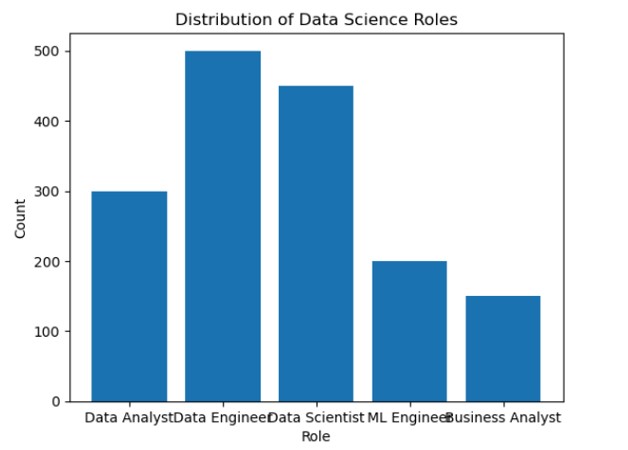
Analyze and visualize the distribution of various data science roles from a dataset

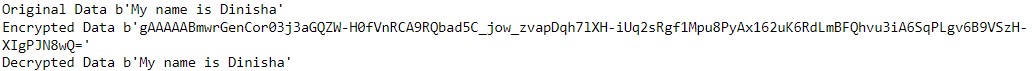
roles= ['Data Analyst', 'Data Engineer', 'Data Scientist', 'ML Engineer",

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



CONDUCT AN EXPERIMENT TO ENCRYPT AND DECRYPT GIVEN SENSITIVE DATA.

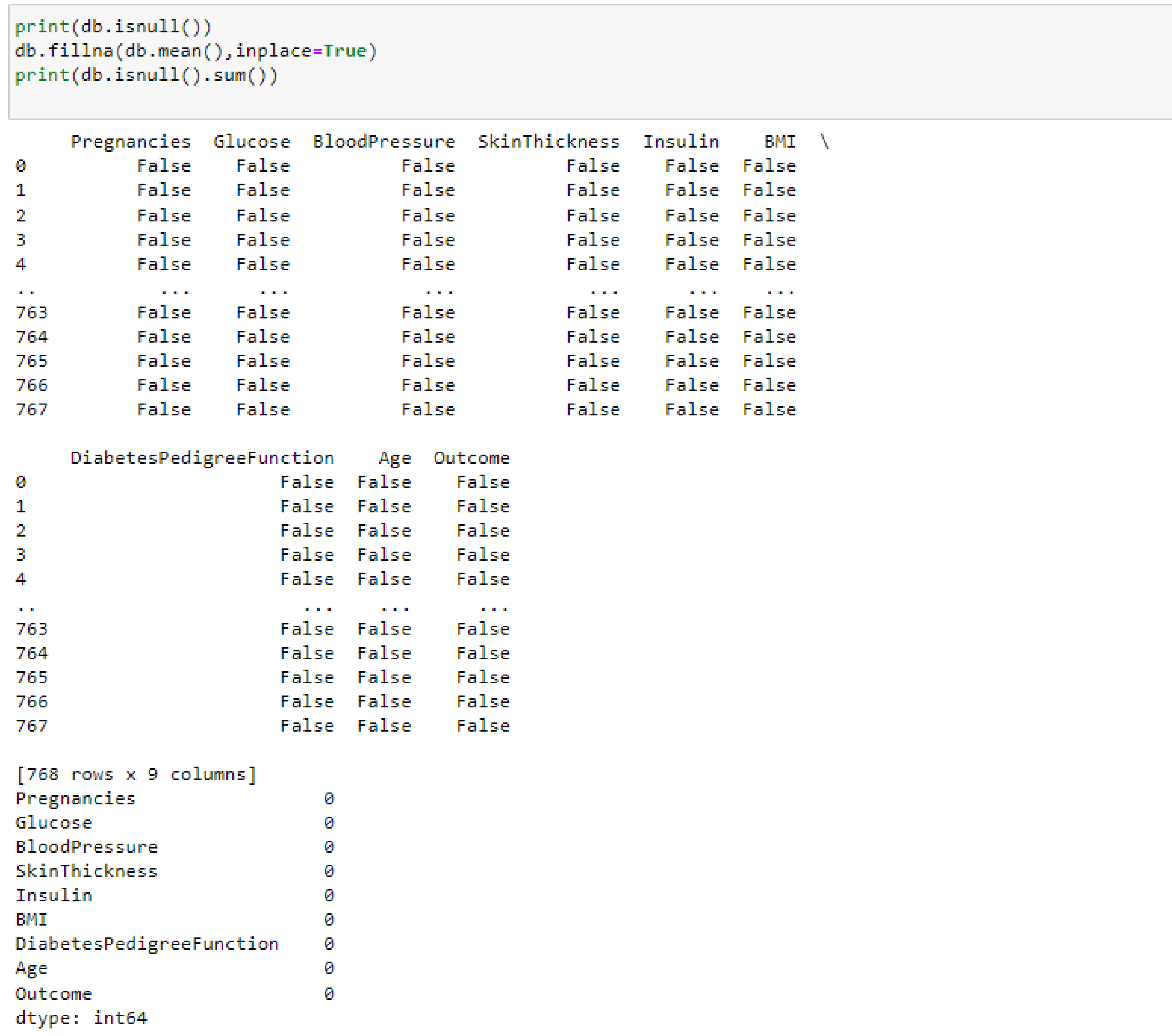


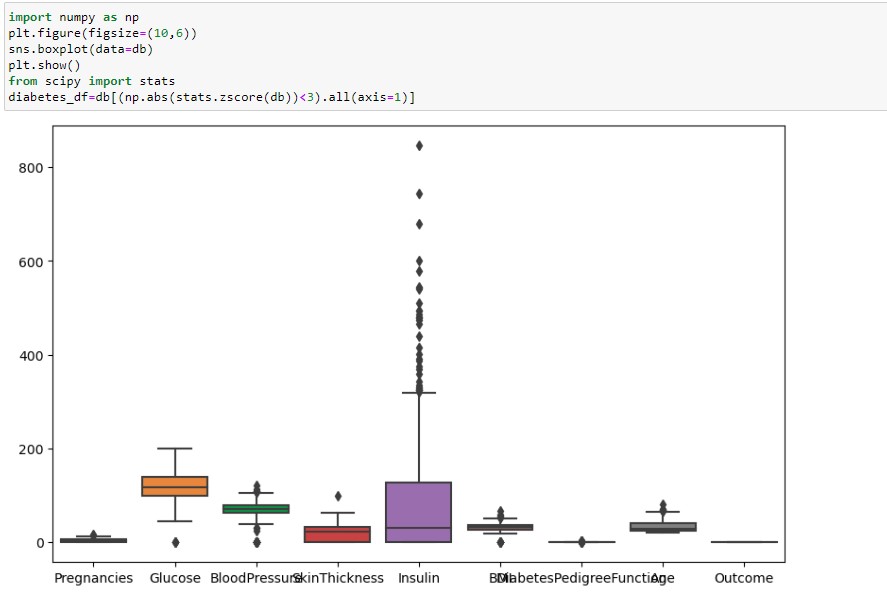


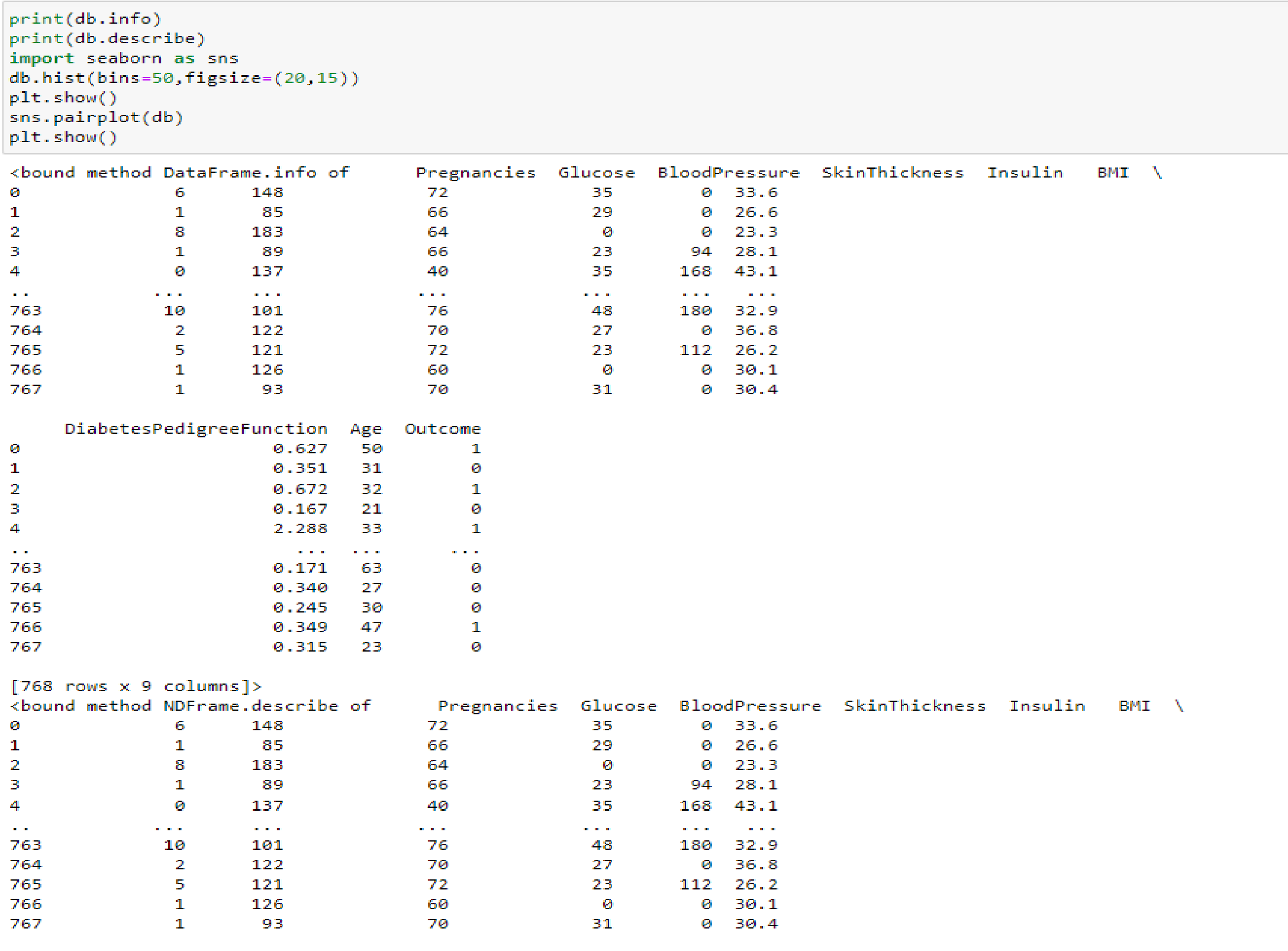
*Count the frequency of occurrence of a word in a body of text is often needed during text processing.*

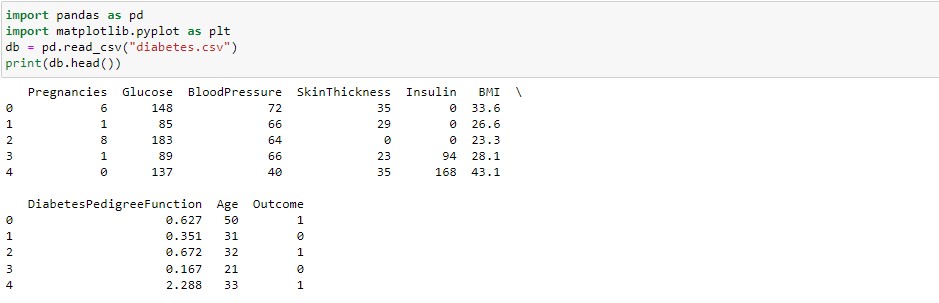


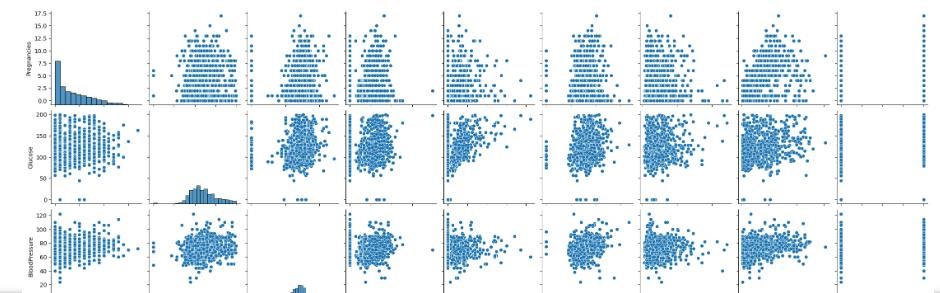
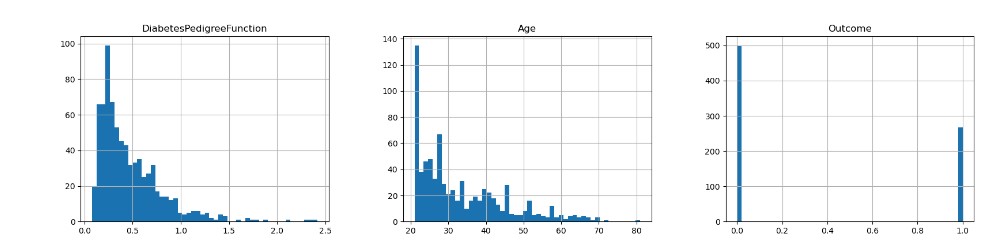
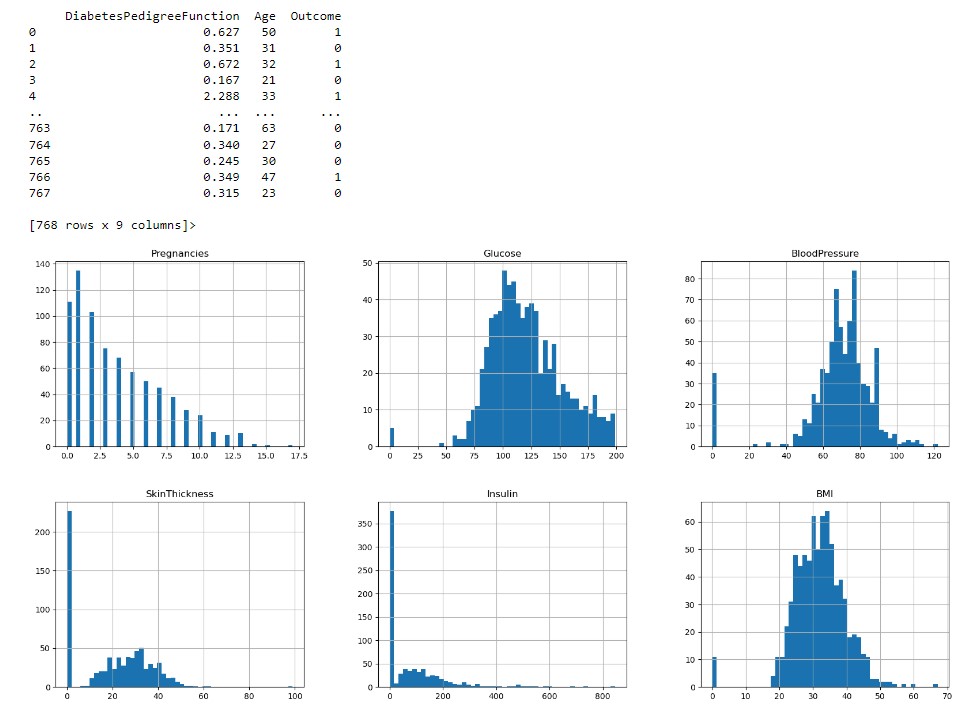
*Data Cleaning*

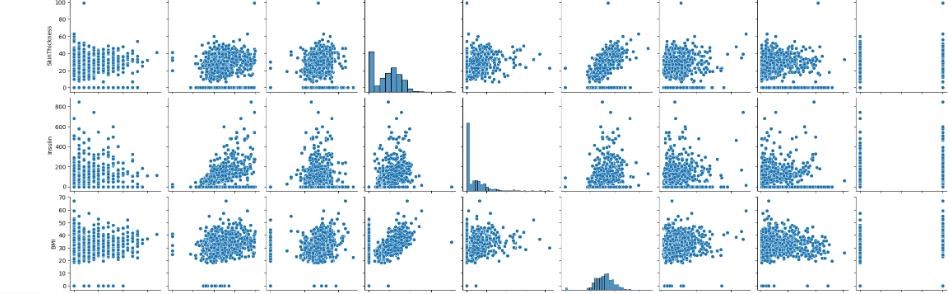


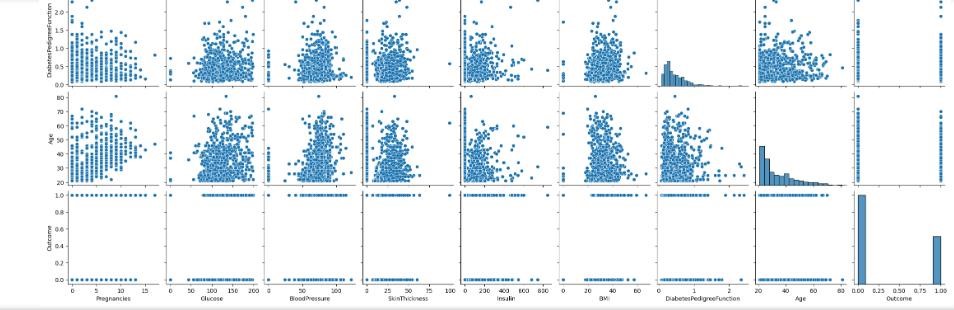


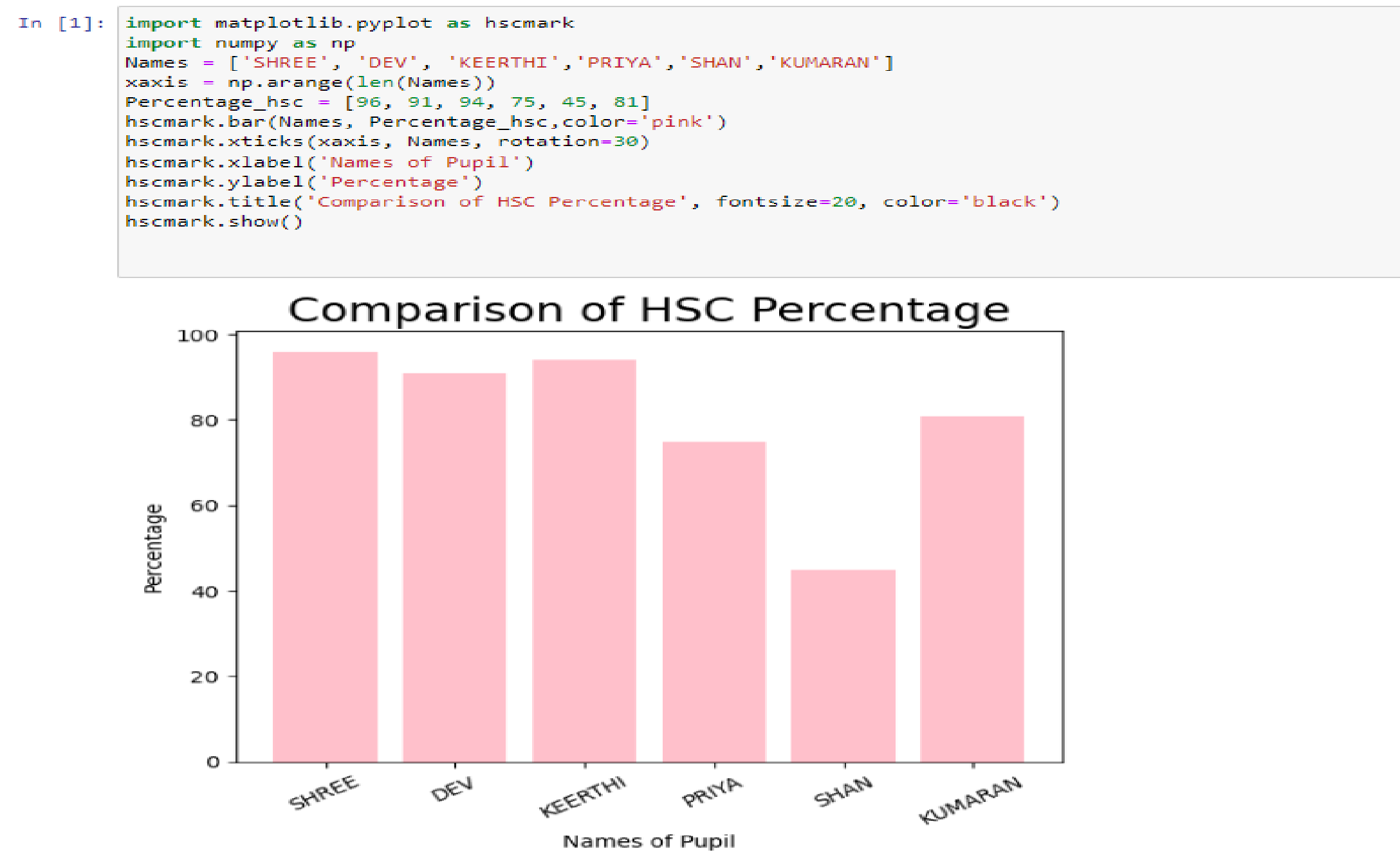
*Data Collection and Initial Exploration*



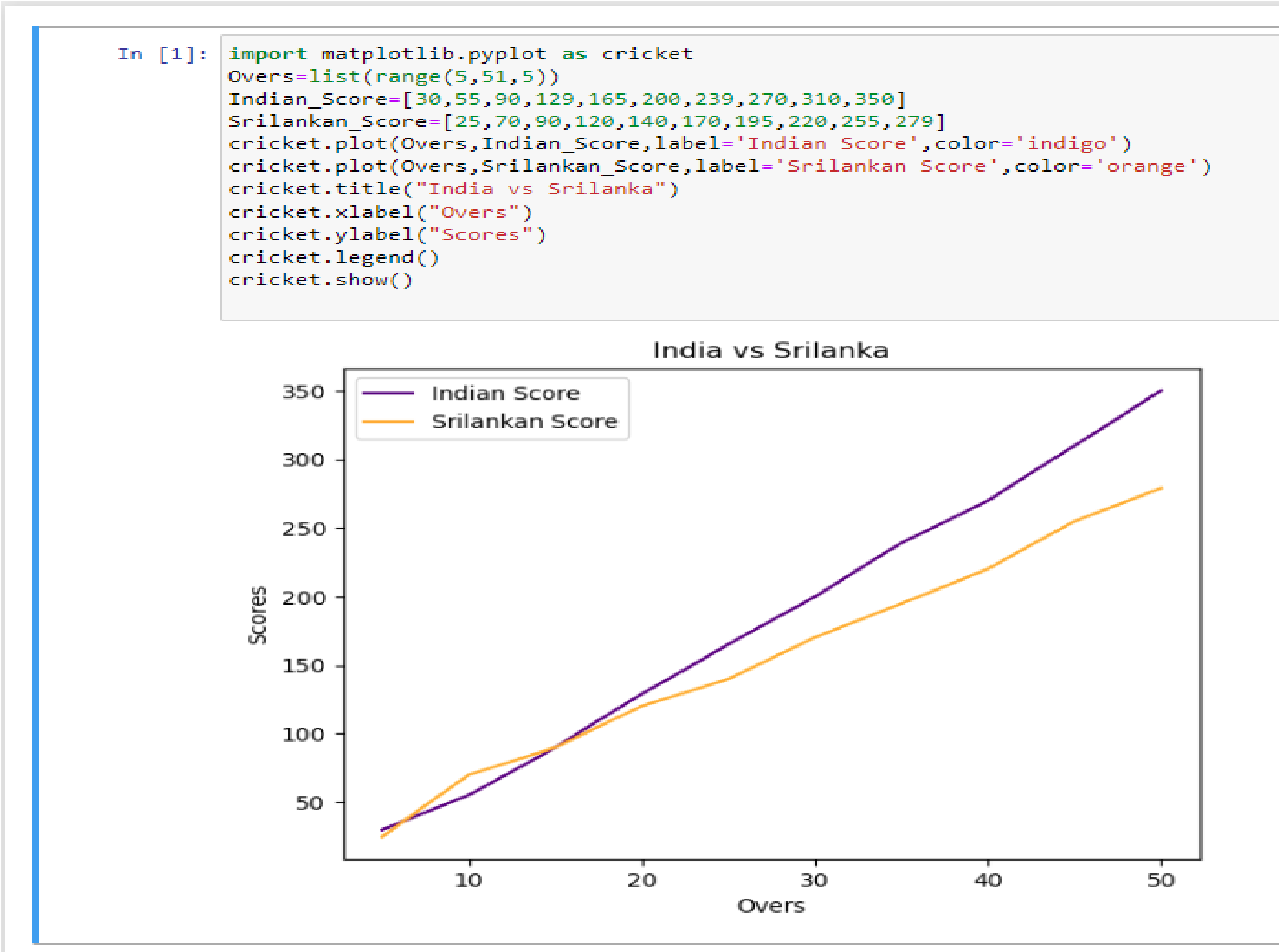


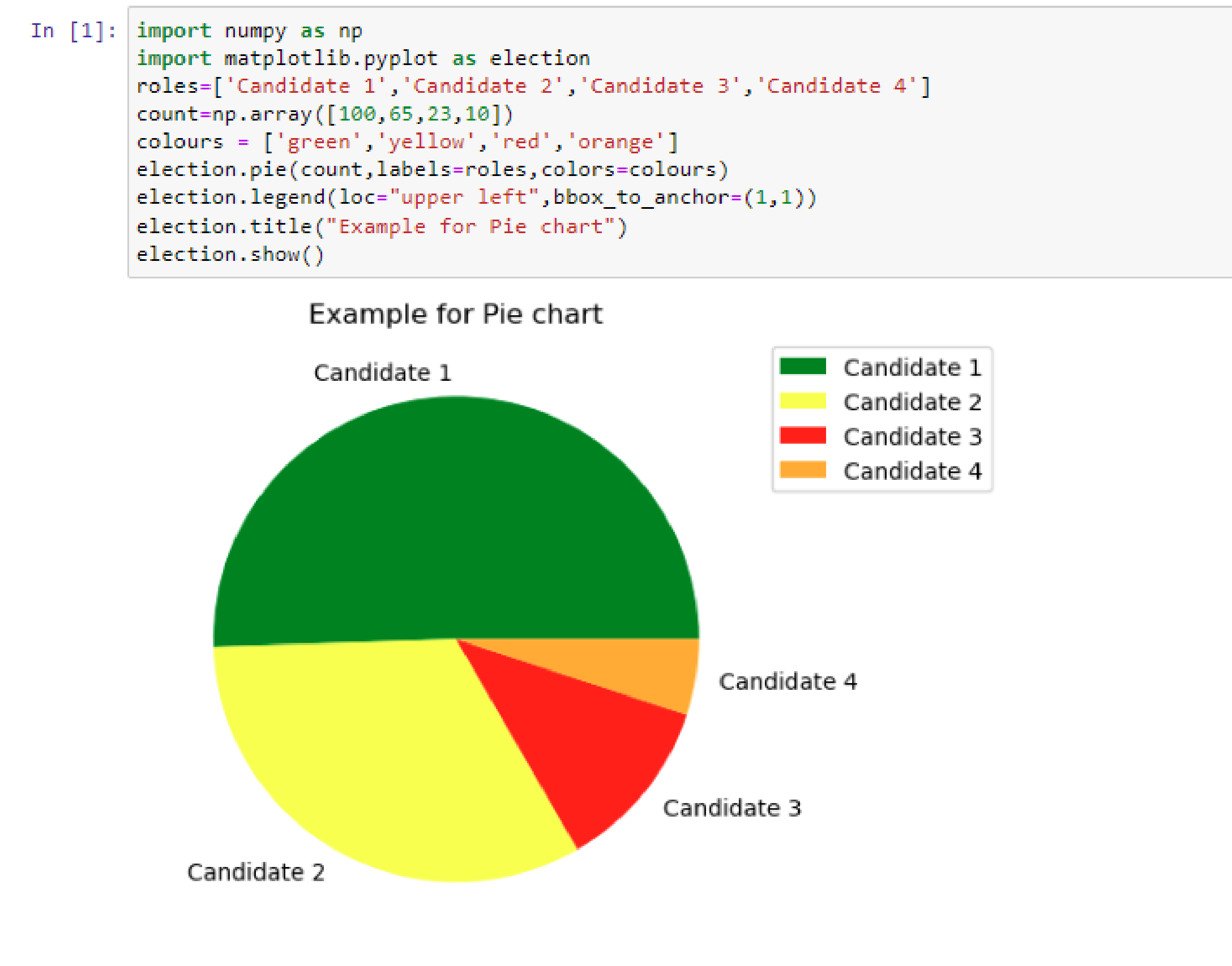




*Experiment to show data visualization using bar chart*

*Experiment to show data visualization using line plot*



*Experiment to show data visualization using pie chart*

Experiments on Structured,

Unstructured and Semi Structured

import pandas as pd structured\_data=pd.DataFrame({

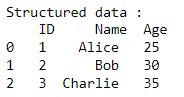
'ID': [1,2,3], 'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25,30,35]

})

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:**





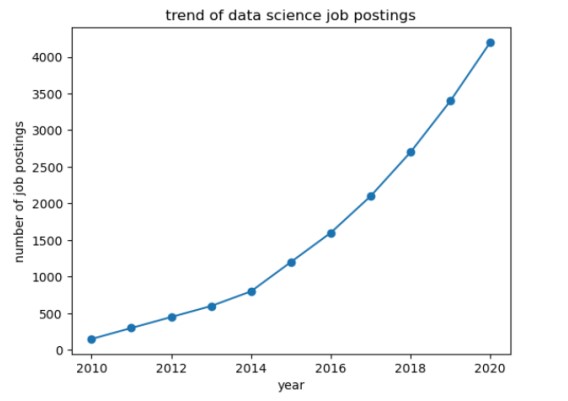
Using Pandas for data manipulation and Matplotlib for visualization

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()



# 4: DATA PREPROCESSING

NAME: Gopikrishnan L

ROLL NO: 230701096

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)

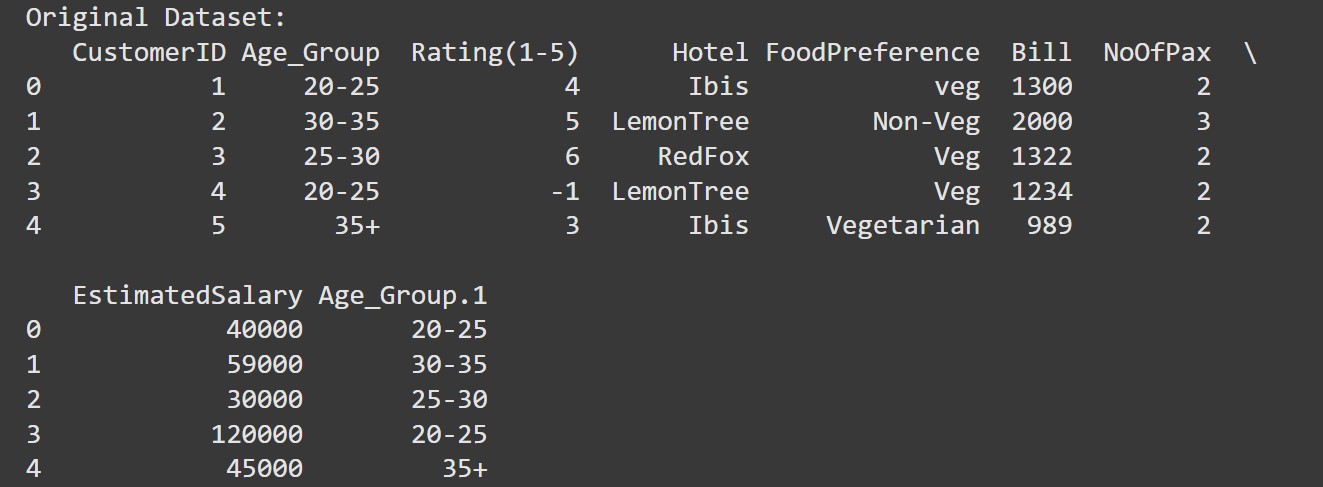
scaler = StandardScaler()

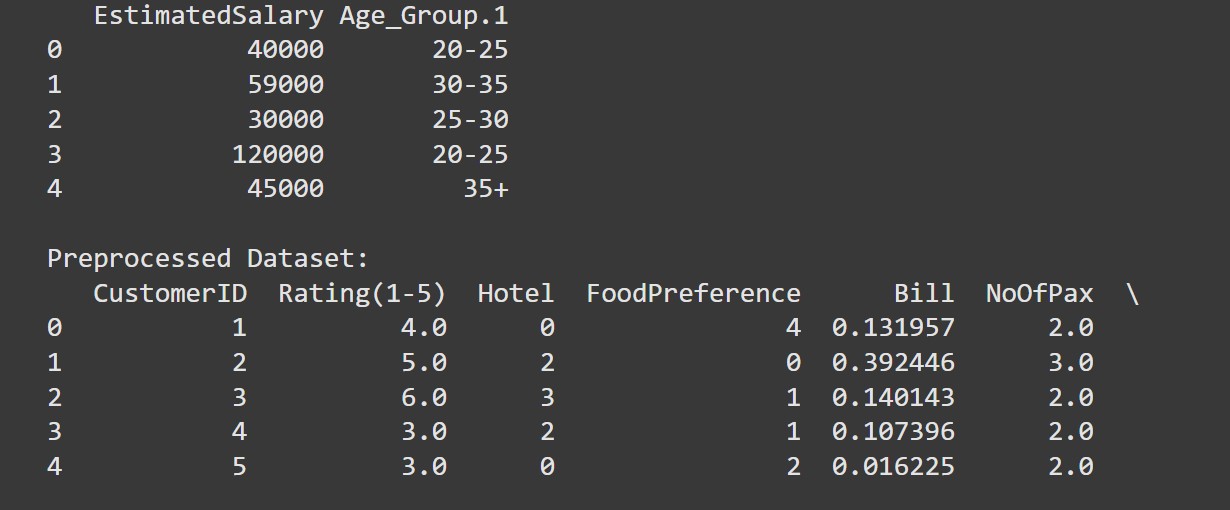
df[['Bill', 'EstimatedSalary']] = scaler.fit\_transform(df[['Bill', 'EstimatedSalary']])

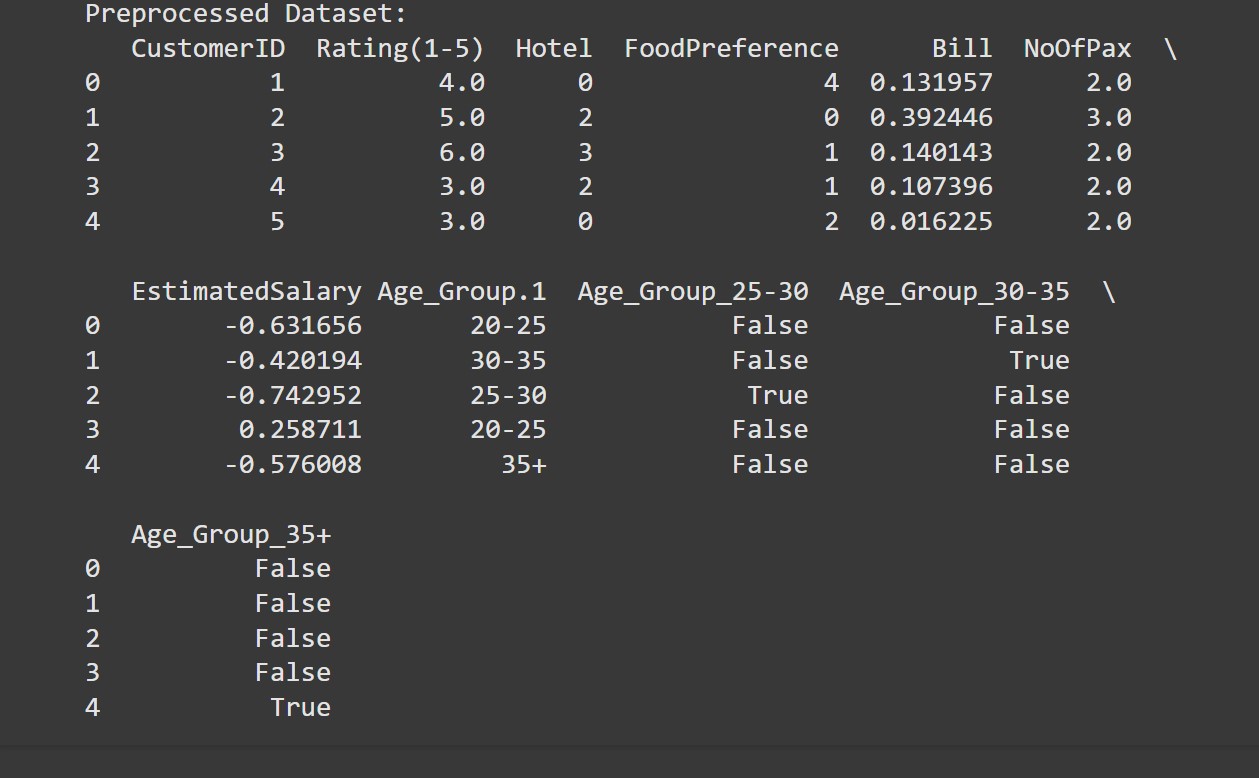
print("\nPreprocessed Dataset:")

print(df.head())

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







5: EDA quantitative and qualitative plot

NAME: GOPIKRISHNAN L

ROLL NO :230701096

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']])

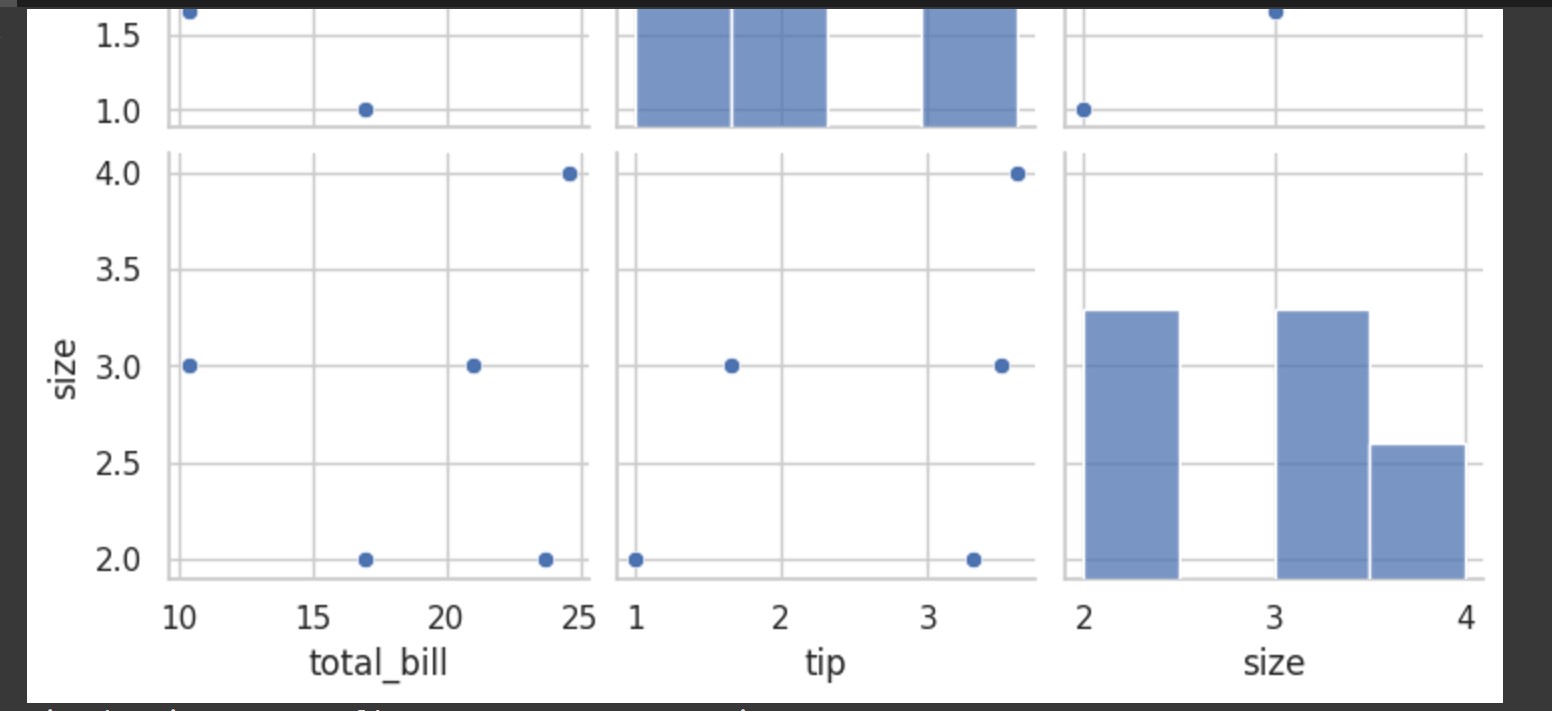
plt.suptitle('Pairplot: Total Bill, Tip, and Size', y=1.02) plt.show()

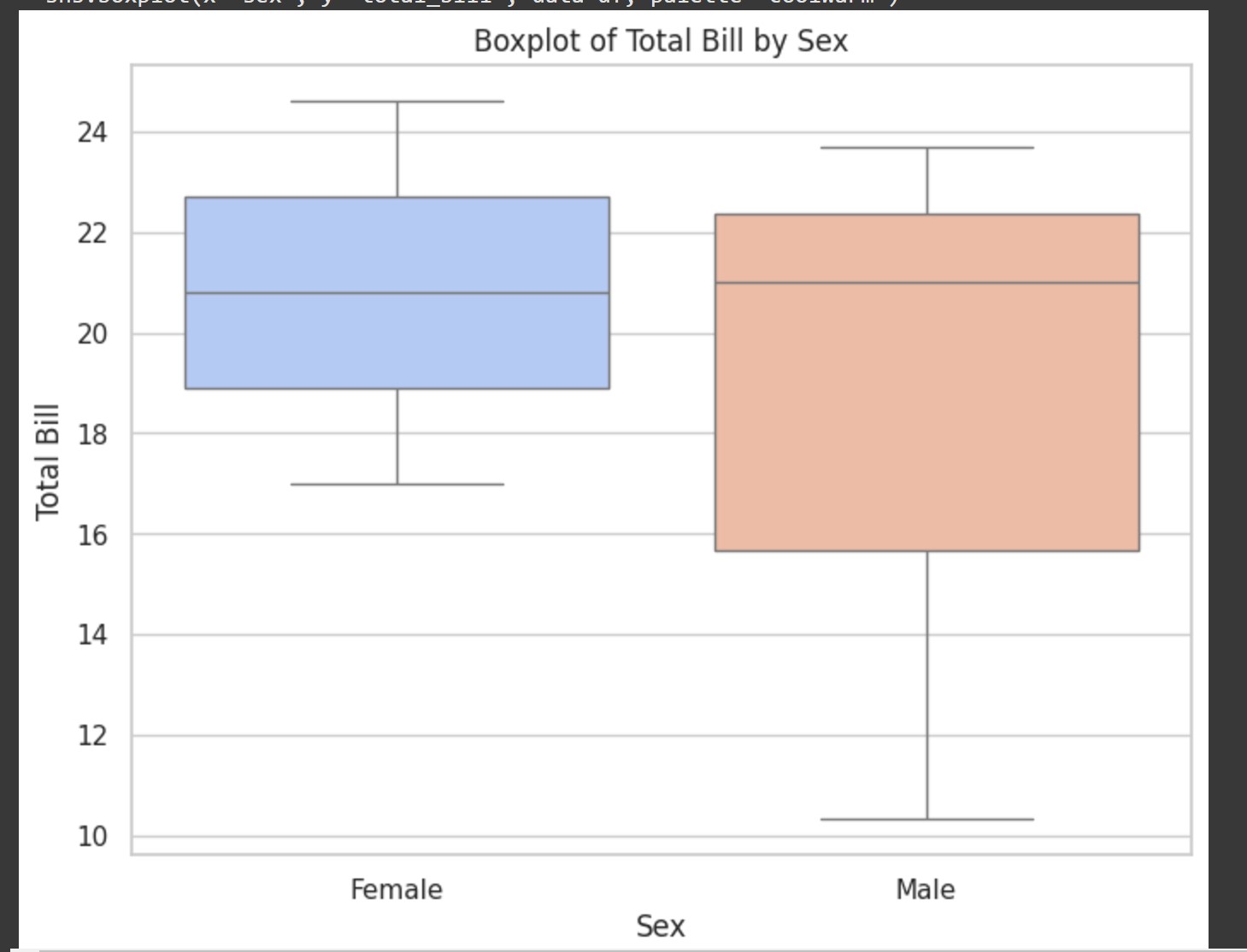
plt.figure(figsize=(8, 6))

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: GOPIKRISHNAN L

ROLL NO: 230701096

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 = []

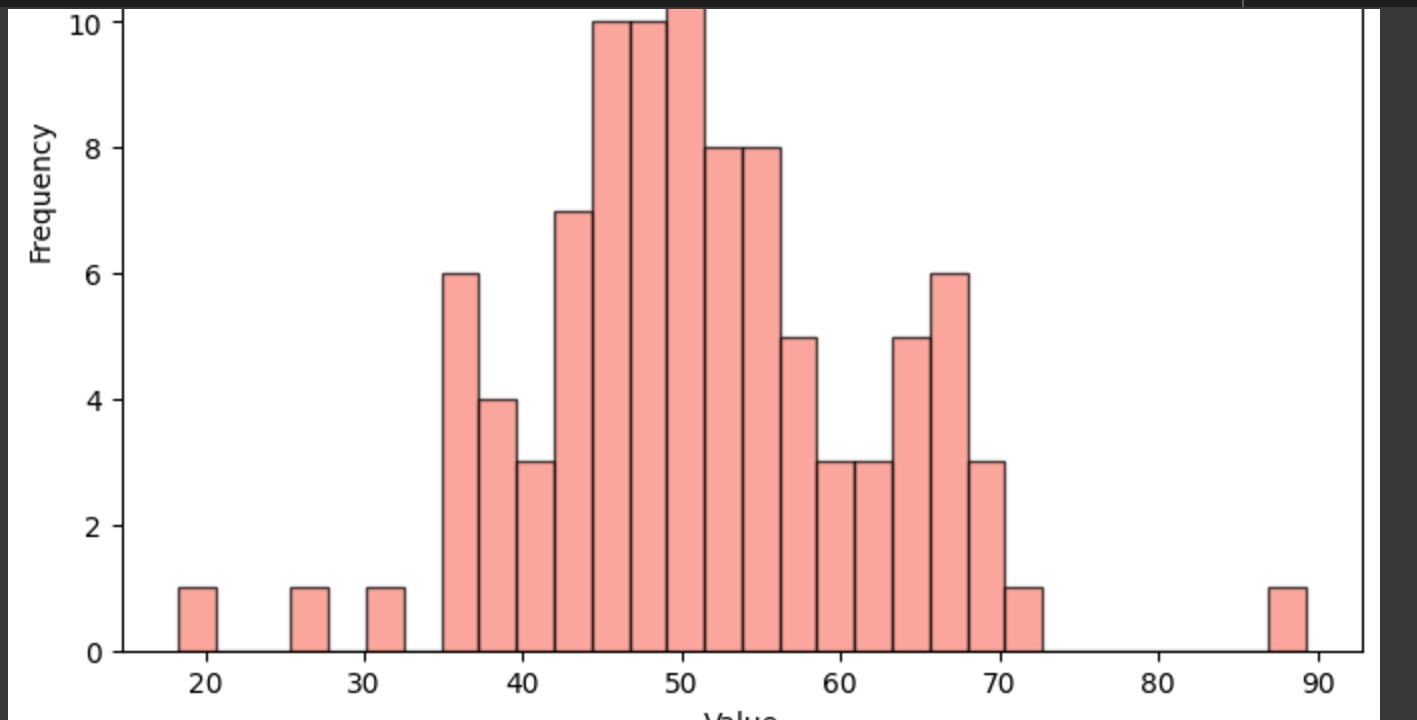
for \_ in range(num\_samples): sample = np.random.choice(population, size=sample\_size, replace=False) sample\_means.append(np.mean(sample))

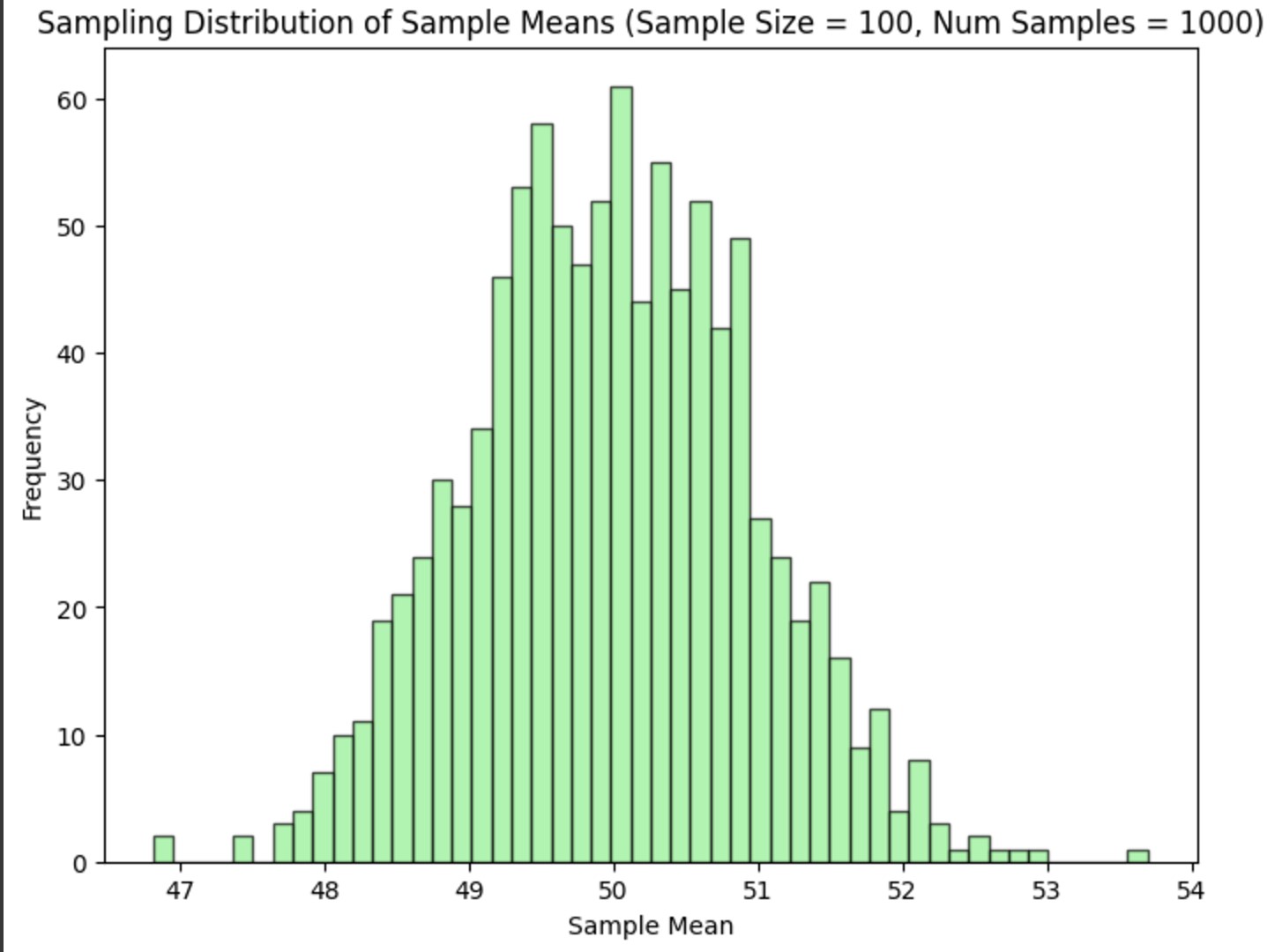
plt.figure(figsize=(8, 6)) plt.hist(sample\_means, bins=50, color='lightgreen', edgecolor='black', alpha=0.7) plt.title(f'Sampling Distribution of Sample Means (Sample Size = {sample\_size}, Num

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 : Gopikrishnan**

**CLASS: CSE-B**

**ROLL NO : 230701096**

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}")

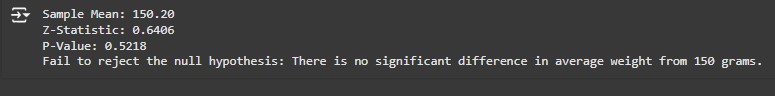
print(f"Z-Statistic: {z\_statistic:.4f}") print(f"P-Value: {p\_value:.4f}")

# 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 : GOPIKRISHNAN L**

**CLASS: CSE-B ROLL NO : 230701096** 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

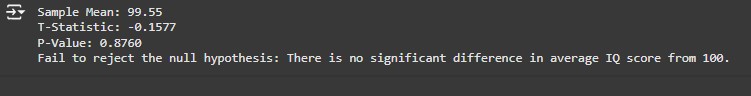
print(f"Sample Mean: {sample\_mean:.2f}") print(f"T-Statistic: {t\_statistic:.4f}") print(f"P-Value: {p\_value:.4f}")

# 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:



# **10.FEATURE SCALING**

**NAME : GOPIKRISHNAN L**

**ROLL NO : 230701096**

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

import numpy as np import pandas as pd df=pd.read\_csv('Data.csv')

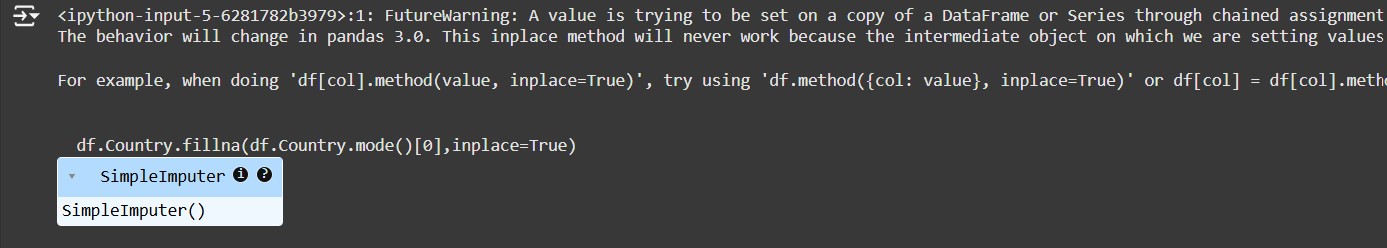
df.head()



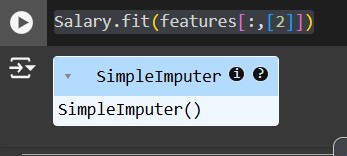
df.Country.fillna(df.Country.mode()[0],inplace=True) features=df.iloc[:,:-1].values label=df.iloc[:,-1].values

from sklearn.impute import SimpleImputer age=SimpleImputer(strategy="mean",missing\_values=np.nan) Salary=SimpleImputer(strategy="mean",missing\_values=np.nan)

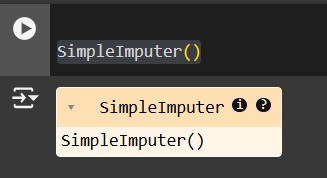
age.fit(features[:,[1]])



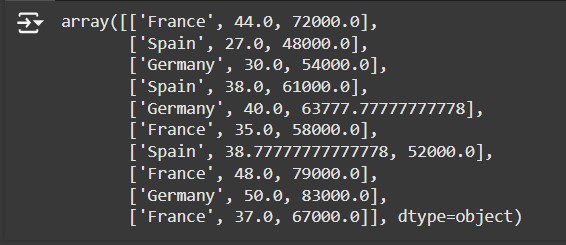
Salary.fit(features[:,[2]])



SimpleImputer()



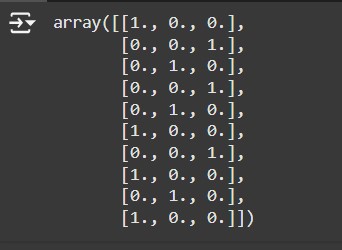
features[:,[1]]=age.transform(features[:,[1]]) features[:,[2]]=Salary.transform(features[:,[2]]) features



from sklearn.preprocessing import OneHotEncoder oh = OneHotEncoder(sparse\_output=False)

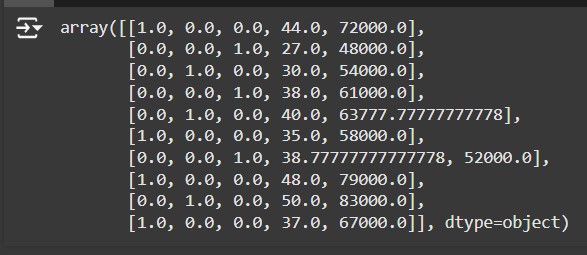
Country=oh.fit\_transform(features[:,[0]])

Country



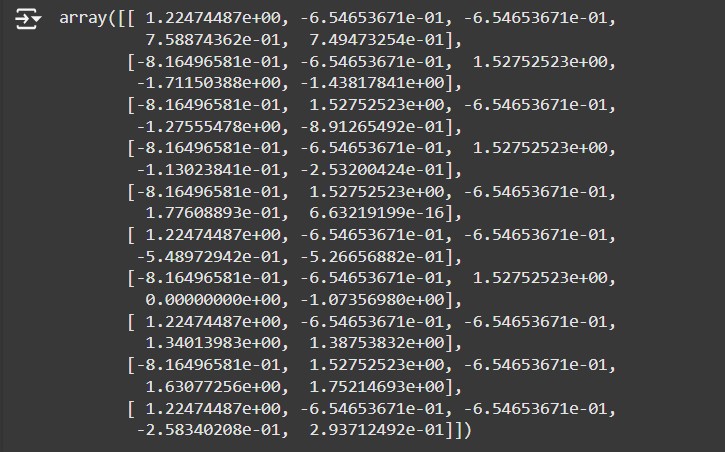
final\_set=np.concatenate((Country,features[:,[1,2]]),axis=1)

final\_set

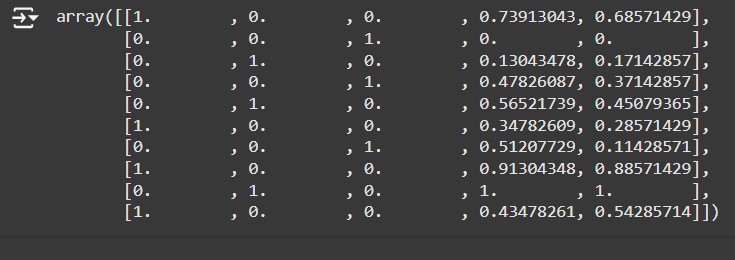


from sklearn.preprocessing import StandardScaler sc=StandardScaler() sc.fit(final\_set)

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

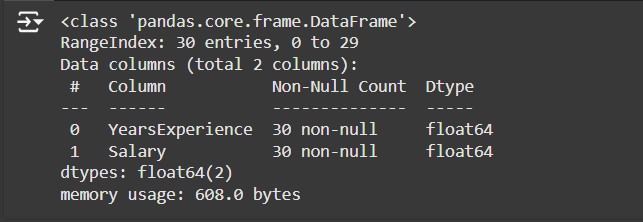
**NAME : DINISHA**

**CLASS: CSE-B**

**ROLL NO : 230701096**

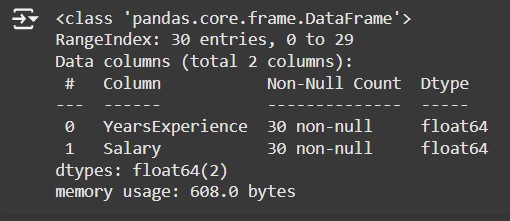
import numpy as np import pandas as pd df=pd.read\_csv('Salary\_data.csv')

df.info()

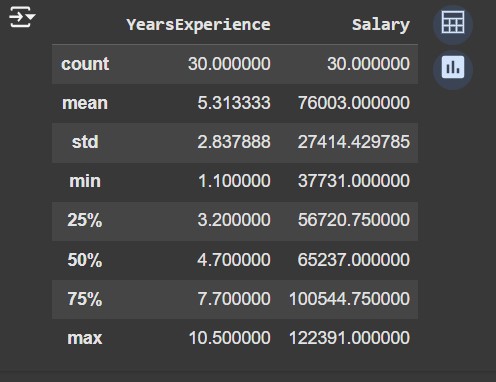


df.dropna(inplace=True)

df.info()



df.describe()

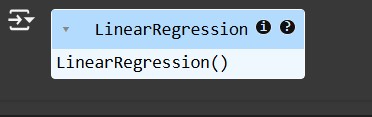


features=df.iloc[:,[0]].values label=df.iloc[:,[1]].values from sklearn.model\_selection import train\_test\_split

# 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))

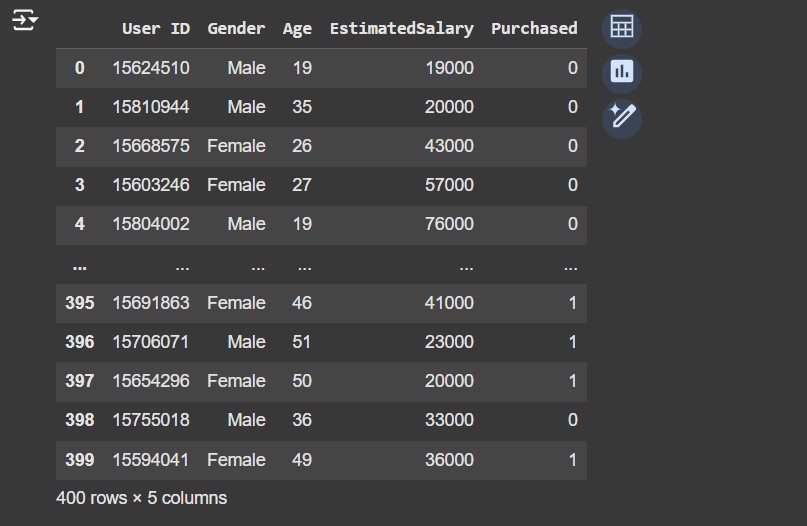


### 12. LOGISTIC REGRESSION

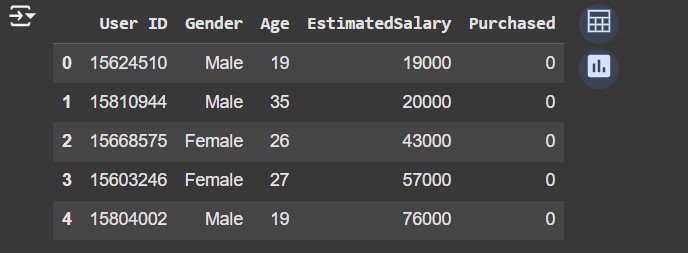
**NAME : GOPIKRISHNAN L**

**ROLL NO : 230701096**

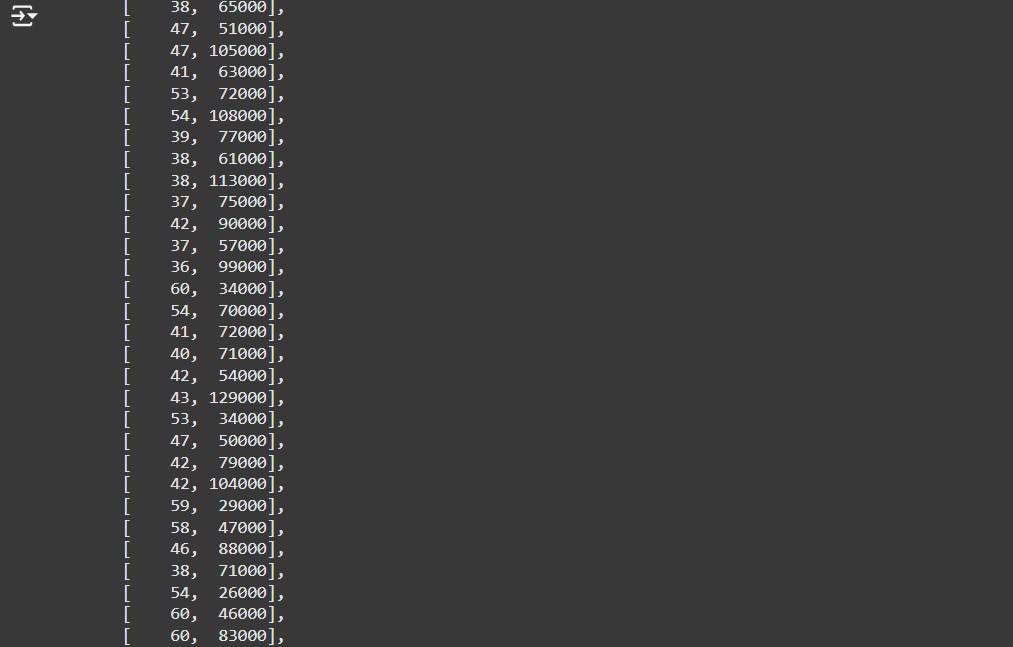
import numpy as np import pandas as pd df=pd.read\_csv('Social\_Network\_Ads.csv') df



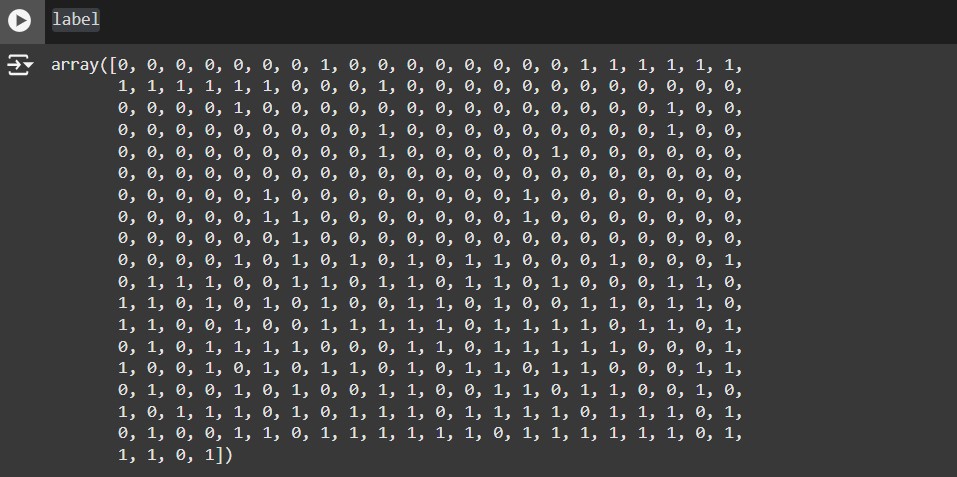
df.head()



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)

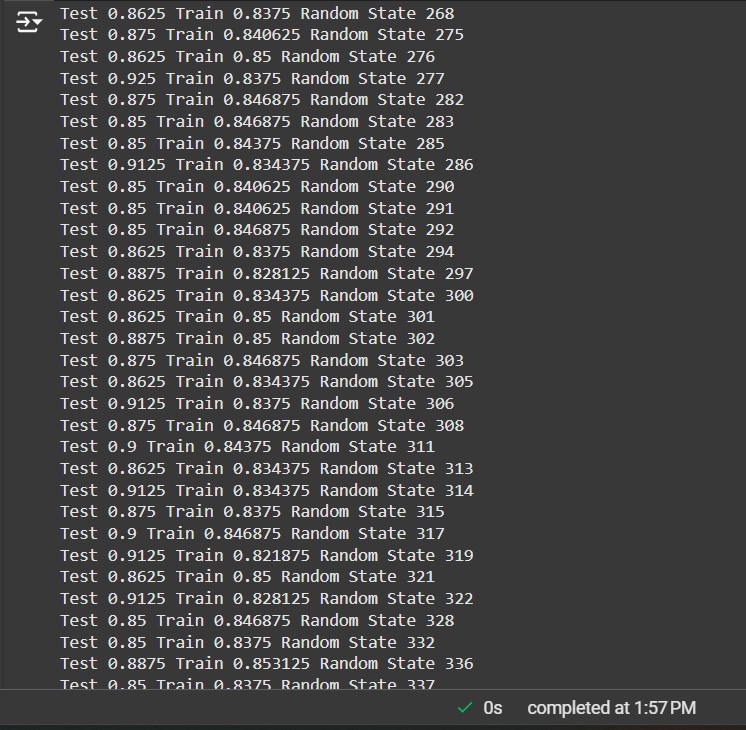
# Initialize the Logistic Regression model model = LogisticRegression()

# 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)

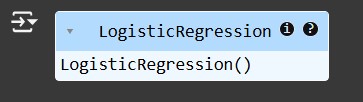
# Print if test score is greater than train score if test\_score > train\_score:

print("Test {} Train {} Random State {}".format(test\_score, train\_score, i))

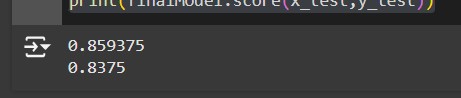


# Assuming features and label are defined earlier in your code x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, label, test\_size=0.2)

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)))

