.8950, Train Score: 0.8200, Random State: 252
Test Score: 0.8550, Train Score: 0.8500, Random State: 254
```

```
Test Score: 0.8750, Train Score: 0.8250, Random State: 256
Test Score: 0.8600, Train Score: 0.8350, Random State: 260
Test Score: 0.8700, Train Score: 0.8150, Random State: 266
Test Score: 0.8550, Train Score: 0.8350, Random State: 268
Test Score: 0.8500, Train Score: 0.8300, Random State: 269
Test Score: 0.8850, Train Score: 0.8200, Random State: 270
Test Score: 0.8700, Train Score: 0.8400, Random State: 277
Test Score: 0.8750, Train Score: 0.8150, Random State: 285
Test Score: 0.8700, Train Score: 0.8350, Random State: 287
Test Score: 0.8550, Train Score: 0.8350, Random State: 289
Test Score: 0.8700, Train Score: 0.8150, Random State: 290
Test Score: 0.8300, Train Score: 0.8200, Random State: 296
Test Score: 0.8550, Train Score: 0.8250, Random State: 297
Test Score: 0.8550, Train Score: 0.8500, Random State: 302
Test Score: 0.8550, Train Score: 0.8500, Random State: 303
Test Score: 0.8650, Train Score: 0.8450, Random State: 304
Test Score: 0.8850, Train Score: 0.8350, Random State: 306
Test Score: 0.8900, Train Score: 0.8400, Random State: 308
Test Score: 0.8700, Train Score: 0.8350, Random State: 314
Test Score: 0.8750, Train Score: 0.8450, Random State: 317
Test Score: 0.8550, Train Score: 0.8500, Random State: 318
Test Score: 0.8600, Train Score: 0.8300, Random State: 319
Test Score: 0.8600, Train Score: 0.8300, Random State: 321
Test Score: 0.8550, Train Score: 0.8500, Random State: 326
Test Score: 0.8350, Train Score: 0.8250, Random State: 331
Test Score: 0.8400, Train Score: 0.8250, Random State: 334
Test Score: 0.8650, Train Score: 0.8350, Random State: 336
Test Score: 0.8650, Train Score: 0.8100, Random State: 337
Test Score: 0.8750, Train Score: 0.8100, Random State: 339
Test Score: 0.8650, Train Score: 0.8600, Random State: 344
Test Score: 0.8750, Train Score: 0.8550, Random State: 347
Test Score: 0.8700, Train Score: 0.8500, Random State: 351
Test Score: 0.8650, Train Score: 0.8200, Random State: 354
Test Score: 0.8700, Train Score: 0.8200, Random State: 358
Test Score: 0.8650, Train Score: 0.8550, Random State: 361
Test Score: 0.8700, Train Score: 0.8400, Random State: 362
Test Score: 0.8900, Train Score: 0.8300, Random State: 363
Test Score: 0.8650, Train Score: 0.8600, Random State: 364
Test Score: 0.8600, Train Score: 0.8350, Random State: 366
Test Score: 0.8750, Train Score: 0.8250, Random State: 369
Test Score: 0.8650, Train Score: 0.8450, Random State: 370
Test Score: 0.8700, Train Score: 0.8550, Random State: 371
Test Score: 0.8650, Train Score: 0.8150, Random State: 376
Test Score: 0.8850, Train Score: 0.8300, Random State: 378
Test Score: 0.8650, Train Score: 0.8500, Random State: 379
Test Score: 0.8650, Train Score: 0.8250, Random State: 383
Test Score: 0.8650, Train Score: 0.8400, Random State: 386
Test Score: 0.8650, Train Score: 0.8550, Random State: 390
Test Score: 0.8750, Train Score: 0.8350, Random State: 393
Test Score: 0.8550, Train Score: 0.8300, Random State: 394
Test Score: 0.8900, Train Score: 0.8400, Random State: 399
```

In [15]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

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

# Initialize the Logistic Regression model
finalModel = LogisticRegression()

# Train the model on the training data
finalModel.fit(x_train, y_train)

# Predict the labels for the test data
y_pred = finalModel.predict(x_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Print a detailed classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Print the confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.8875

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.96      0.92        52
           1       0.91      0.75      0.82        28

    accuracy                           0.89        80
   macro avg       0.90      0.86      0.87        80
weighted avg       0.89      0.89      0.88        80


Confusion Matrix:
[[50  2]
 [ 7 21]]
```

In [17]:
```
print(finalModel.score(x_train,y_train))
print(finalModel.score(x_test,y_test))
```

```
0.8375
0.8875
```

In [19]:
```
from sklearn.metrics import classification_report
print(classification_report(label,finalModel.predict(features)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.93      0.89       257
           1       0.85      0.70      0.77       143

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

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js