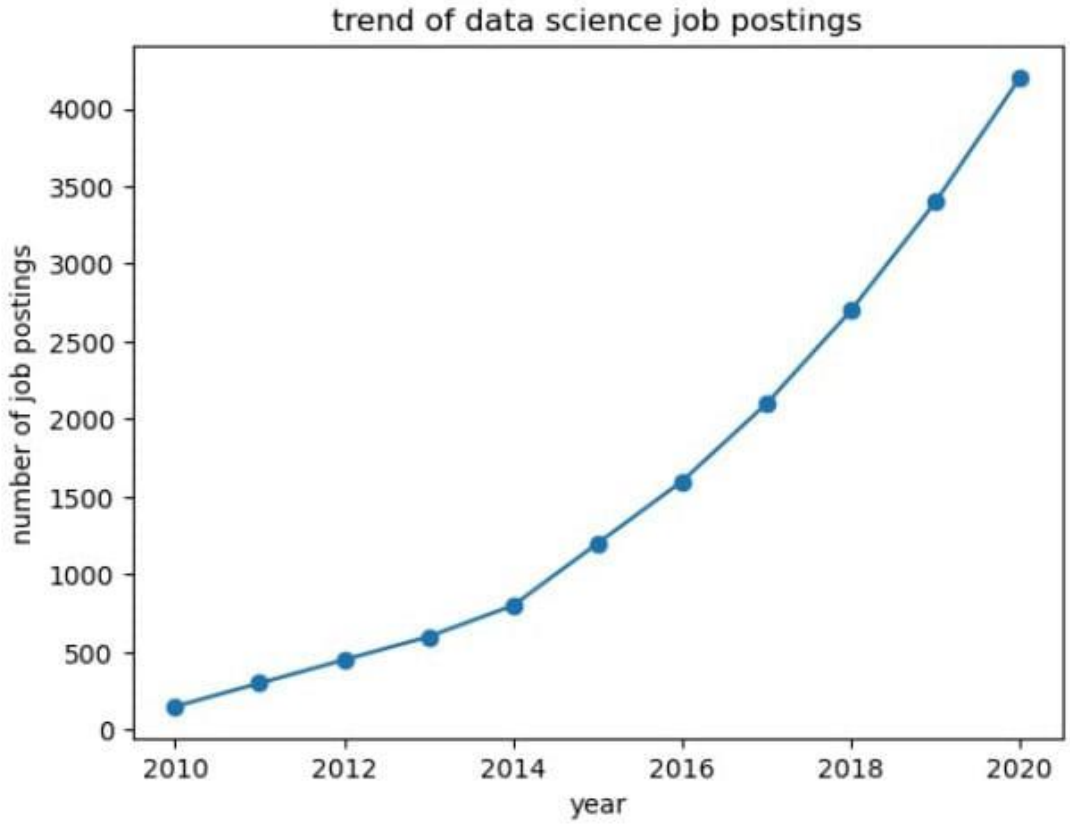
NAME: Elumalai B
ROLL NO: 230701084

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

CODE:

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

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

OUTPUT:



Encrypt and decrypt given sensitive data Use the cryptography library fernet.

NAME: Elumalai B

ROLL NO: 230701084

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

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

```
# Generate key and encrypt data
from cryptography.fernet import Fernet
key = Fernet.generate_key()
f = Fernet(key)
token = f.encrypt(b"Greatest Of All
Time") token
b'...'
f.decrypt(token)
b'Greatest Of All Time'
key = Fernet.generate_key()
cipher_suite = Fernet(key)
plain_text = b"Greatest Of All
Time."
cipher_text = cipher_suite.encrypt(plain_text)
# Decrypt data
decrypted_text = cipher_suite.decrypt(cipher_text)
print("Original Data:", plain_text)
print("Encrypted Data:", cipher_text)
print("Decrypted Data:", decrypted_text)
```

OUTPUT:

```
Original Data: b'Greatest Of All Time.'
Encrypted Data: b'gAAAAABmtETED7_nKo-Y-Ohkm6Z5N7zCPDp7DImNKTmxx6j9oTorBIjYwyARN1SwGbsPp-zDCi9zAI8H53LpDXv5ce8ev7Vns7GpbzHDjYXko
9IWHD0EDcE='
Decrypted Data: b'Greatest Of All Time.'
```

9.Univariate and Bivariate Analysis of Sales Data

ROLL:230701084

Objective: To analyze the distribution and relationships of key variables in a sales dataset.

Dataset: A dataset containing sales data with columns such as Sales, Profit, Region,Product_Category, and Order_Date.

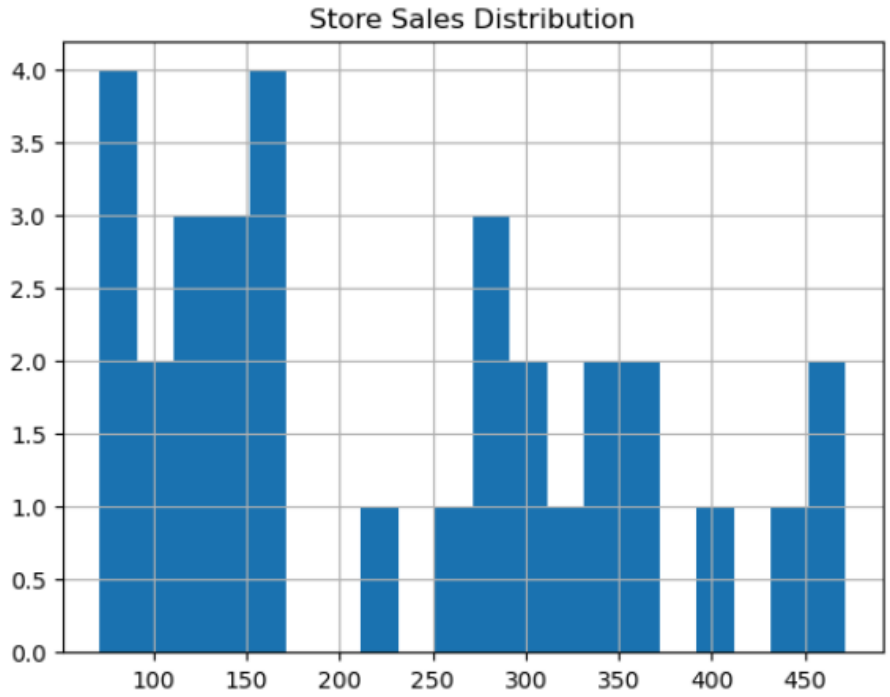
Code:

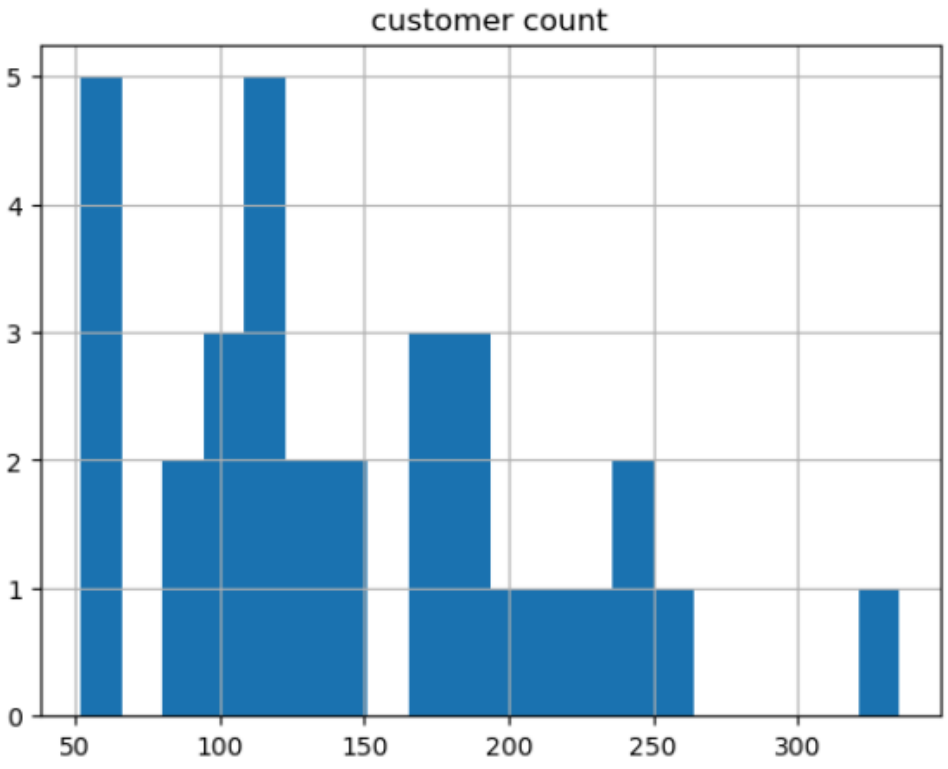
```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt


# Load dataset

data = pd.read_csv('stores.csv')


# Summary Statistics

print(data.describe())


# univariate analysis

data['TotalSales'].hist(bins=20)

plt.title('Store Sales Distribution')

plt.show()


data['Total_Customers'].hist(bins=20)

plt.title('customer count')

plt.show()


# Bivariate Analysis

sns.scatterplot(x='TotalSales', y='Total_Customers', data=data)
```

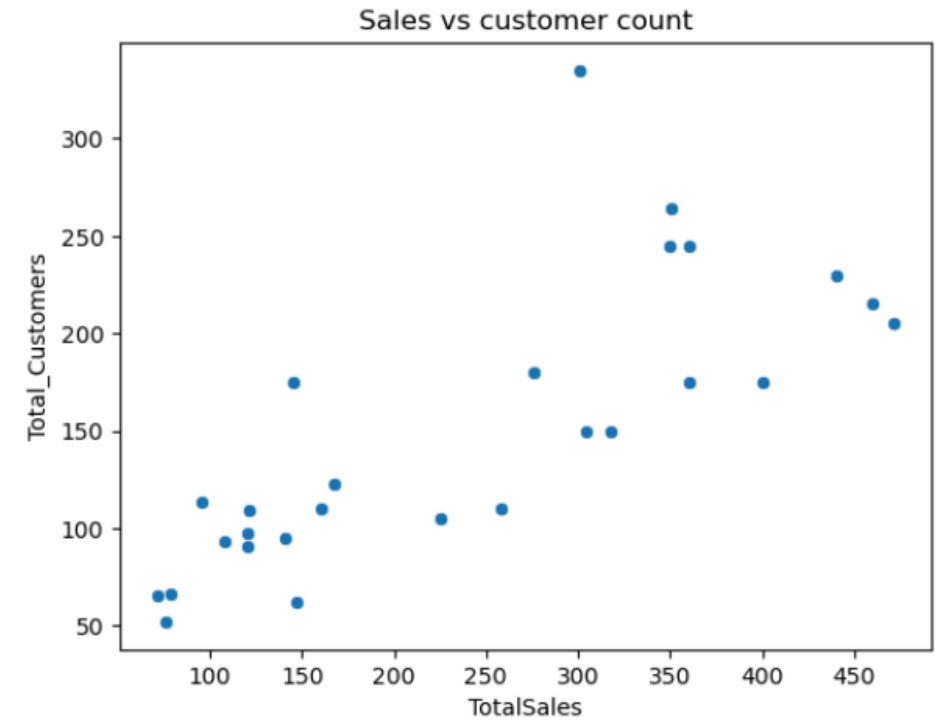
```
plt.title('Sales vs customer count')
plt.show()

sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

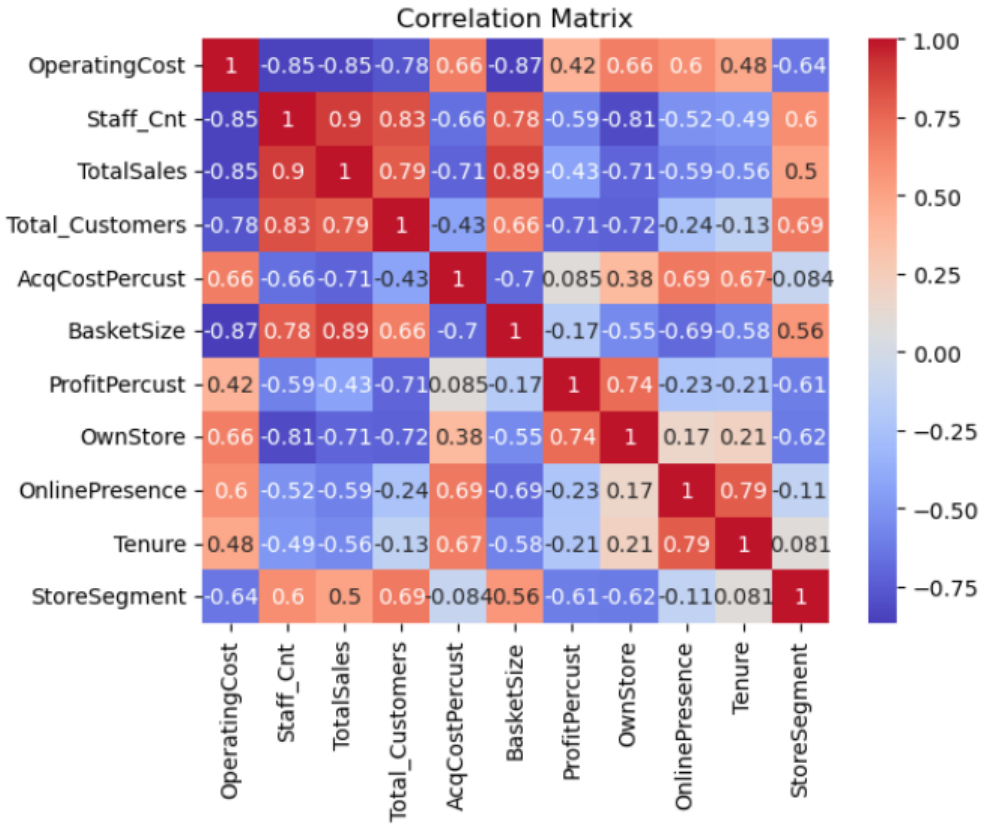
OUTPUT:

```
       OperatingCost  Staff_Cnt  TotalSales  Total_Customers  AcqCostPercust  \
count      32.000000  32.000000   32.000000        32.000000       29.000000
mean       20.090625  61.875000  230.721875       146.687500        3.651034
std         6.026948  17.859216  123.938694        68.562868        0.532664
min        10.400000  40.000000   71.100000        52.000000        2.760000
25%        15.425000  40.000000  120.825000        96.500000        3.150000
50%        19.200000  60.000000  196.300000       123.000000        3.730000
75%        22.800000  80.000000  326.000000       180.000000        3.920000
max        33.900000  80.000000  472.000000       335.000000        4.930000

       BasketSize  ProfitPercust   OwnStore  OnlinePresence     Tenure  \
count   32.000000      32.000000  32.000000       32.000000  32.000000
mean     3.217250      17.848750   0.437500        0.406250   3.687500
std      0.978457       1.786943   0.504016        0.498991   0.737804
min      1.513000      14.500000   0.000000        0.000000   3.000000
25%      2.581250      16.892500   0.000000        0.000000   3.000000
50%      3.325000      17.710000   0.000000        0.000000   4.000000
75%      3.610000      18.900000   1.000000        1.000000   4.000000
max      5.424000      22.900000   1.000000        1.000000   5.000000

       StoreSegment
count     32.000000
mean       2.625000
std        1.211504
min        1.000000
25%        2.000000
50%        2.000000
75%        4.000000
max        4.000000
```


Store Sales Distribution
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
100
150
200
250
300
350
400
450


customer count
5
4
3
2
1
0
50
100
150
200
250
300


Sales vs customer count
Total_Customers
TotalSales
300
250
200
150
100
50
100
150
200
250
300
350
400
450


Correlation Matrix
OperatingCost 1 -0.85 -0.85 -0.78 0.66 -0.87 0.42 0.66 0.6 0.48 -0.64
Staff_Cnt -0.85 1 0.9 0.83 -0.66 0.78 -0.59 -0.81 -0.52 -0.49 0.6
TotalSales -0.85 0.9 1 0.79 -0.71 0.89 -0.43 -0.71 -0.59 -0.56 0.5
Total_Customers -0.78 0.83 0.79 1 -0.43 0.66 -0.71 -0.72 -0.24 -0.13 0.69
AcqCostPercust 0.66 -0.66 -0.71 -0.43 1 -0.7 0.085 0.38 0.69 0.67 -0.084
BasketSize -0.87 0.78 0.89 0.66 -0.7 1 -0.17 -0.55 -0.69 -0.58 0.56
ProfitPercust 0.42 -0.59 -0.43 -0.71 0.085 -0.17 1 0.74 -0.23 -0.21 -0.61
OwnStore 0.66 -0.81 -0.71 -0.72 0.38 -0.55 0.74 1 0.17 0.21 -0.62
OnlinePresence 0.6 -0.52 -0.59 -0.24 0.69 -0.69 -0.23 0.17 1 0.79 -0.11
Tenure 0.48 -0.49 -0.56 -0.13 0.67 -0.58 -0.21 0.21 0.79 1 0.081
StoreSegment -0.64 0.6 0.5 0.69 -0.084 0.56 -0.61 -0.62 -0.11 0.081 1
OperatingCost
Staff_Cnt
TotalSales
Total_Customers
AcqCostPercust
BasketSize
ProfitPercust
OwnStore
OnlinePresence
Tenure
StoreSegment
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

Roll no: 230701084

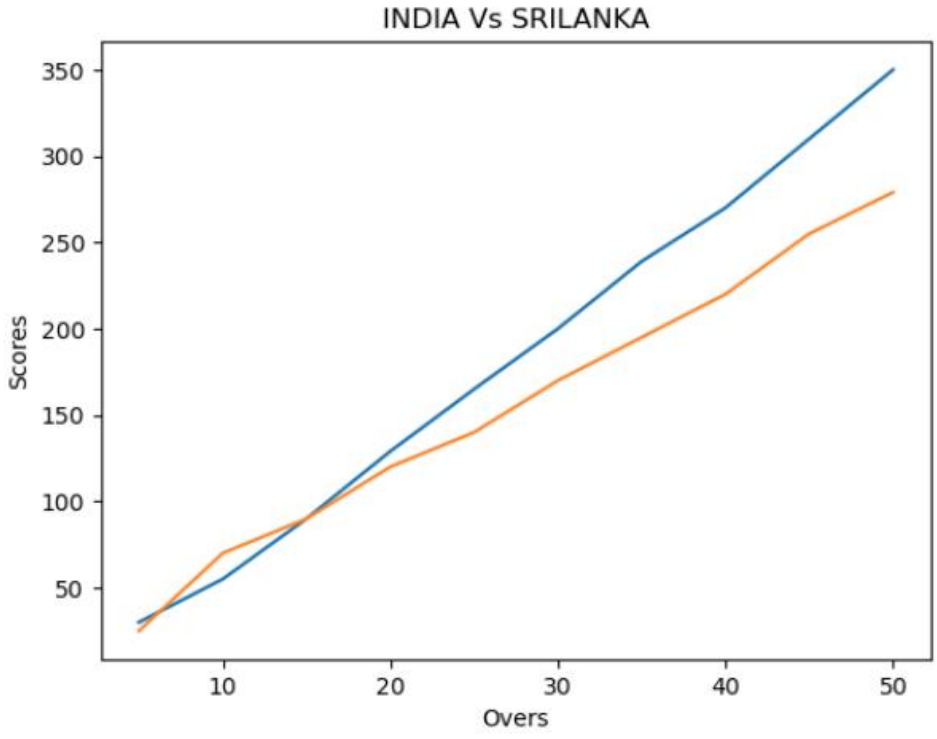
Exp No:3.a Conduct an experiment to show data visualization using line plot.

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

CODE:

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

OUTPUT:



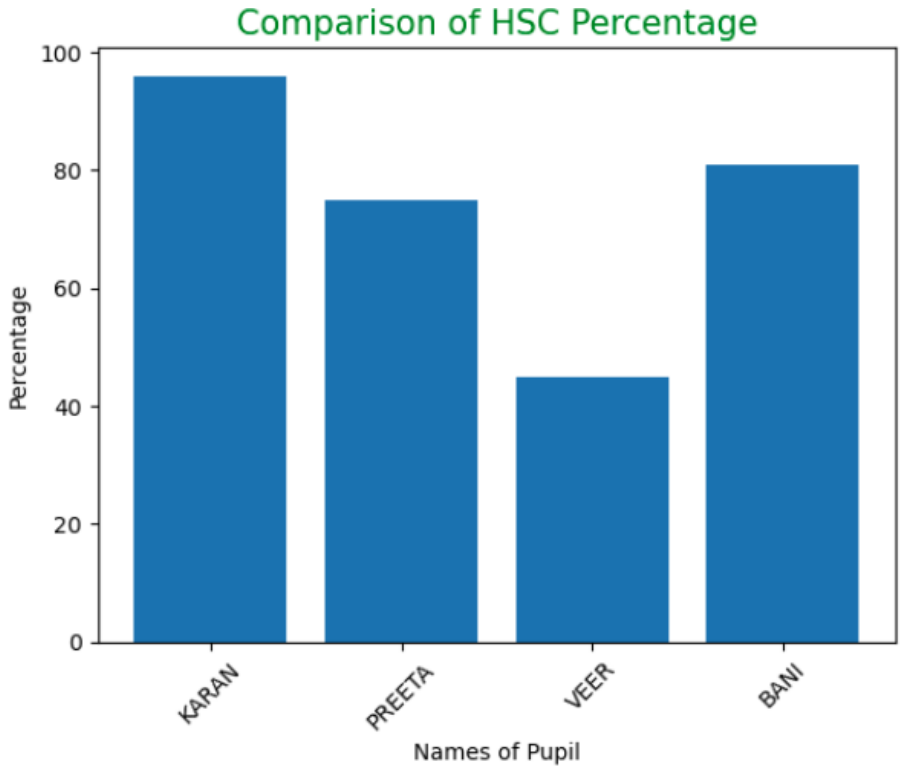
Exp No:3.b Conduct an experiment to show data visualization using bar chart.

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

CODE:

```
import matplotlib.pyplot as hscmark
import numpy as np
Names = ['KARAN', 'PREETA', 'VEER','BANI']
xaxis = np.arange(len(Names))
Percentage_hsc = [96, 75, 45, 81]
hscmark.bar(Names, Percentage_hsc)
hscmark.xticks(xaxis, Names, rotation=45)
hscmark.xlabel('Names of Pupil')
hscmark.ylabel('Percentage')
hscmark.title('Comparison of HSC Percentage', fontsize=15, color='green')
hscmark.show()
```

OUTPUT:



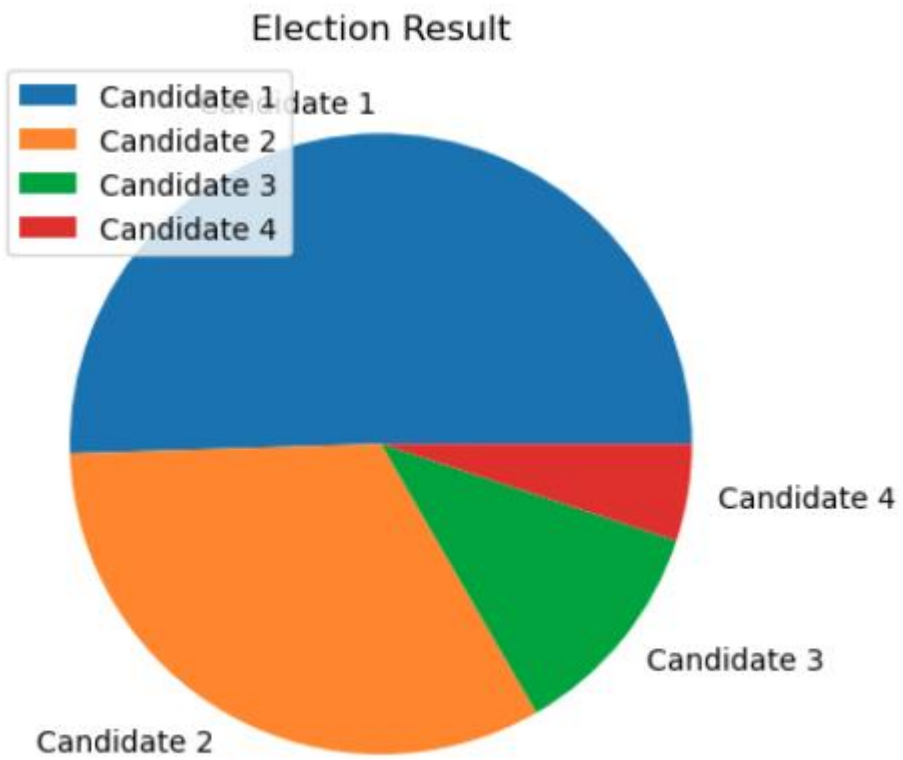
Exp No:3.c Conduct an experiment to show data visualization using pie chart

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

CODE:

```
import numpy as np
import matplotlib.pyplot as election
roles=['Candidate 1','Candidate 2','Candidate 3','Candidate 4']
count=np.array([100,65,23,10])
colours = ['red','blue','green','yellow']
election.pie(count,labels=roles)
election.legend()
election.title("Election Result")
election.show()
```

OUTPUT:



NAME: Elumalai B

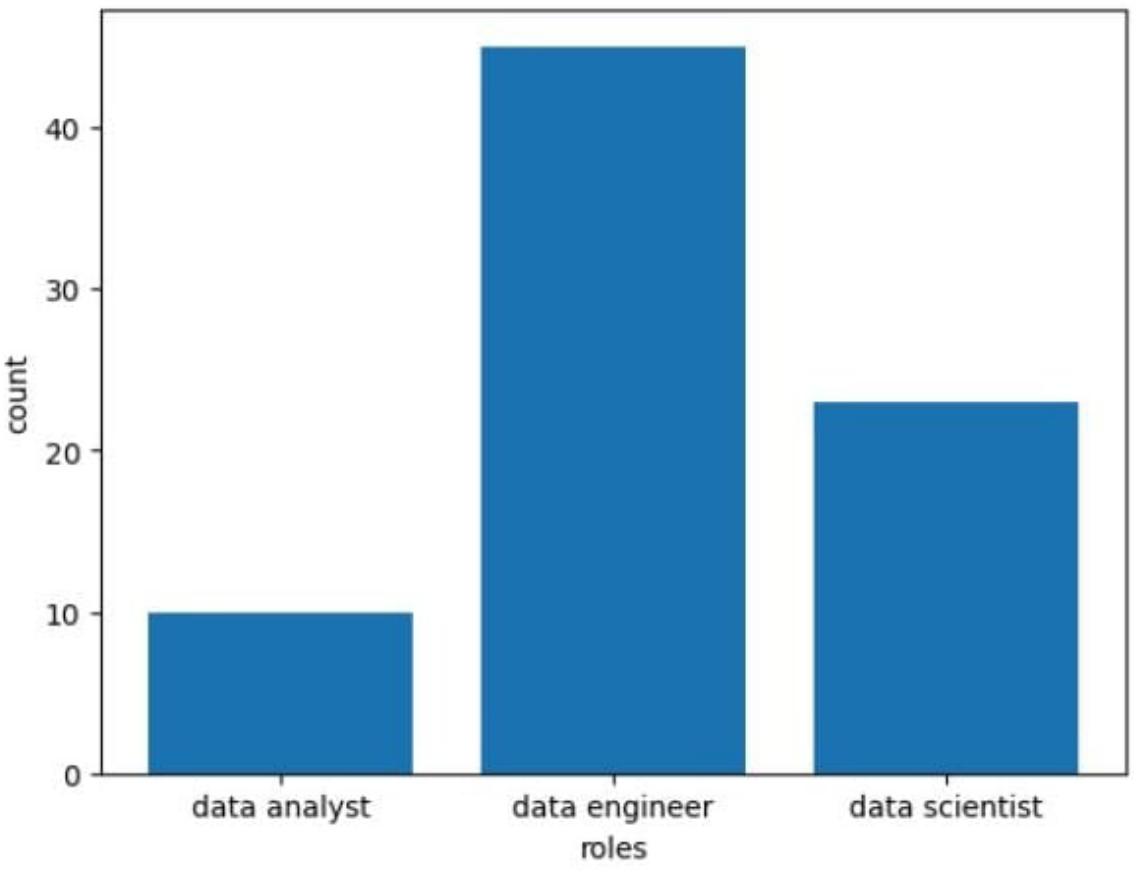
ROLL NO: 230701084

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

CODE:

```
import pandas as pd
import matplotlib.pyplot as plt

roles=['data analyst','data engineer','data scientist']
count=[5,10,45]
plt.bar(roles,count)
plt.xlabel('roles')
plt.ylabel('count')
plt.show()
```

output:


40
30
20
10
0
count
data analyst
data engineer
data scientist
roles

Experiments on Structured, Unstructured and Semi Structured

NAME: ELUMALAI B

ROLL NO: 230701084

Code:

```
import pandas as pd

structured_data=pd.DataFrame({

'ID': [1,2,3], 'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25,30,35]

})

print("Structured data: \n", structured_data)

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

print("Unstructured data: \n", unstructured_data)

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

print("Semi Structed data: \n", semi_structured)
```

output:

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

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

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

4: DATA PREPROCESSING

NAME: Elumalai B

ROLL NO: 230701084

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler


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


print("Original Dataset:")
print(df.head())

df.replace({'Bill': { -1: np.nan, -99999: np.nan, 0: np.nan},
            'NoOfPax': {-1: np.nan, 0: np.nan},
            'EstimatedSalary': {-99999: np.nan, 0: np.nan},
            'Rating(1-5)': { -1: np.nan}},
            inplace=True)


df = df.drop_duplicates()


df['Bill'] = df['Bill'].fillna(df['Bill'].mean())
df['NoOfPax'] = df['NoOfPax'].fillna(df['NoOfPax'].mode()[0])  # Mode for categorical-like column
df['EstimatedSalary'] = df['EstimatedSalary'].fillna(df['EstimatedSalary'].mean())
df['Rating(1-5)'] = df['Rating(1-5)'].fillna(df['Rating(1-5)'].mode()[0])

label_encoder = LabelEncoder()
df['Hotel'] = label_encoder.fit_transform(df['Hotel'])
df['FoodPreference'] = label_encoder.fit_transform(df['FoodPreference'])


df = pd.get_dummies(df, columns=['Age_Group'], drop_first=True)
```

```
scaler = StandardScaler()
df[['Bill', 'EstimatedSalary']] = scaler.fit_transform(df[['Bill', 'EstimatedSalary']])

print("\nPreprocessed Dataset:")
print(df.head())

df.to_csv('Preprocessed_Hotel_Dataset.csv', index=False)
```

```
Original Dataset:
   CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference  Bill  NoOfPax  \
0           1     20-25            4       Ibis            veg  1300        2
1           2     30-35            5  LemonTree        Non-Veg  2000        3
2           3     25-30            6    RedFox            Veg  1322        2
3           4     20-25           -1  LemonTree            Veg  1234        2
4           5       35+            3       Ibis     Vegetarian   989        2

   EstimatedSalary Age_Group.1
0            40000       20-25
1            59000       30-35
2            30000       25-30
3           120000       20-25
4            45000         35+
```

```
    EstimatedSalary Age_Group.1
0            40000       20-25
1            59000       30-35
2            30000       25-30
3           120000       20-25
4            45000         35+

Preprocessed Dataset:
   CustomerID  Rating(1-5)  Hotel  FoodPreference      Bill  NoOfPax  \
0           1          4.0      0               4  0.131957      2.0
1           2          5.0      2               0  0.392446      3.0
2           3          6.0      3               1  0.140143      2.0
3           4          3.0      2               1  0.107396      2.0
4           5          3.0      0               2  0.016225      2.0
```

```
Preprocessed Dataset:
   CustomerID  Rating(1-5)  Hotel  FoodPreference      Bill  NoOfPax  \
0           1          4.0      0               4  0.131957      2.0
1           2          5.0      2               0  0.392446      3.0
2           3          6.0      3               1  0.140143      2.0
3           4          3.0      2               1  0.107396      2.0
4           5          3.0      0               2  0.016225      2.0

   EstimatedSalary Age_Group.1  Age_Group_25-30  Age_Group_30-35  \
0        -0.631656       20-25            False            False
1        -0.420194       30-35            False             True
2        -0.742952       25-30             True            False
3         0.258711       20-25            False            False
4        -0.576008         35+            False            False

   Age_Group_35+
0          False
1          False
2          False
3          False
4           True
```

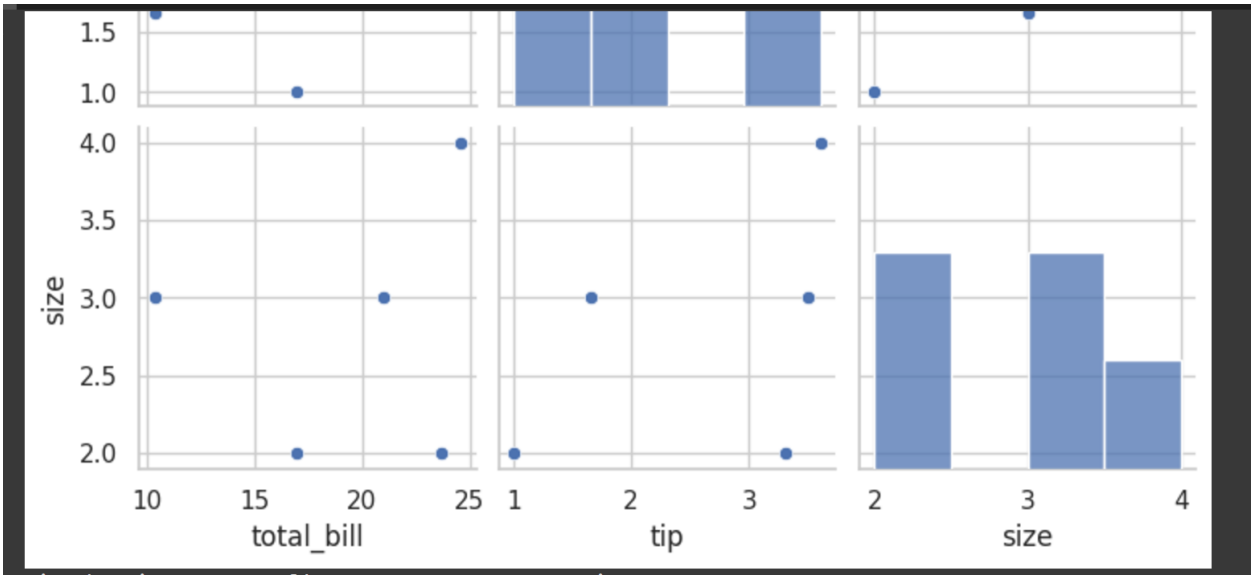
5: EDA quantitative and qualitative plot

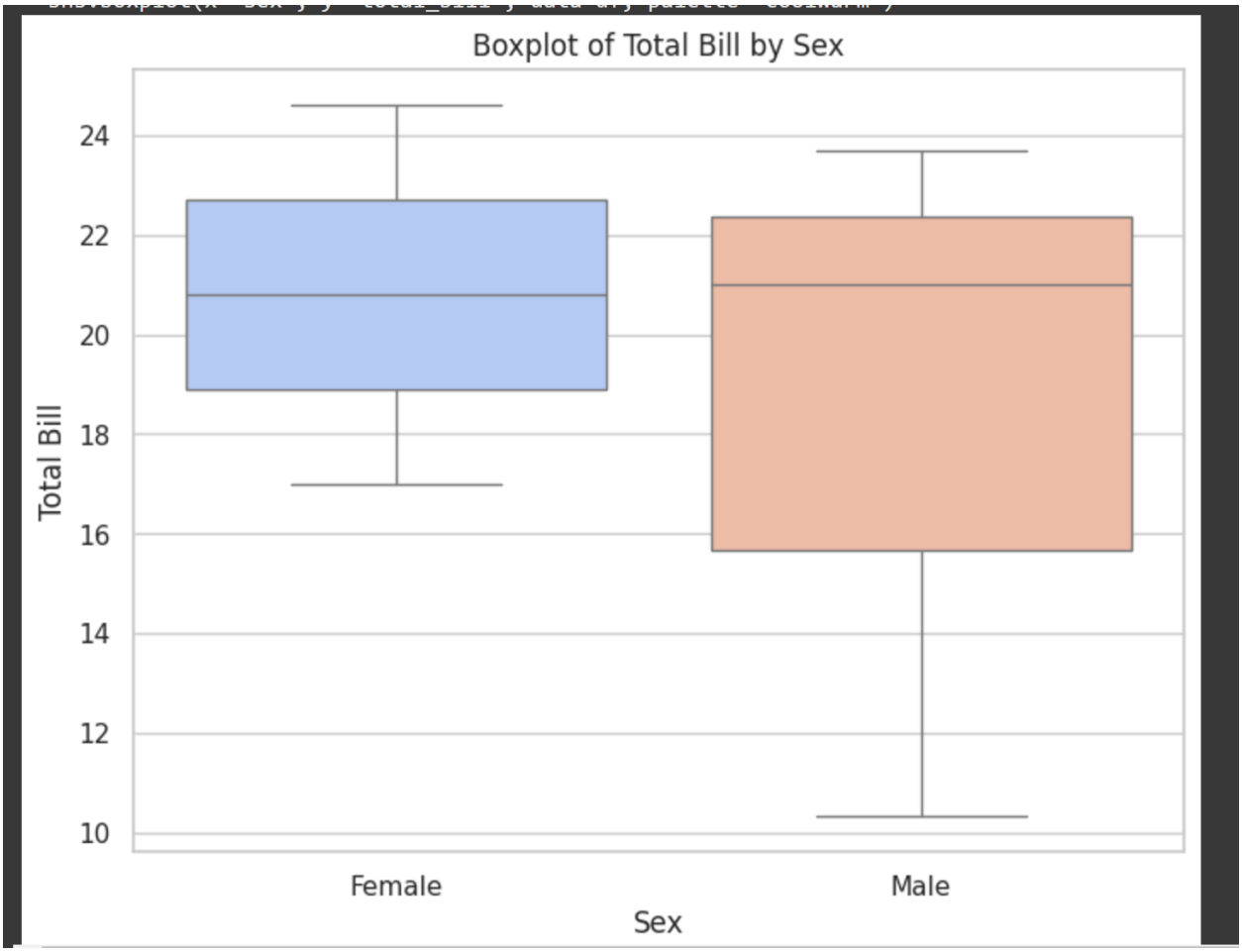
NAME: Elumalai B
ROLL NO :230701084

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


data = {
    'total_bill': [16.99, 10.34, 21.01, 23.68, 24.59],
    'tip': [1.01, 1.66, 3.50, 3.31, 3.61],
    'sex': ['Female', 'Male', 'Male', 'Male', 'Female'],
    'smoker': ['No', 'No', 'No', 'No', 'No'],
    'day': ['Sun', 'Sun', 'Sun', 'Sun', 'Sun'],
    'time': ['Dinner', 'Dinner', 'Dinner', 'Dinner', 'Dinner'],
    'size': [2, 3, 3, 2, 4]
}


df = pd.DataFrame(data)

# Set up Seaborn style for plots
sns.set(style="whitegrid")

# --------------------------
# Quantitative Plots
# --------------------------


plt.figure(figsize=(8, 6))
sns.histplot(df['total_bill'], kde=True, color='blue', bins=10)
plt.title('Distribution of Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Frequency')
plt.show()


plt.figure(figsize=(8, 6))
sns.histplot(df['tip'], kde=True, color='green', bins=10)
plt.title('Distribution of Tip')
```

```
plt.xlabel('Tip')
plt.ylabel('Frequency')
plt.show()


plt.figure(figsize=(8, 6))
sns.boxplot(x=df['total_bill'], color='orange')
plt.title('Boxplot of Total Bill')
plt.xlabel('Total Bill')
plt.show()


plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['total_bill'], y=df['tip'], color='purple')
plt.title('Total Bill vs Tip')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()

# --------------------------
# Qualitative Plots
# --------------------------


plt.figure(figsize=(8, 6))
sns.countplot(x='sex', data=df, palette='Set2')
plt.title('Count of Customers by Sex')
plt.xlabel('Sex')
plt.ylabel('Count')
plt.show()
plt.figure(figsize=(8, 6))
sns.countplot(x='smoker', data=df, palette='Set3')
plt.title('Count of Smokers vs Non-Smokers')
plt.xlabel('Smoker')
plt.ylabel('Count')
plt.show()


plt.figure(figsize=(8, 6))
sns.countplot(x='day', data=df, palette='muted')
plt.title('Count of Customers by Day')
plt.xlabel('Day')
plt.ylabel('Count')
plt.show()
```

```
plt.figure(figsize=(8, 6))
sns.countplot(x='time', data=df, palette='pastel')
plt.title('Count of Customers by Time')
plt.xlabel('Time')
plt.ylabel('Count')
plt.show()

sns.pairplot(df[['total_bill', 'tip', 'size']])
plt.suptitle('Pairplot: Total Bill, Tip, and Size', y=1.02)
plt.show()


plt.figure(figsize=(8, 6))
sns.boxplot(x='sex', y='total_bill', data=df, palette='coolwarm')
plt.title('Boxplot of Total Bill by Sex')
plt.xlabel('Sex')
plt.ylabel('Total Bill')
plt.show()
```




Boxplot of Total Bill by Sex
Total Bill
24
22
20
18
16
14
12
10
Female
Male
Sex

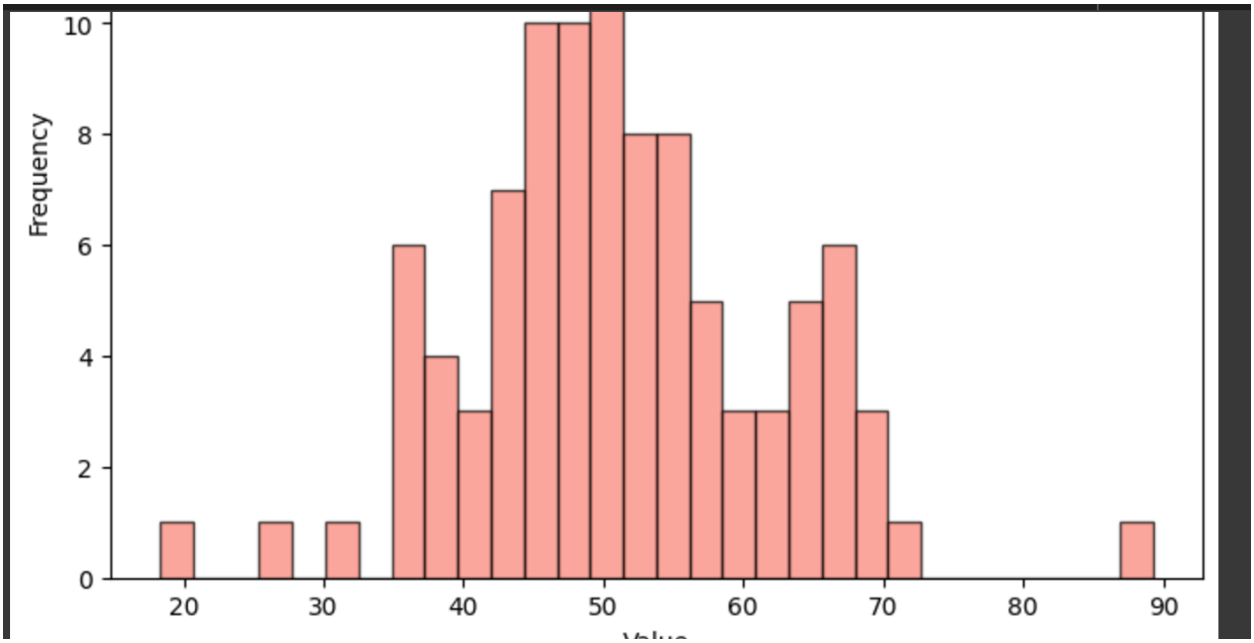
6: RANDOM SAMPLING AND SAMPLING DISTRIBUTION

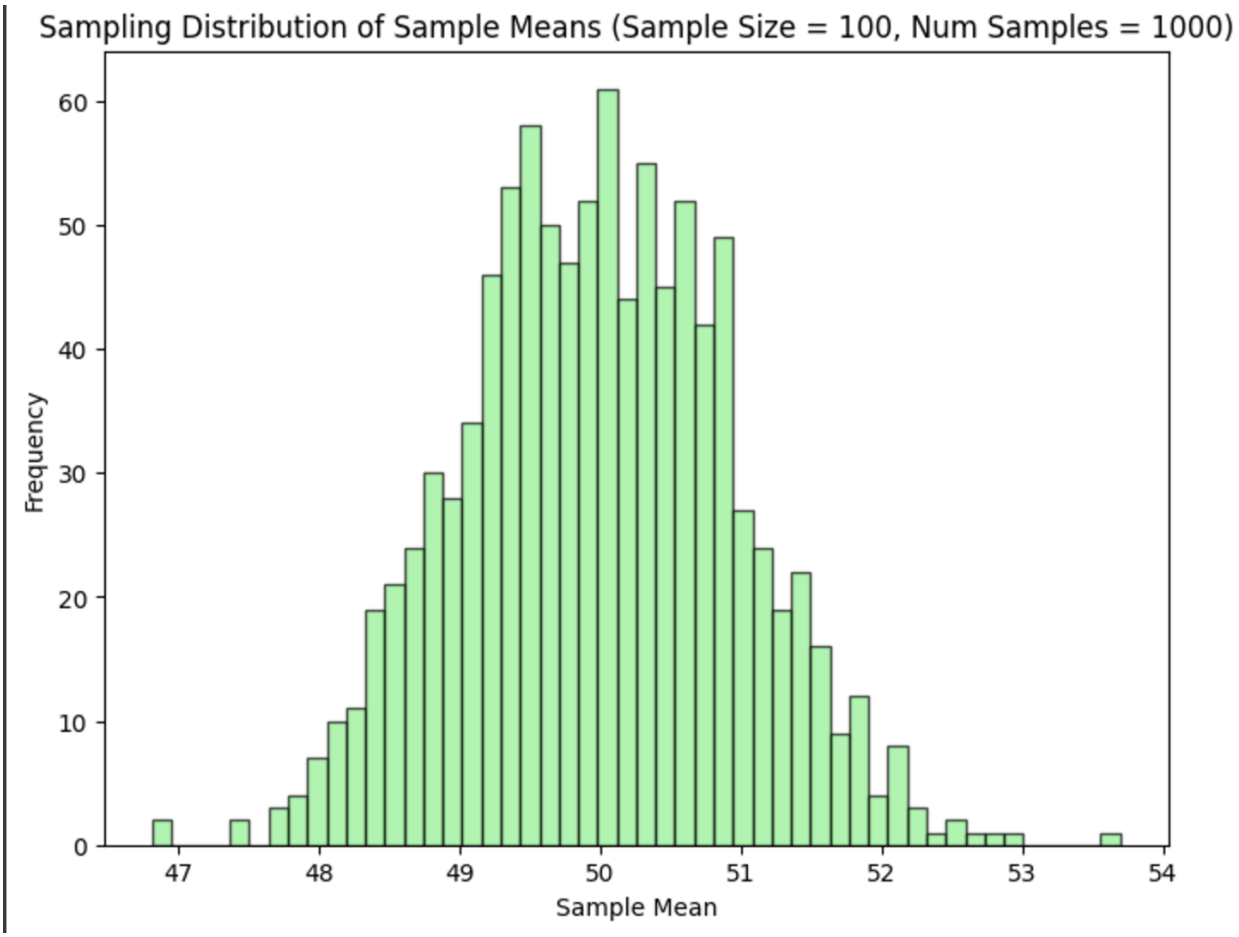
NAME: P Elumalai B
ROLL NO: 230701084

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

np.random.seed(42)

population = np.random.normal(loc=50, scale=10, size=10000)  # Mean=50, SD=10,
Population size=10,000

plt.figure(figsize=(8, 6))
plt.hist(population, bins=50, color='skyblue', edgecolor='black', alpha=0.7)
plt.title('Population Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()


sample_size = 100
random_sample = np.random.choice(population, size=sample_size, replace=False)

plt.figure(figsize=(8, 6))
plt.hist(random_sample, bins=30, color='salmon', edgecolor='black', alpha=0.7)
plt.title(f'Random Sample Distribution (Sample Size = {sample_size})')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

num_samples = 1000  # Number of samples to draw
sample_means = []

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

```
plt.figure(figsize=(8, 6))
plt.hist(sample_means, bins=50, color='lightgreen', edgecolor='black', alpha=0.7)
plt.title(f'Sampling Distribution of Sample Means (Sample Size = {sample_size}, Num
Samples = {num_samples})')
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')
plt.show()
```

```
print(f"Mean of population: {np.mean(population)}")
print(f"Mean of sampling distribution: {np.mean(sample_means)}")
print(f"Standard Deviation of population: {np.std(population)}")
print(f"Standard Deviation of sampling distribution: {np.std(sample_means)}")
```




Sampling Distribution of Sample Means (Sample Size = 100, Num Samples = 1000)
Frequency
Sample Mean
0
10
20
30
40
50
60
47
48
49
50
51
52
53
54

7. Z-TEST

NAME : Elumalai B

CLASS: CSE-B

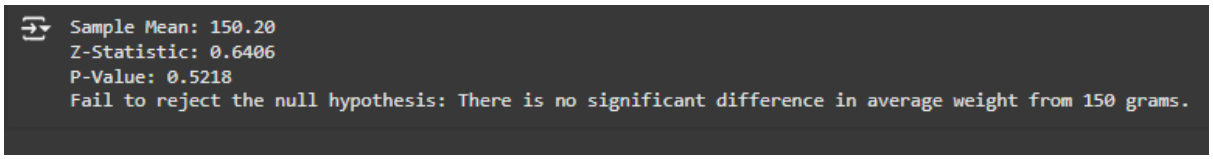
ROLL NO : 230701084

CODE:

```
import numpy as np
import scipy.stats as stats
# Define the sample data (hypothetical weights in grams)
sample_data = np.array([152, 148, 151, 149, 147, 153, 150, 148, 152,
149,151, 150, 149, 152, 151, 148, 150, 152, 149, 150,148, 153, 151,
150, 149, 152, 148, 151, 150, 153])
# Population mean under the null hypothesis
population_mean = 150
# Calculate sample statistics
sample_mean = np.mean(sample_data)
sample_std = np.std(sample_data, ddof=1) # Using sample standard  deviation

# Number of observations
n = len(sample_data)
# Calculate the Z-statistic
z_statistic = (sample_mean - population_mean) / (sample_std /
np.sqrt(n))

# Calculate the p-value

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

```
# Print results
print(f"Sample Mean: {sample_mean:.2f}")
print(f"Z-Statistic: {z_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")
# Decision based on the significance level
alpha = 0.05
if p_value < alpha:
  print("Reject the null hypothesis: The average weight is  significantly different from 150 grams.")
else:
  print("Fail to reject the null hypothesis: There is no significant difference in average weight from 150 grams.")
```

OUTPUT:



8. T-TEST

NAME : Elumalai

CLASS: CSE-B

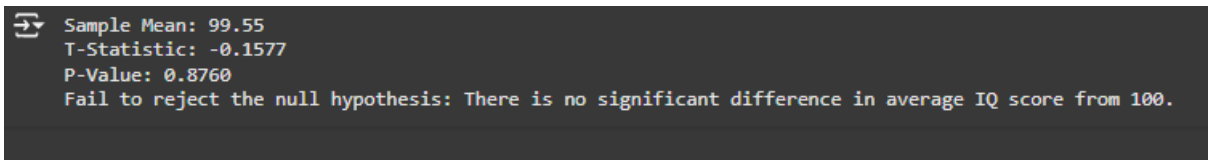
ROLL NO : 230701084

CODE:

```
import numpy as np
import scipy.stats as stats
# Set a random seed for reproducibility
np.random.seed(42)
# Generate hypothetical sample data (IQ scores)
sample_size = 25
sample_data = np.random.normal(loc=102, scale=15,
size=sample_size) # Mean IQ of 102, SD of 15
# Population mean under the null hypothesis
population_mean = 100
# Calculate sample statistics
sample_mean = np.mean(sample_data)
sample_std = np.std(sample_data, ddof=1)
n = len(sample_data)
# Calculate the T-statistic and p-value
t_statistic, p_value = stats.ttest_1samp(sample_data,
```

```
population_mean)
# Print results
print(f"Sample Mean: {sample_mean:.2f}")
print(f"T-Statistic: {t_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")
# Decision based on the significance level
alpha = 0.05
if p_value < alpha:
print("Reject the null hypothesis: The average IQ score is
significantly different from 100.")
else:
print("Fail to reject the null hypothesis: There is no
significant difference in average IQ score from 100.")
```

OUTPUT:



9. ANOVA-TEST

NAME : Elumalai B

CLASS: CSE-B

ROLL NO : 230701084

CODE:

```
import numpy as np
import scipy.stats as stats
# Set a random seed for reproducibility
np.random.seed(42)
# Generate hypothetical growth data for three treatments (A, B, C)
n_plants = 25

# Growth data (in cm) for Treatment A, B, and C
growth_A = np.random.normal(loc=10, scale=2, size=n_plants)
growth_B = np.random.normal(loc=12, scale=3, size=n_plants)
growth_C = np.random.normal(loc=15, scale=2.5, size=n_plants)
# Combine all data into one array
all_data = np.concatenate([growth_A, growth_B, growth_C])
```

```
# Treatment labels for each group
treatment_labels = ['A'] * n_plants + ['B'] * n_plants + ['C'] *
n_plants
# Perform one-way ANOVA
f_statistic, p_value = stats.f_oneway(growth_A, growth_B,
growth_C)
# Print results
print("Treatment A Mean Growth:", np.mean(growth_A))
print("Treatment B Mean Growth:", np.mean(growth_B))
print("Treatment C Mean Growth:", np.mean(growth_C))
print()
print(f"F-Statistic: {f_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")
# Decision based on the significance level
alpha = 0.05
if p_value < alpha:
print("Reject the null hypothesis: There is a significant
difference in mean growth rates among the three
treatments.")
else:
print("Fail to reject the null hypothesis: There is no
significant difference in mean growth rates among the three
```

treatments.")

Additional: Post-hoc analysis (Tukey's HSD) if ANOVA is

significant

if p_value < alpha:

from statsmodels.stats.multicomp import pairwise_tukeyhsd

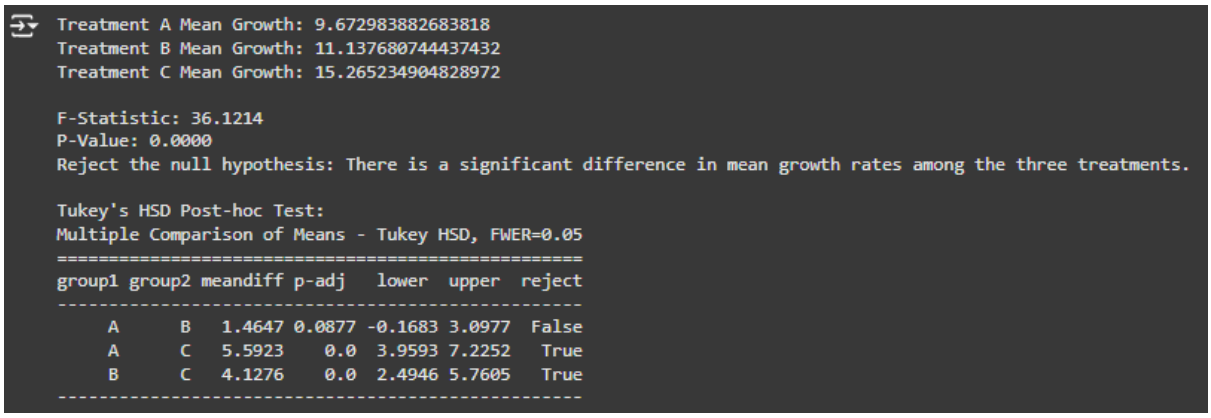
tukey_results = pairwise_tukeyhsd(all_data,
treatment_labels,

alpha=0.05)

print("\nTukey's HSD Post-hoc Test:")

print(tukey_results)

OUTPUT:



10.FEATURE SCALING

NAME : Elumalai B

ROLL NO : 230701084

AIM: To do feature scaling in the given dataset.

```
import numpy as np
```

```
import pandas as pd
```

```
df=pd.read_csv('Data.csv')
```

```
df.head()
```

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

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

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

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

```
from sklearn.impute import SimpleImputer
```

```
age=SimpleImputer(strategy="mean",missing_values=np.nan)
```

```
Salary=SimpleImputer(strategy="mean",missing_values=np.nan)
```

```
age.fit(features[:,[1]])
```

```
<ipython-input-5-6281782b3979>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values

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


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

SimpleImputer

```
SimpleImputer()
```

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

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

```
SimpleImputer
SimpleImputer()
```

SimpleImputer()

```
SimpleImputer()
```

```
SimpleImputer
SimpleImputer()
```

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

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

features

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

from sklearn.preprocessing import OneHotEncoder

oh = OneHotEncoder(sparse_output=False)

Country=oh.fit_transform(features[:,[0]])

Country

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

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

final_set

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

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

sc.fit(final_set)

feat_standard_scaler=sc.transform(final_set)

feat_standard_scaler

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

```
from sklearn.preprocessing import MinMaxScaler

mms=MinMaxScaler(feature_range=(0,1))

mms.fit(final_set)

feat_minmax_scaler=mms.transform(final_set)

feat_minmax_scaler
```

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

11.LINEAR REGRESSION

NAME : Elumalai B

CLASS: CSE-B

ROLL NO : 230701084

```
import numpy as np
```

```
import pandas as pd
```

```
df=pd.read_csv('Salary_data.csv')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     float64
dtypes: float64(2)
memory usage: 608.0 bytes
```

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

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     float64
dtypes: float64(2)
memory usage: 608.0 bytes
```

```
df.describe()
```

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

features=df.iloc[:,[0]].values

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

from sklearn.model_selection import train_test_split

Assuming `features` and `label` are already defined in your code

x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2, random_state=42)

from sklearn.linear_model import LinearRegression

model=LinearRegression()

model.fit(x_train,y_train)

```
LinearRegression
LinearRegression()
```

model.score(x_train,y_train)

```
0.9645401573418146
```

model.score(x_test,y_test)

```
0.9024461774180497
```

model.coef_

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

```
import pickle

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

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

yr_of_exp=float(input("Enter Years of Experience: "))

yr_of_exp_NP=np.array([[yr_of_exp]])

Salary=model.predict(yr_of_exp_NP)
```

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

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

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

12.Logistic Regression

NAME : Elumalai B

ROLL NO : 230701084

```
import numpy as np
```

```
import pandas as pd
```

```
df=pd.read_csv('Social_Network_Ads.csv')
```

```
df
```

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

400 rows × 5 columns

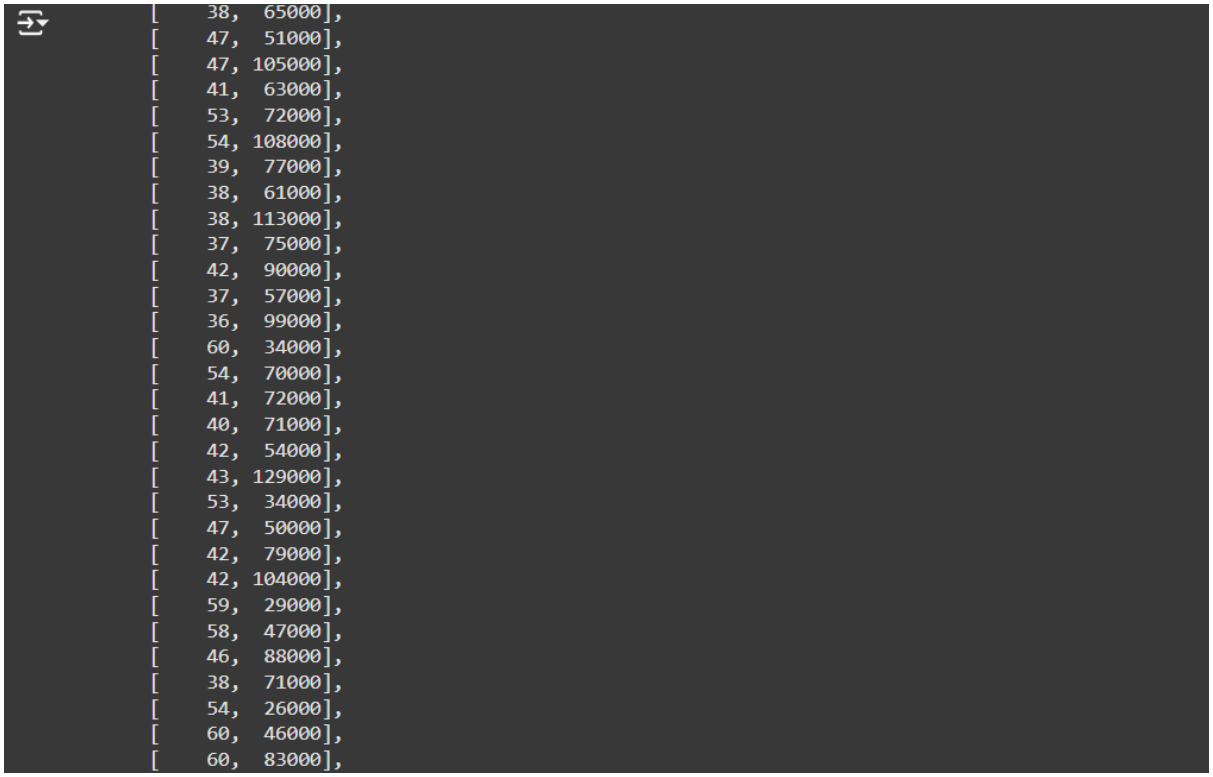
```
df.head()
```

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

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

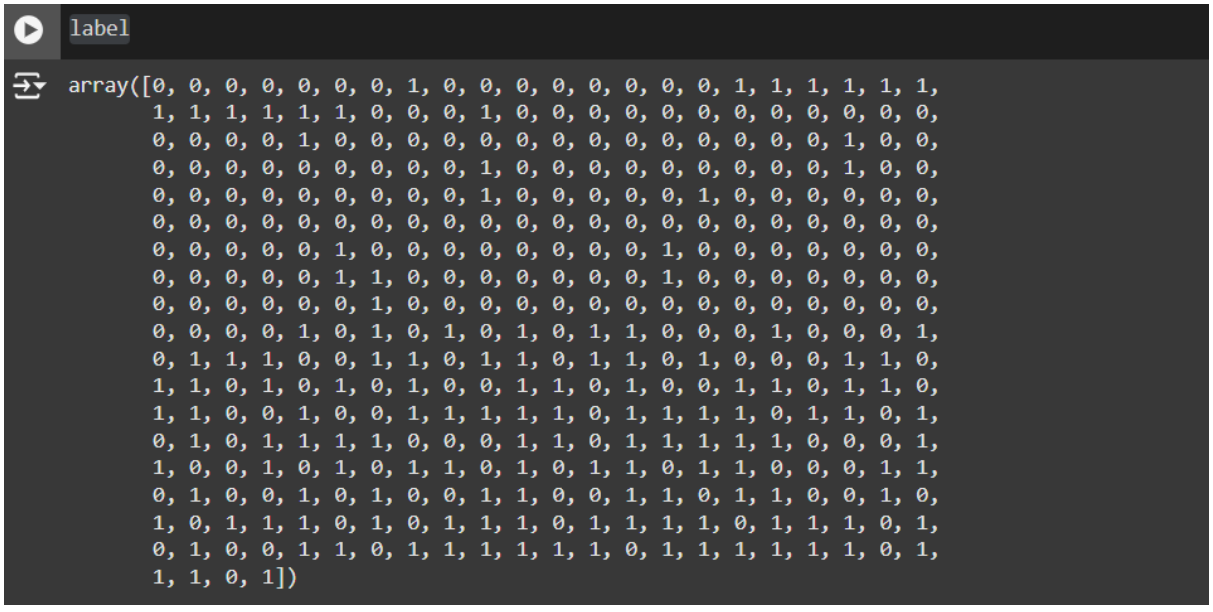
label=df.iloc[:,4].values

features



```
[    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],
```

label



```
label

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

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

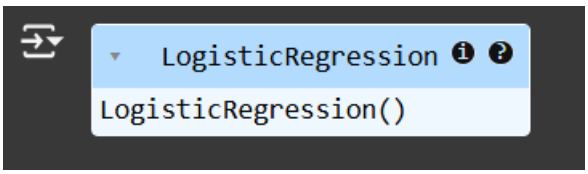
for i in range(1, 401):

```
# 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=i)

# Initialize the Logistic Regression model
model = LogisticRegression()

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

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

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

```
Test 0.8625 Train 0.8375 Random State 268
Test 0.875 Train 0.840625 Random State 275
Test 0.8625 Train 0.85 Random State 276
Test 0.925 Train 0.8375 Random State 277
Test 0.875 Train 0.846875 Random State 282
Test 0.85 Train 0.846875 Random State 283
Test 0.85 Train 0.84375 Random State 285
Test 0.9125 Train 0.834375 Random State 286
Test 0.85 Train 0.840625 Random State 290
Test 0.85 Train 0.840625 Random State 291
Test 0.85 Train 0.846875 Random State 292
Test 0.8625 Train 0.8375 Random State 294
Test 0.8875 Train 0.828125 Random State 297
Test 0.8625 Train 0.834375 Random State 300
Test 0.8625 Train 0.85 Random State 301
Test 0.8875 Train 0.85 Random State 302
Test 0.875 Train 0.846875 Random State 303
Test 0.8625 Train 0.834375 Random State 305
Test 0.9125 Train 0.8375 Random State 306
Test 0.875 Train 0.846875 Random State 308
Test 0.9 Train 0.84375 Random State 311
Test 0.8625 Train 0.834375 Random State 313
Test 0.9125 Train 0.834375 Random State 314
Test 0.875 Train 0.8375 Random State 315
Test 0.9 Train 0.846875 Random State 317
Test 0.9125 Train 0.821875 Random State 319
Test 0.8625 Train 0.85 Random State 321
Test 0.9125 Train 0.828125 Random State 322
Test 0.85 Train 0.846875 Random State 328
Test 0.85 Train 0.8375 Random State 332
Test 0.8875 Train 0.853125 Random State 336
Test 0.85 Train 0.8375 Random State 337
```

✓ 0s completed at 1:57 PM

```
# Assuming features and label are defined earlier in your code

x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2)


finalModel = LogisticRegression()

finalModel.fit(x_train, y_train)
```




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

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

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

0.859375
0.8375
```

from sklearn.metrics import classification_report

print(classification_report(label,finalModel.predict(features)))

```
              precision    recall  f1-score   support

           0       0.86      0.92      0.89       257
           1       0.84      0.73      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
```