**FUNDAMENTALS OF DATA SCIENCE**

**Name : ENIYA.B.A**

**Class : CSE-B**

**Roll no : 230701085**

Description: Use web scraping (e.g., BeautifulSoup) or APIs (e.g., LinkedIn API) to gather data on

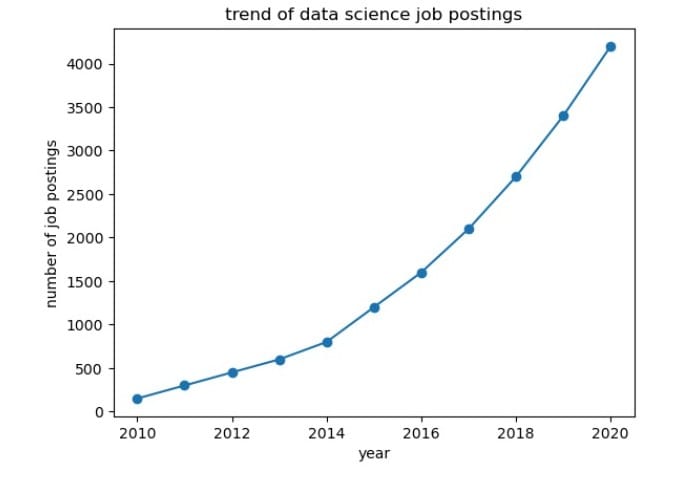
the number of data science job postings each year. Use pandas for data manipulation and

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:



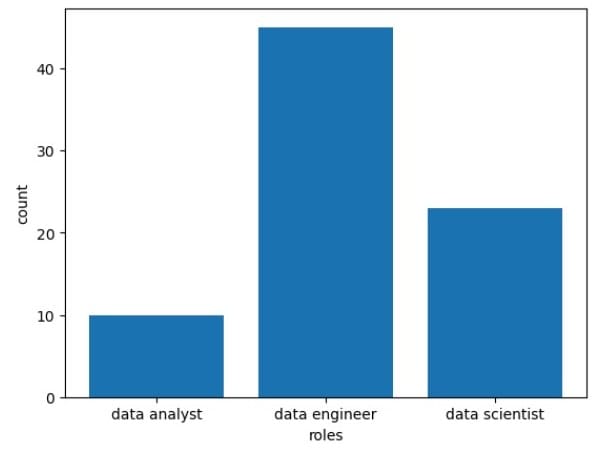
Description: Use a dataset of job postings and categorize them into different roles. Visualize

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:



Experiments on Structured, Unstructured and Semi Structured

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)

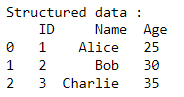
unstructured\_data="This is an example of unstructured data. It can be a piece of text, an image, or a video file."

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:







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

**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:**



**DESCRIPTION:** Bar plot of total sales by product and their weekly salary

**CODE:**

import pandas as pd

import matplotlib.pyplot as plt

df = pd.read\_csv('Sales\_Transactions\_Dataset\_Weekly.csv')

subset\_df = df.head(10)

plt.figure(figsize=(10, 6))

plt.bar(subset\_df['Product\_Code'], subset\_df['W1'],color='Purple')

plt.xlabel('Product\_Code')

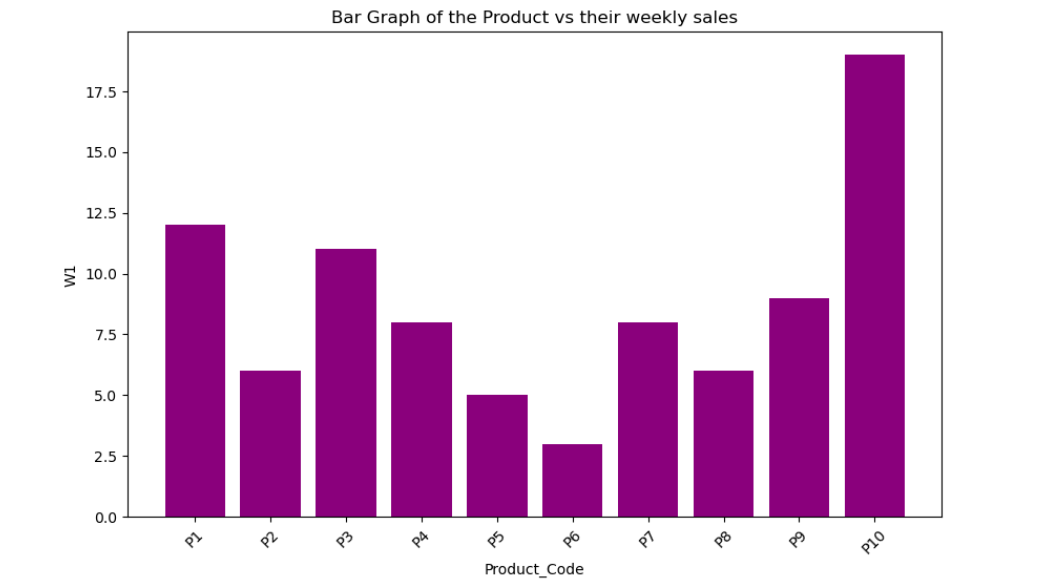
plt.ylabel('W1')

plt.title('Bar Graph of the Product vs their weekly sales')

plt.xticks(rotation=45)

plt.show()

**OUTPUT:**



**Upload and Analyze the data set given in csv format and perform data**

**preprocessing and visualization.**

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

**CODE:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

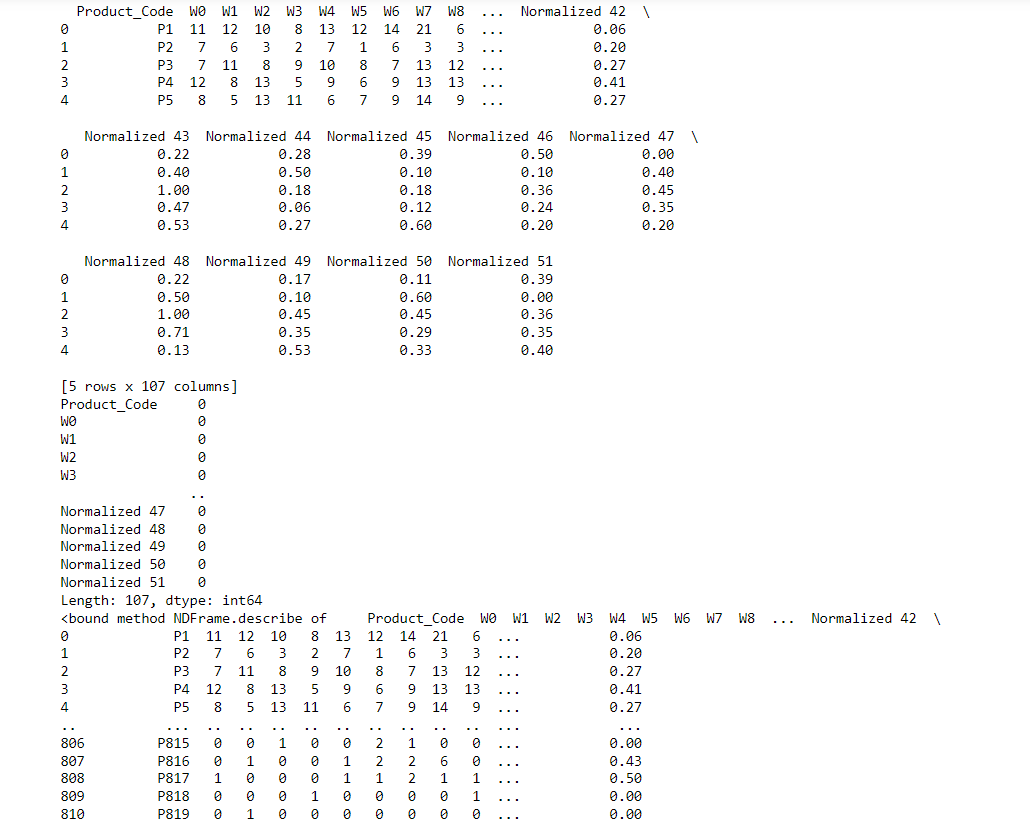
df=pd.read\_csv('Sales\_Transactions\_Dataset\_Weekly.csv')

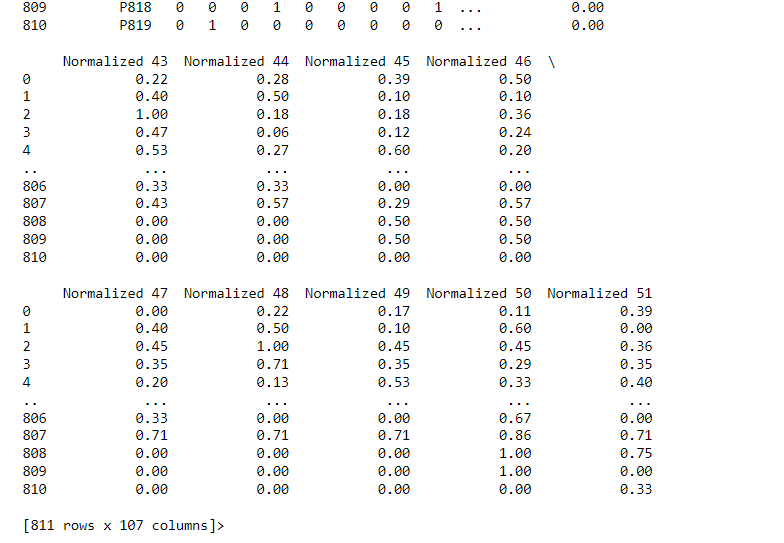
print(df.head())

print(df.isnull().sum())

print(df.describe)

**OUTPUT:**





**9.Univariate and Bivariate Analysis of Sales Data**

**ROLL:230701085**

**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')

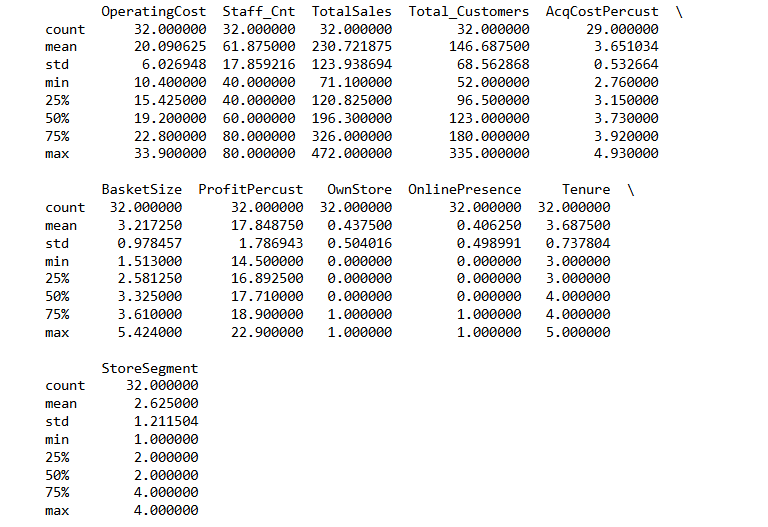
plt.show()

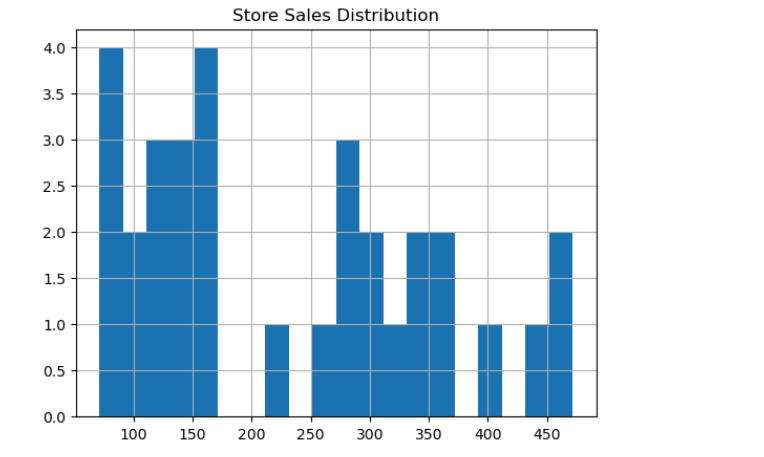
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')

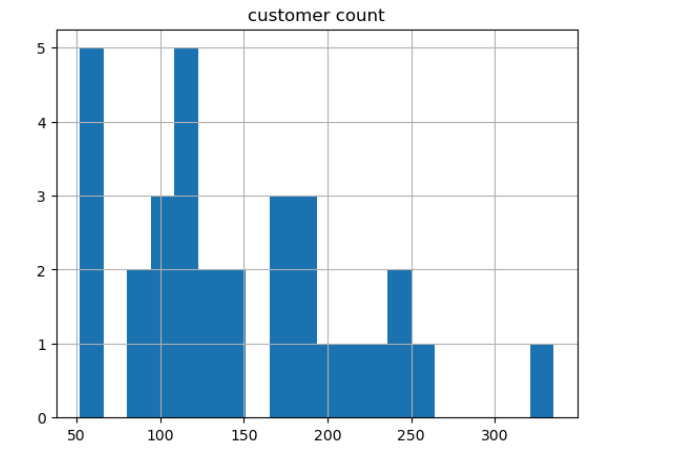
plt.title('Correlation Matrix')

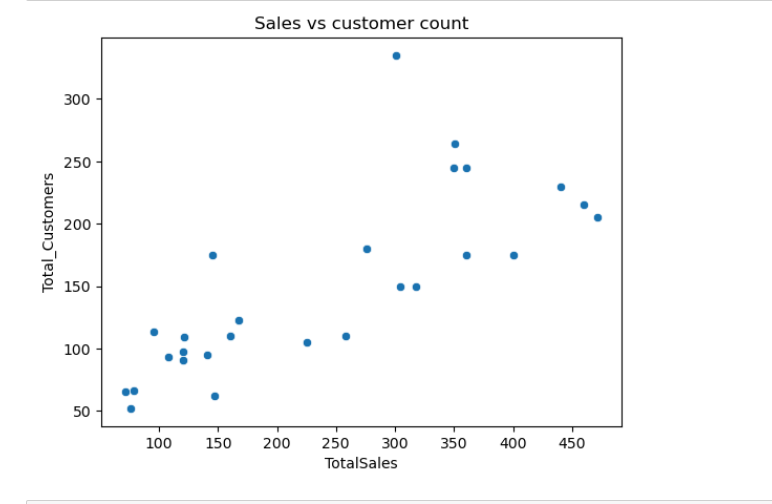
plt.show()

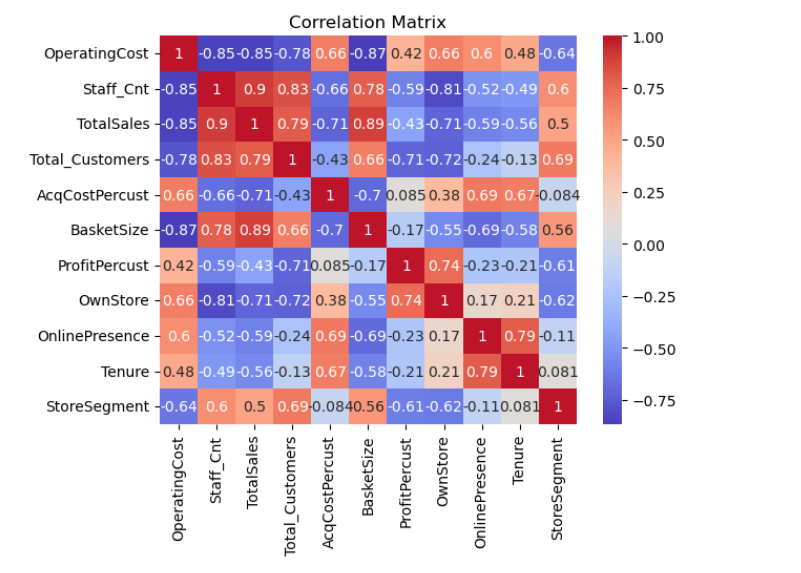
**OUTPUT:**

****

****

****

****

****

**Exp No:5 Data Collection and Initial Exploration**

**NAME:ENIYA.B.A**

**ROLL NO:230701085**

**Objective:** To collect, load, and perform initial exploration of the diabetes dataset.

import pandas as pd

db=pd.read\_csv("diabetes.csv")

print(db.head())

**OUTPUT:**

Pregnancies Glucose BloodPressure SkinThickness Insulin BMI \

0 6 148 72 35 0 33.6

1 1 85 66 29 0 26.6

2 8 183 64 0 0 23.3

3 1 89 66 23 94 28.1

4 0 137 40 35 168 43.1

DiabetesPedigreeFunction Age Outcome

0 0.627 50 1

1 0.351 31 0

2 0.672 32 1

3 0.167 21 0

4 2.288 33 1

print(db.info())

print(db.describe())

**OUTPUT:**

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 9 entries, 0 to 8

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Pregnancies 9 non-null int64

1 Glucose 9 non-null int64

2 BloodPressure 9 non-null int64

3 SkinThickness 9 non-null int64

4 Insulin 9 non-null int64

5 BMI 9 non-null float64

6 DiabetesPedigreeFunction 9 non-null float64

7 Age 9 non-null int64

8 Outcome 9 non-null int64

dtypes: float64(2), int64(7)

memory usage: 776.0 bytes

None

Pregnancies Glucose BloodPressure SkinThickness Insulin \

count 9.000000 9.000000 9.000000 9.000000 9.000000

mean 4.000000 127.555556 55.777778 22.111111 99.222222

std 3.464102 42.497386 23.631429 17.567331 177.146675

min 0.000000 78.000000 0.000000 0.000000 0.000000

25% 1.000000 89.000000 50.000000 0.000000 0.000000

50% 3.000000 116.000000 66.000000 29.000000 0.000000

75% 6.000000 148.000000 70.000000 35.000000 94.000000

max 10.000000 197.000000 74.000000 45.000000 543.000000

BMI DiabetesPedigreeFunction Age Outcome

count 9.000000 9.000000 9.000000 9.000000

mean 30.788889 0.538444 33.888889 0.555556

std 5.997175 0.686137 10.635371 0.527046

min 23.300000 0.134000 21.000000 0.000000

25% 26.600000 0.167000 29.000000 0.000000

50% 30.500000 0.248000 31.000000 1.000000

75% 33.600000 0.627000 33.000000 1.000000

max 43.100000 2.288000 53.000000 1.000000

import matplotlib.pyplot as plt

import seaborn as sns

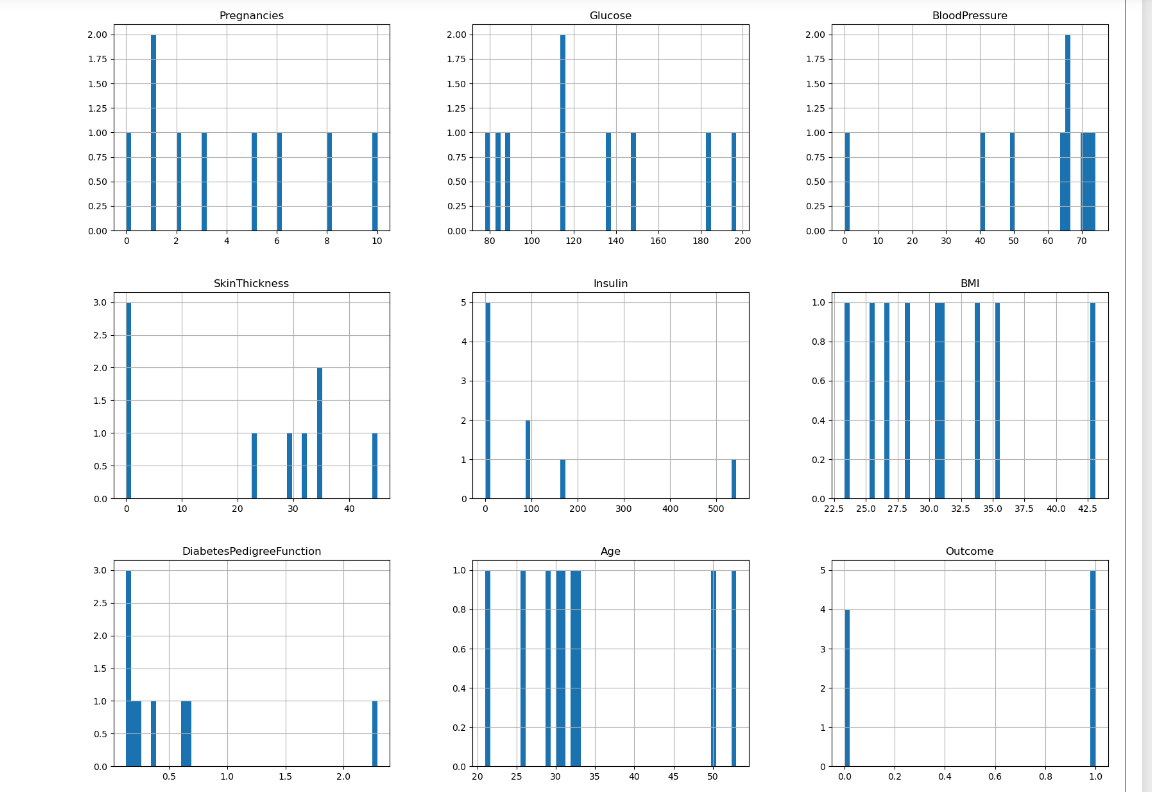
db.hist(bins=50,figsize=(20,15))

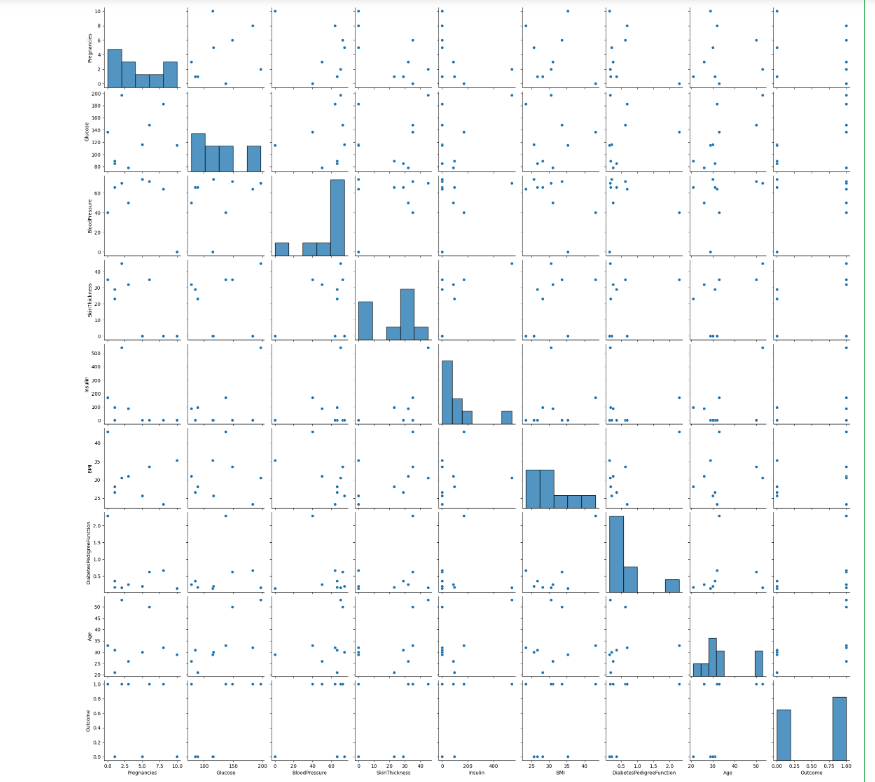
plt.show()

sns.pairplot(db)

plt.show()

**OUTPUT:**





**Data Cleaning**

**ROLL NO: 230701085**

**Objective:** To clean the diabetes dataset by handling missing values, outliers, and inconsistent data.

**Materials Needed:**

 Software Engineer Salaries dataset

 Python with libraries: pandas, numpy, matplotlib

**Procedure and Code:**

1. **Handling Missing Values:**

**CODE:**

import pandas as pd

import matplotlib.pyplot as plt

db=pd.read\_csv('Software Engineer Salaries.csv')

print(db.isnull().sum())

db.fillna(db.mean(),inplace=True)

print(db.isnull().sum())

**OUTPUT:**

Company 2

Company Score 0

Job Title 0

Location 13

Date 0

Salary 106

dtype: int64

**Handling Outliers:**

import numpy as np

import matplotlib.pyplot as plt

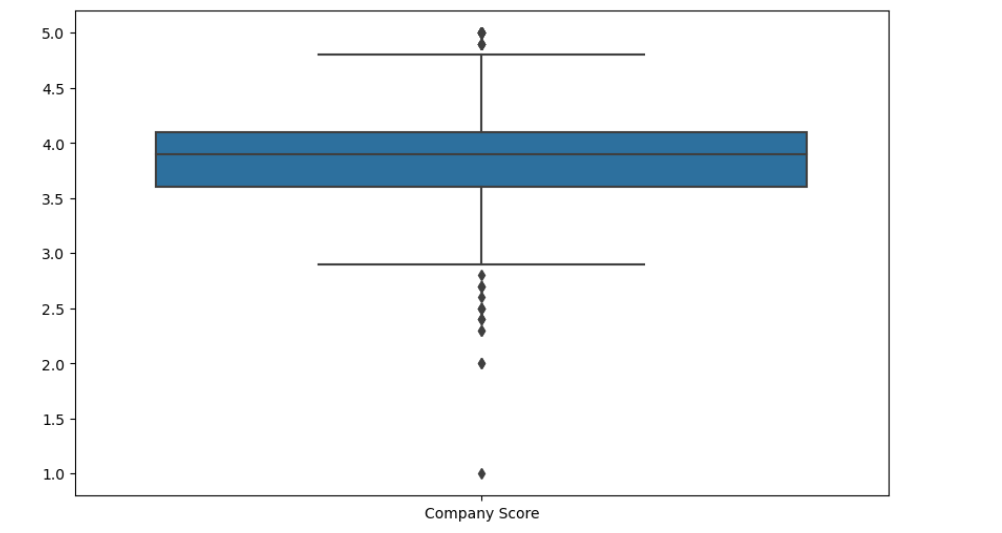
import seaborn as sns

plt.figure(figsize=(10, 6))

sns.boxplot(data=db)

plt.show()

**OUTPUT:**



# 4.DATA PREPROCESSING

## NAME: ENIYA.B.A

CLASS : CSE-B

## ROLL NO: 230701085

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)

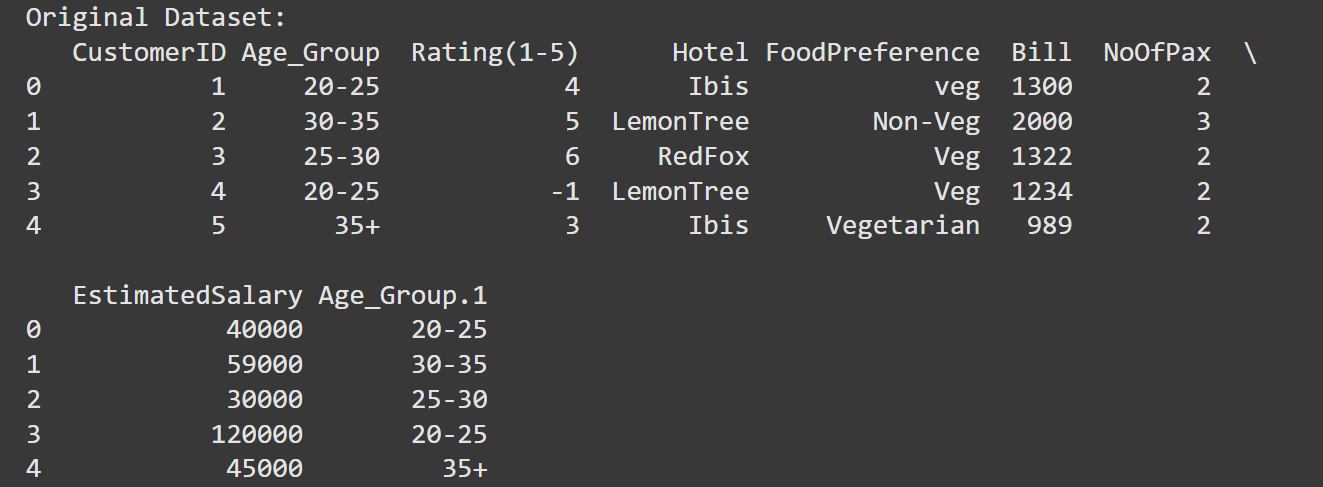
scaler = StandardScaler()

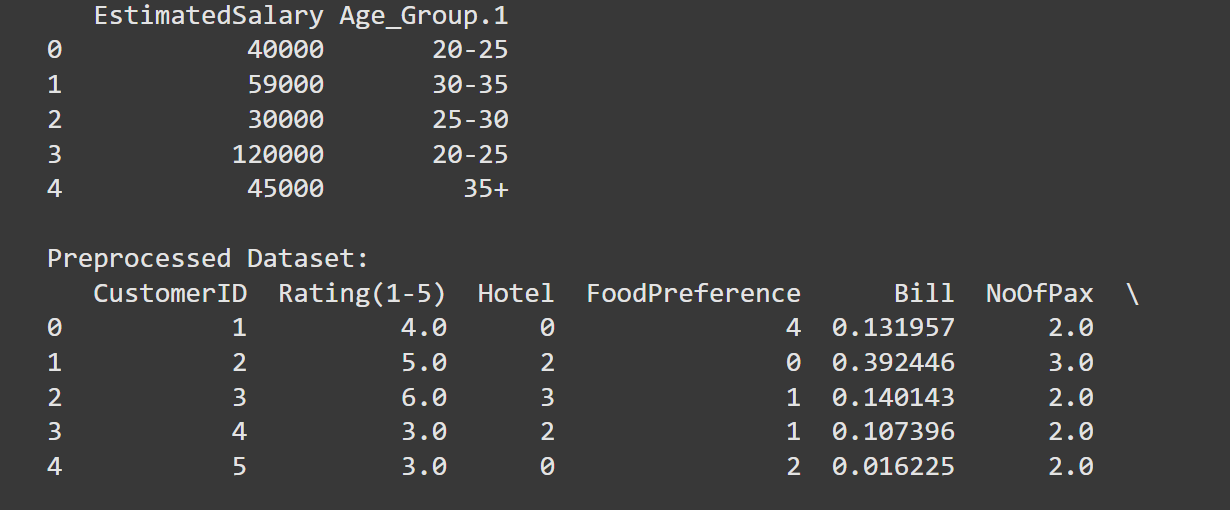
df[['Bill', 'EstimatedSalary']] = scaler.fit\_transform(df[['Bill', 'EstimatedSalary']])

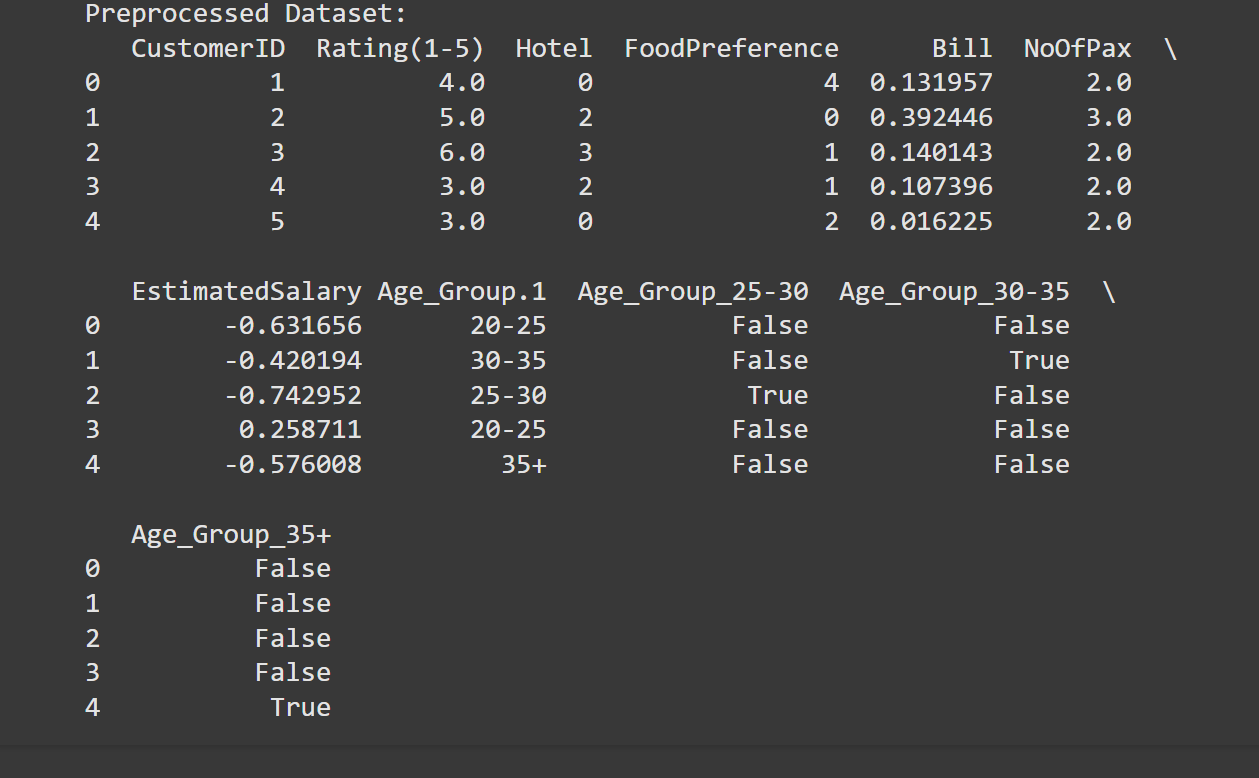
print("\nPreprocessed Dataset:")

print(df.head())

df.to\_csv('Preprocessed\_Hotel\_Dataset.csv', index=False)







# 5: EDA quantitative and qualitative ploT

**NAME: ENIYA.B.A**

**CLASS:CSE-B**

**ROLL NO :230701085**

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))

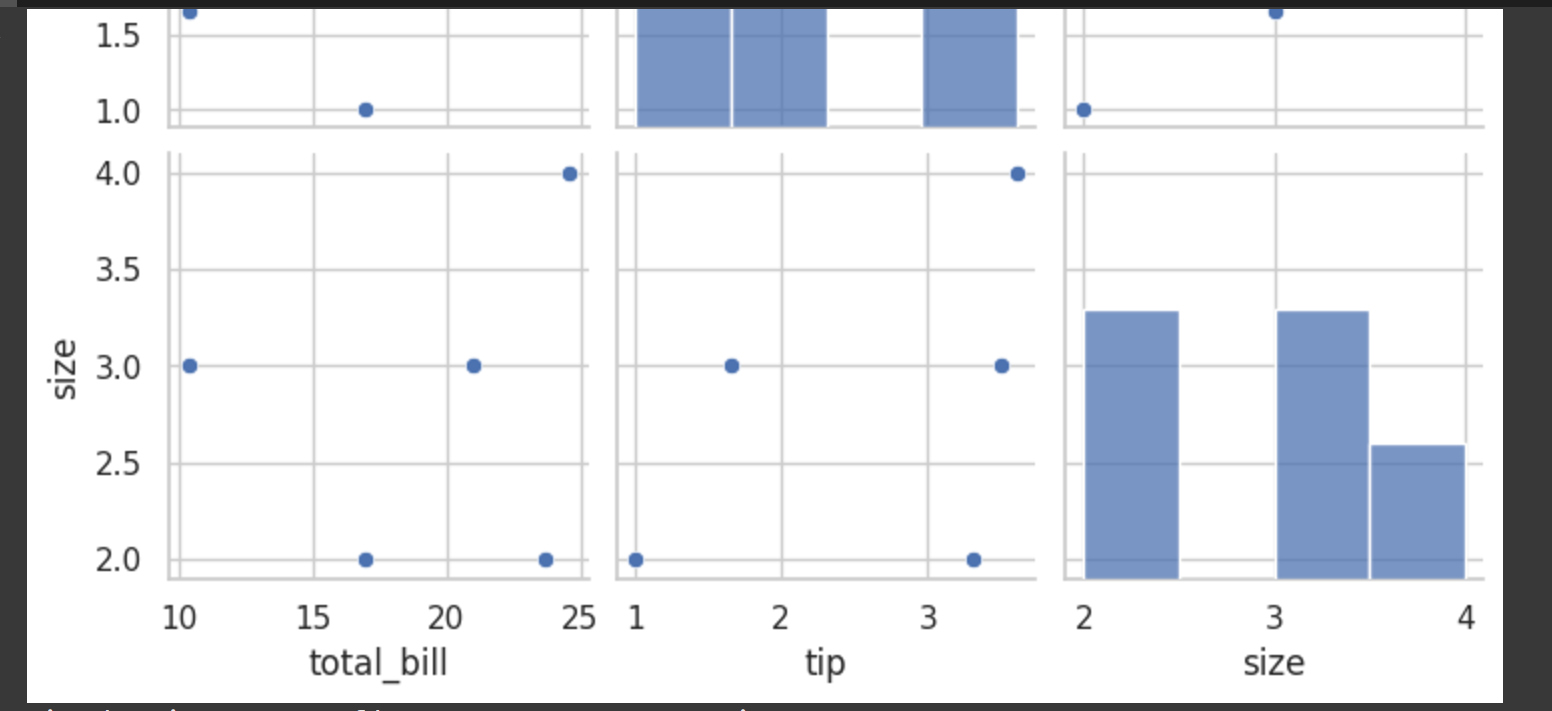
sns.boxplot(x='sex', y='total\_bill', data=df, palette='coolwarm')

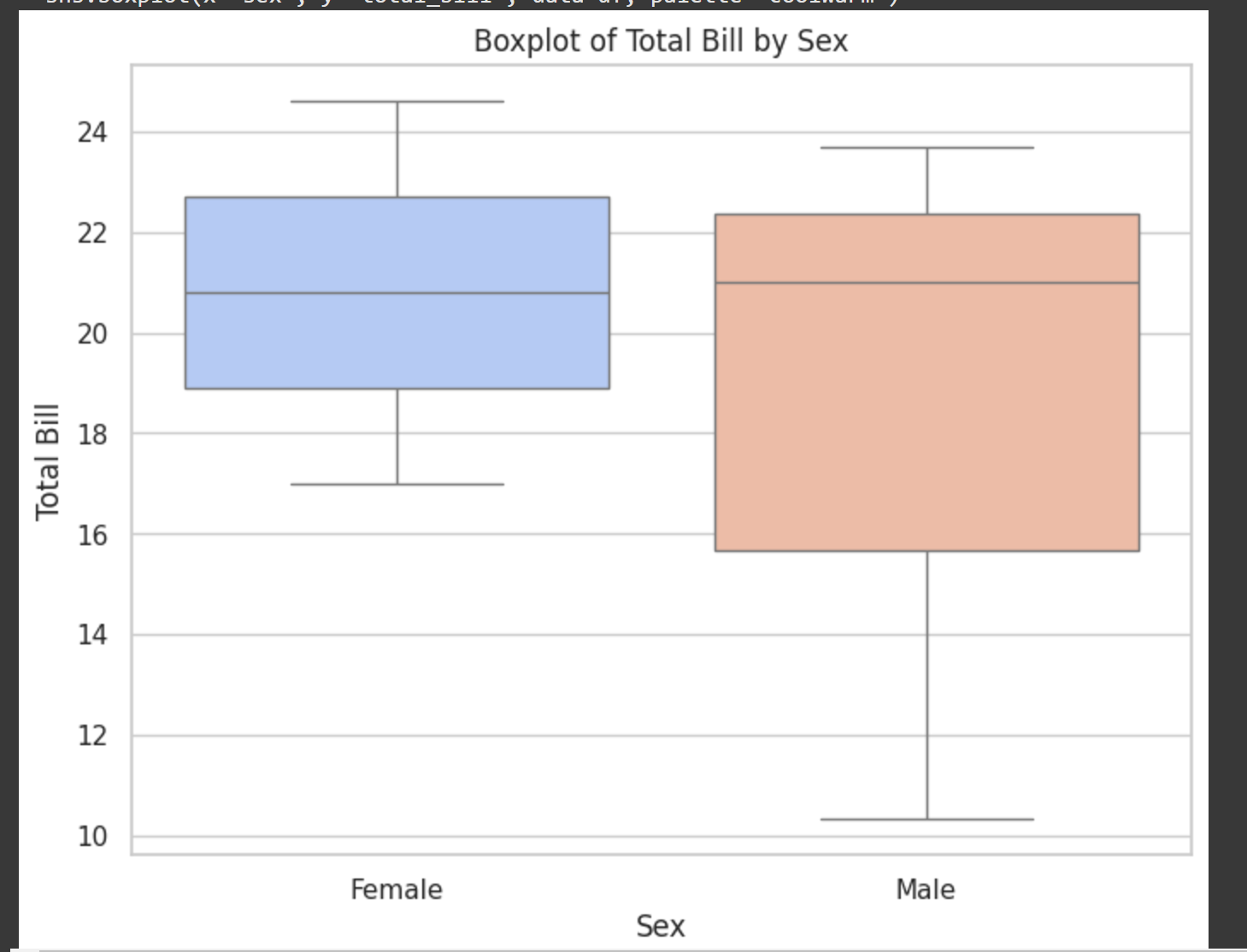
plt.title('Boxplot of Total Bill by Sex')

plt.xlabel('Sex')

plt.ylabel('Total Bill')

plt.show()





# 6: RANDOM SAMPLING AND SAMPLING DISTRIBUTION

**NAME: ENIYA.B.A**

**CLASS : CSE-B**

**ROLL NO: 230701085**

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')

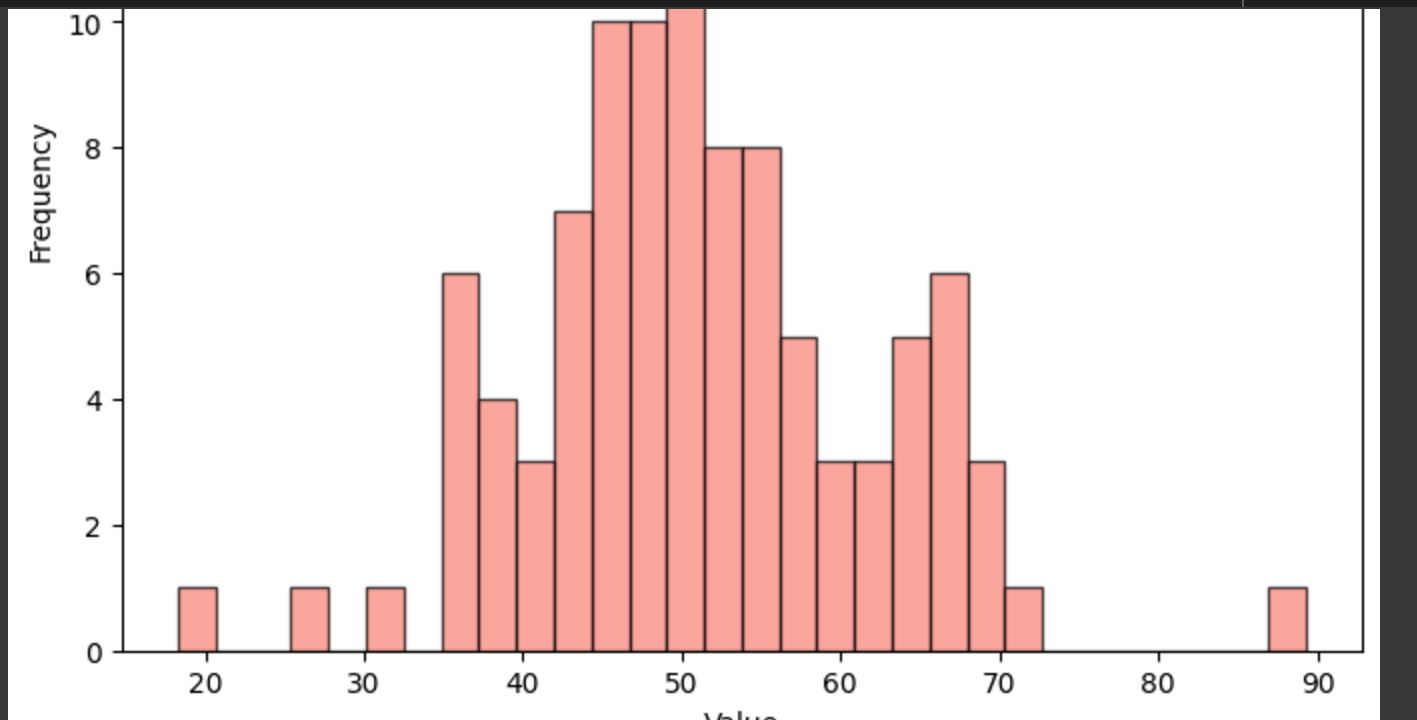
plt.show()

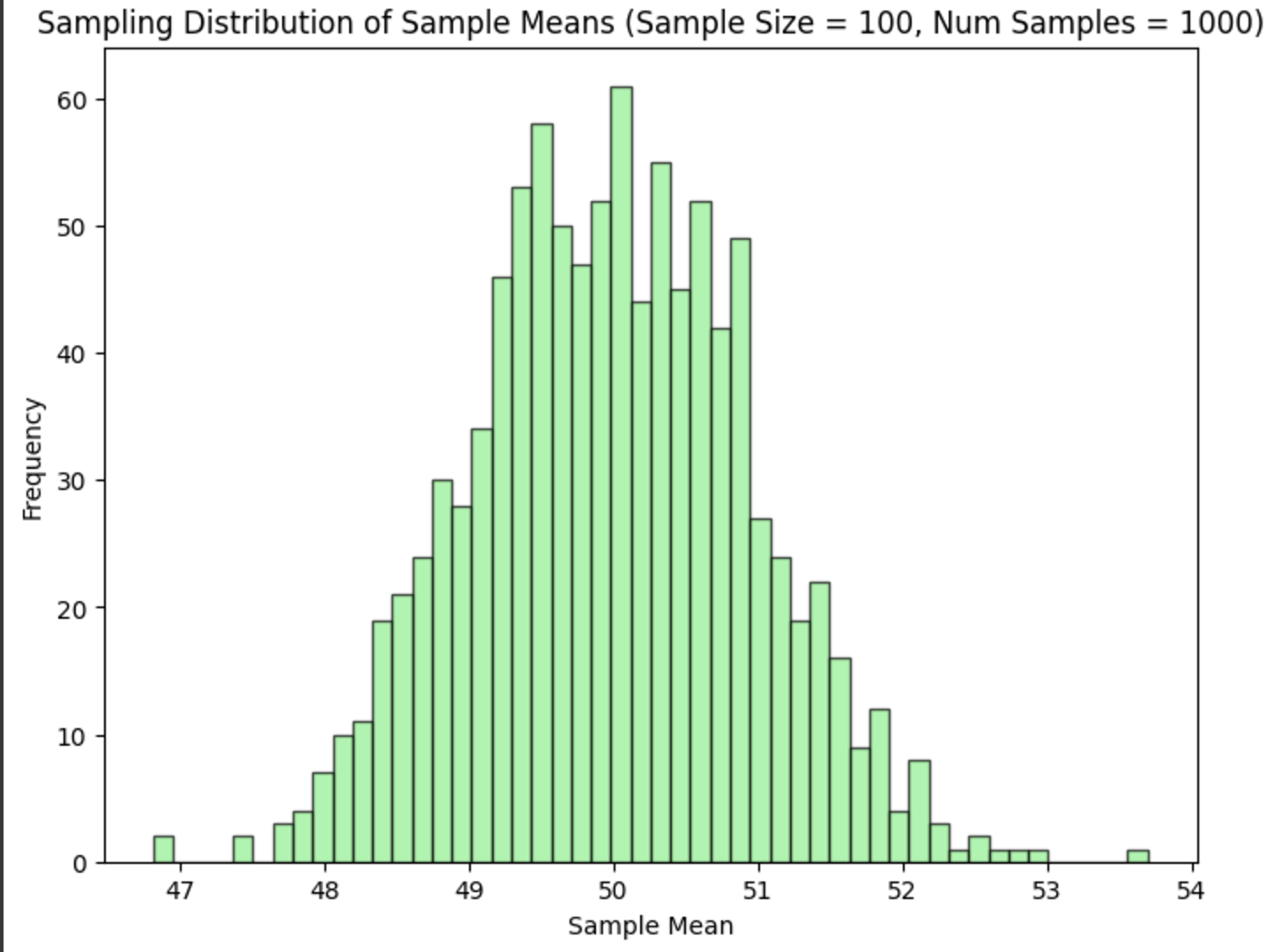
print(f"Mean of population: {np.mean(population)}")

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)}")





**7. Z-TEST**

**NAME : ENIYA B A**

**CLASS: CSE-B**

**ROLL NO : 230701085**

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

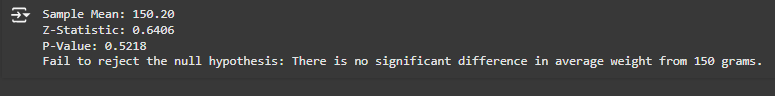
if p\_value < alpha:

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 : ENIYA B A**

**CLASS: CSE-B**

**ROLL NO : 230701085**

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

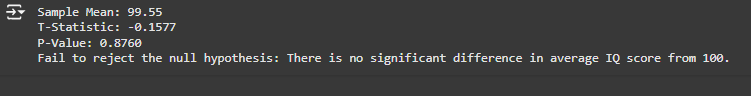
significantly different from 100.")

else:

print("Fail to reject the null hypothesis: There is no

significant difference in average IQ score from 100.")

OUTPUT:



**9. ANOVA-TEST**

**NAME : ENIYA B A**

**CLASS: CSE-B**

**ROLL NO : 230701085**

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

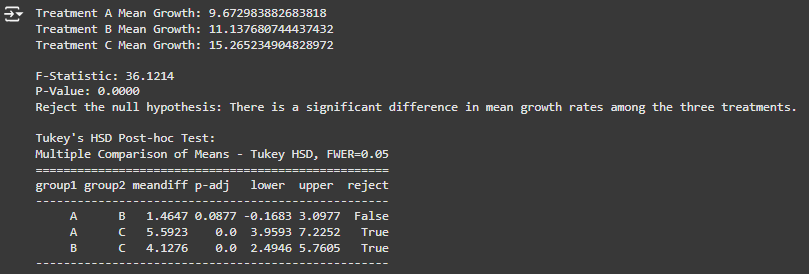
tukey\_results = pairwise\_tukeyhsd(all\_data, treatment\_labels,

alpha=0.05)

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

print(tukey\_results)

OUTPUT:



**10.FEATURE SCALING**

**NAME : ENIYA.B.A**

**ROLL NO : 230701085**

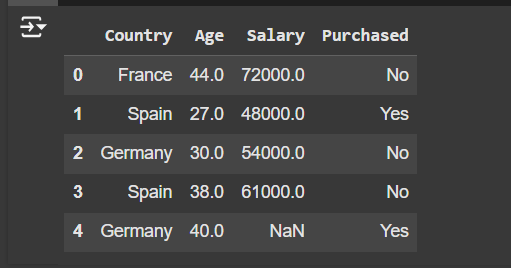
**AIM: To do feature scaling in the given dataset.**

import numpy as np

import pandas as pd

df=pd.read\_csv('Data.csv')

df.head()



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

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

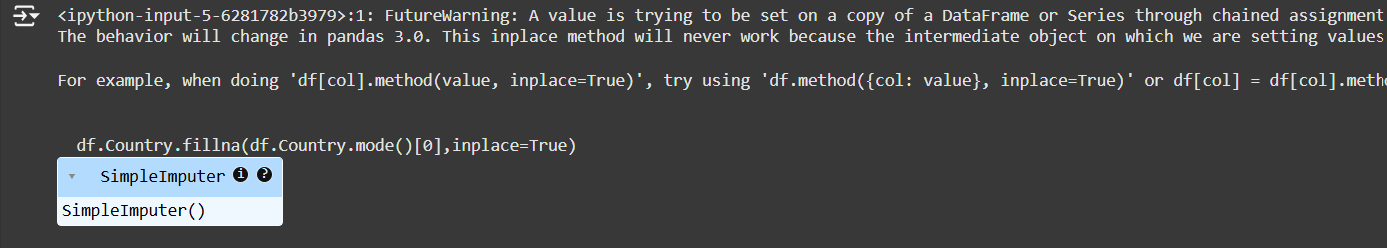
label=df.iloc[:,-1].values

from sklearn.impute import SimpleImputer

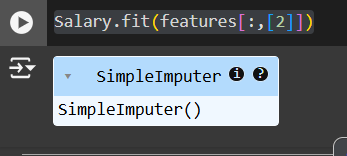
age=SimpleImputer(strategy="mean",missing\_values=np.nan)

Salary=SimpleImputer(strategy="mean",missing\_values=np.nan)

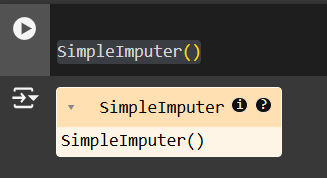
age.fit(features[:,[1]])



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



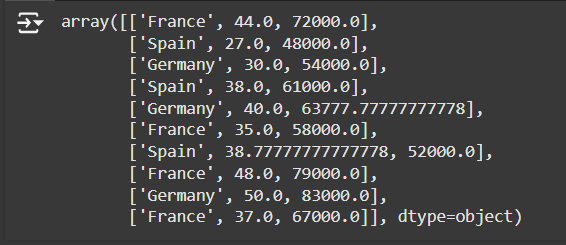
SimpleImputer()



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

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

features

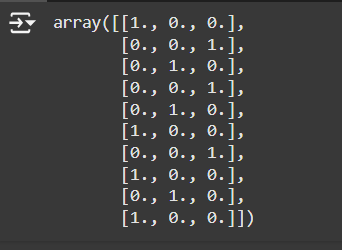


from sklearn.preprocessing import OneHotEncoder

oh = OneHotEncoder(sparse\_output=False)

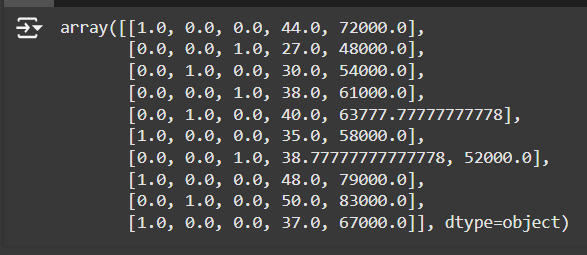
Country=oh.fit\_transform(features[:,[0]])

Country



final\_set=np.concatenate((Country,features[:,[1,2]]),axis=1)

final\_set



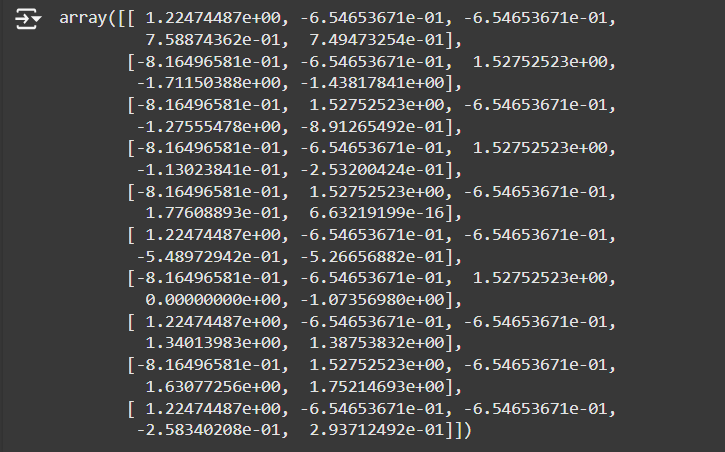
from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

sc.fit(final\_set)

feat\_standard\_scaler=sc.transform(final\_set)

feat\_standard\_scaler



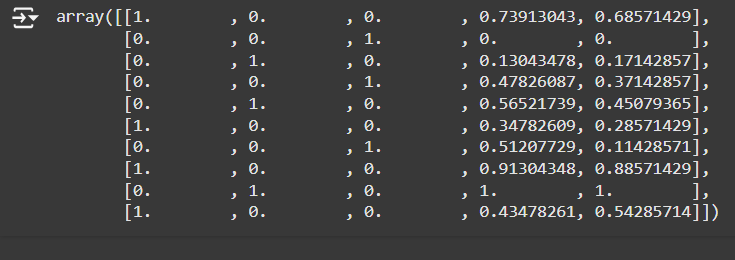
from sklearn.preprocessing import MinMaxScaler

mms=MinMaxScaler(feature\_range=(0,1))

mms.fit(final\_set)

feat\_minmax\_scaler=mms.transform(final\_set)

feat\_minmax\_scaler



**11.LINEAR REGRESSION**

**NAME : ENIYA.B.A**

**CLASS: CSE-B**

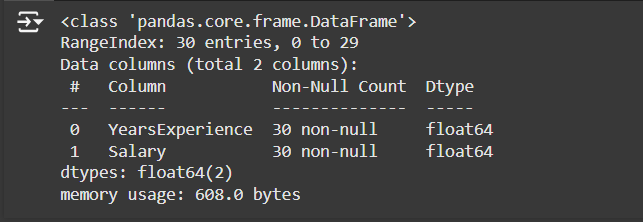
**ROLL NO : 230701085**

import numpy as np

import pandas as pd

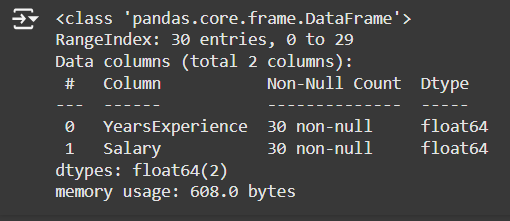
df=pd.read\_csv('Salary\_data.csv')

df.info()

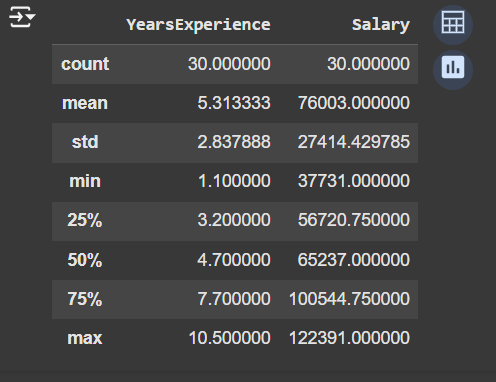


df.dropna(inplace=True)

df.info()



df.describe()



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

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

from sklearn.model\_selection import train\_test\_split

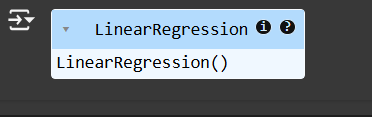
# Assuming `features` and `label` are already defined in your code

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)



model.score(x\_train,y\_train)



model.score(x\_test,y\_test)



model.coef\_



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)



print("Estimated Salary for {} years of experience is {}: " .format(yr\_of\_exp,Salary))



**12.Logistic Regression**

**NAME : ENIYA.B.A**

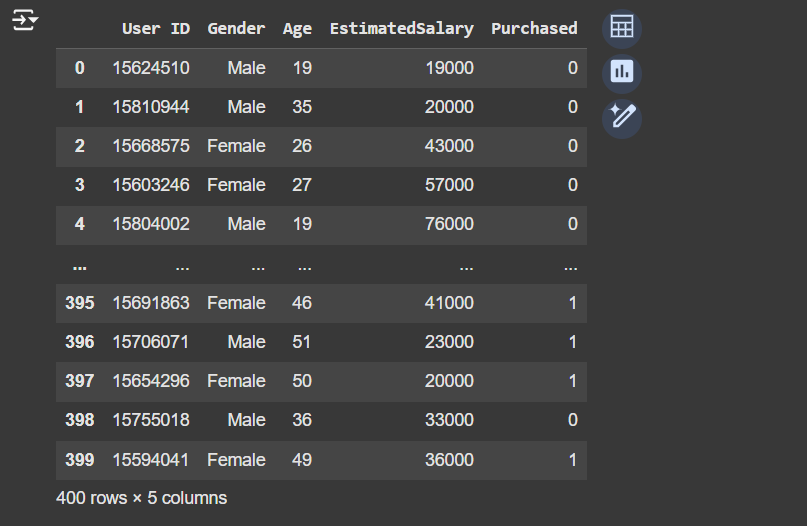
**ROLL NO : 230701085**

import numpy as np

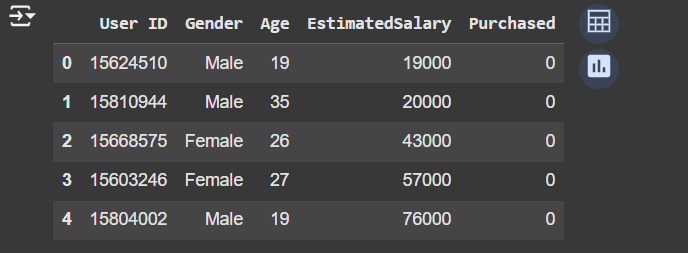
import pandas as pd

df=pd.read\_csv('Social\_Network\_Ads.csv')

df



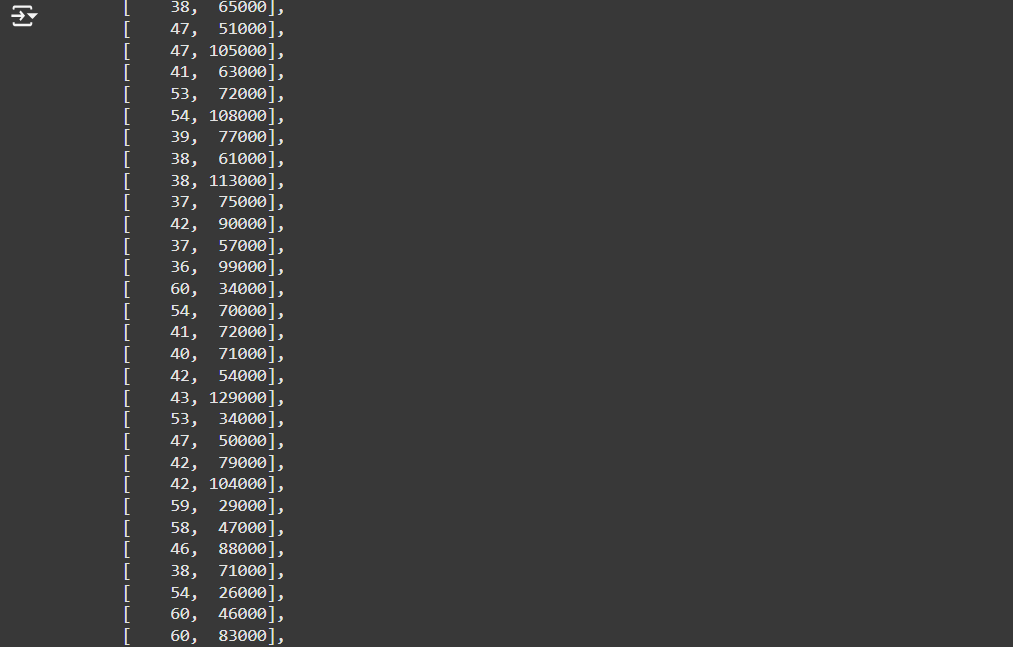
df.head()



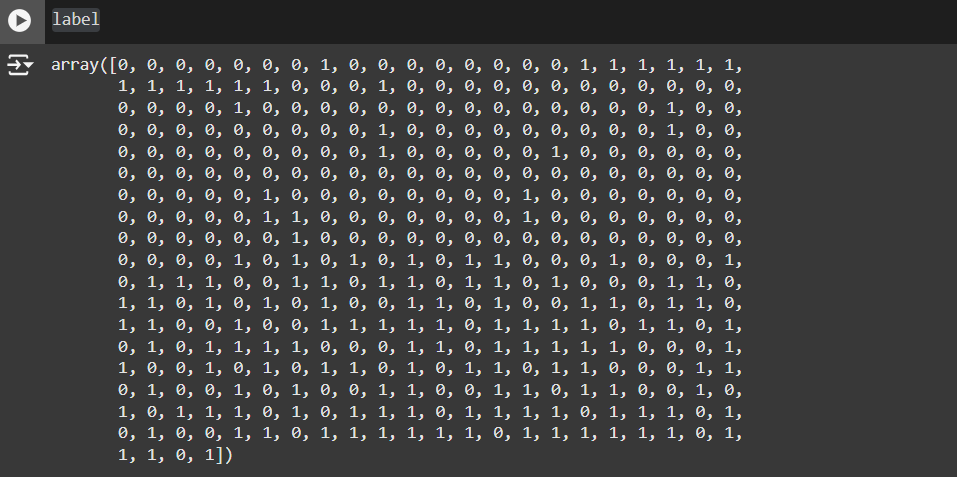
features=df.iloc[:,[2,3]].values

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

features



label



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

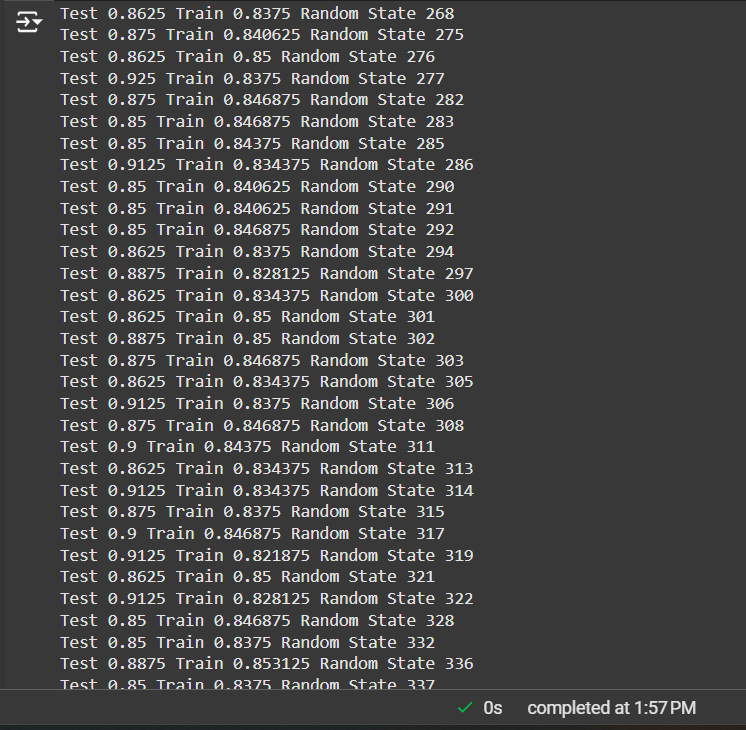
    train\_score = model.score(x\_train, y\_train)

    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))

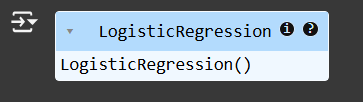


# Assuming features and label are defined earlier in your code

x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, label, test\_size=0.2)

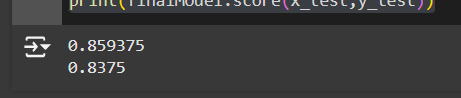
finalModel = LogisticRegression()

finalModel.fit(x\_train, y\_train)



print(finalModel.score(x\_train,y\_train))

print(finalModel.score(x\_test,y\_test))



from sklearn.metrics import classification\_report

print(classification\_report(label,finalModel.predict(features)))

