



RAJALAKSHMI ENGINEERING COLLEGE

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

Department of Computer Science and Engineering

CS23334 Fundamentals of Data Science Lab

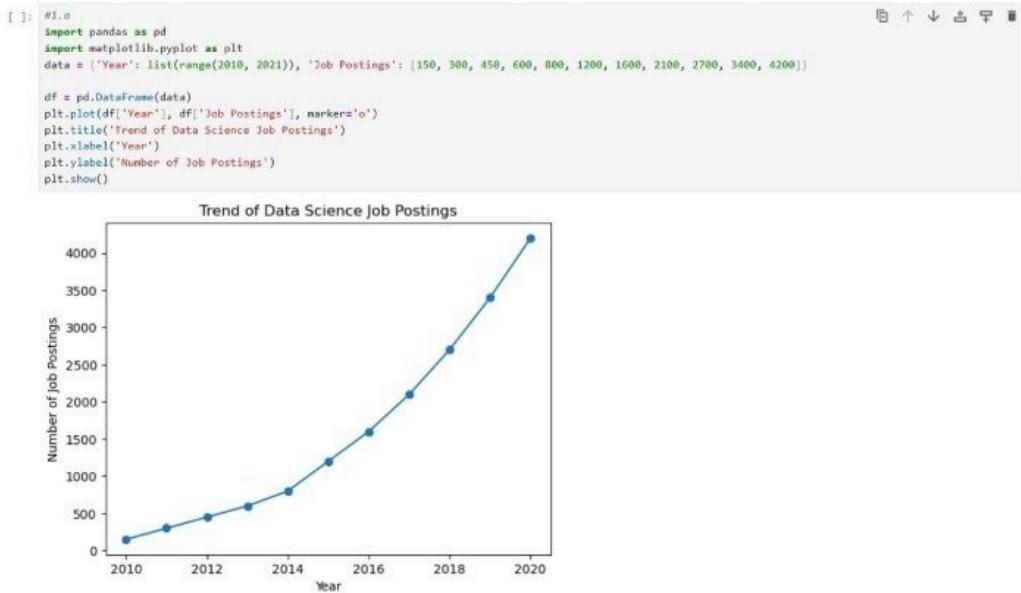
III semester II Year (2023R)

Name of the Student :Amrutha Valli M

Register Number :240701037

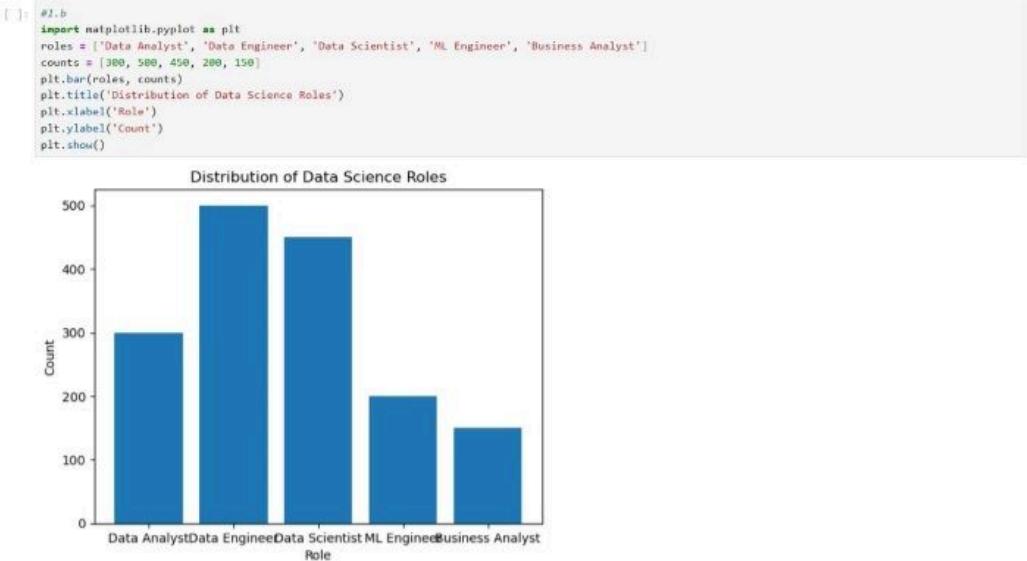
Exp No:1. a Analyze the trend of data science job postings over the last decade

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.



Exp No:1. b Analyze and visualize the distribution of various data science roles (Data Analyst, Data Engineer, Data Scientist, etc.) from a dataset.

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



Exp No:1. c Conduct an Experiment to differentiate Structured, Un-structured and Semi-structured data based on data sets given.

Description: Create small datasets for each type and Explain their characteristics.

```
[ ]: #1.c
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("\nUnstructured Data:\n", unstructured_data)

semi_structured_data = {'ID': 1, 'Name': 'Alice', 'Attributes': {'Height': 165, 'Weight': 68}}
print("\nSemi-structured Data:\n", semi_structured_data)

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-structured Data:
{'ID': 1, 'Name': 'Alice', 'Attributes': {'Height': 165, 'Weight': 68}}
```

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.

```
[ ]: #1.d
from cryptography.fernet import Fernet
key = Fernet.generate_key()
f = Fernet(key)
token = f.encrypt(b"Rajalakshmi Engineering College")
token
b'...'
f.decrypt(token)
b'Rajalakshmi Engineering College'
key = Fernet.generate_key()
cipher_suite = Fernet(key)
plain_text = b"Rajalakshmi Engineering College."
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'Rajalakshmi Engineering College.'
Encrypted Data: b'gAAAAABpAtwBc1lw79rOH2yoV5Nr4J3GM9jy_iQWre9kOcoxM-NXLCRYLGz9_z4STpPBH6hWlkgyxVp1vJBTKAORCwERgFkkQGff23WaMAqp3BYDeB0BN5sNS3xLs62KOyYIDH
7dr8a'
Decrypted Data: b'Rajalakshmi Engineering College.'
```

Exp No:2 Upload and Analyze the data set given in csv format and perform data preprocessing and visualization.

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

```
[ ]: #2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

file_path="sales_data.csv"
df = pd.read_csv(file_path)

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

df['Sales'].fillna(df['Sales'].mean())
df.dropna(subset=['Product', 'Quantity', 'Region'], inplace=True)

print(df.describe())

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

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

df['Date'] = pd.to_datetime(df['Date'], dayfirst=True)
sales_over_time = df.groupby('Date').agg({'Sales': 'sum'}).reset_index()
plt.figure(figsize=(10, 6))
plt.plot(sales_over_time['Date'], sales_over_time['Sales'])
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.title('Sales Over Time')
plt.show()

df['Date'] = pd.to_datetime(df['Date'], dayfirst=True)
sales_over_time = df.groupby('Date').agg({'Sales': 'sum'}).reset_index()
plt.figure(figsize=(10, 6))
plt.plot(sales_over_time['Date'], sales_over_time['Sales'])
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.title('Sales Over Time')
plt.show()

pivot_table = df.pivot_table(values='Sales', index='Region', columns='Product', aggfunc='sum', fill_value=0)
print(pivot_table)

correlation_matrix = df.corr(numeric_only=True)
print(correlation_matrix)

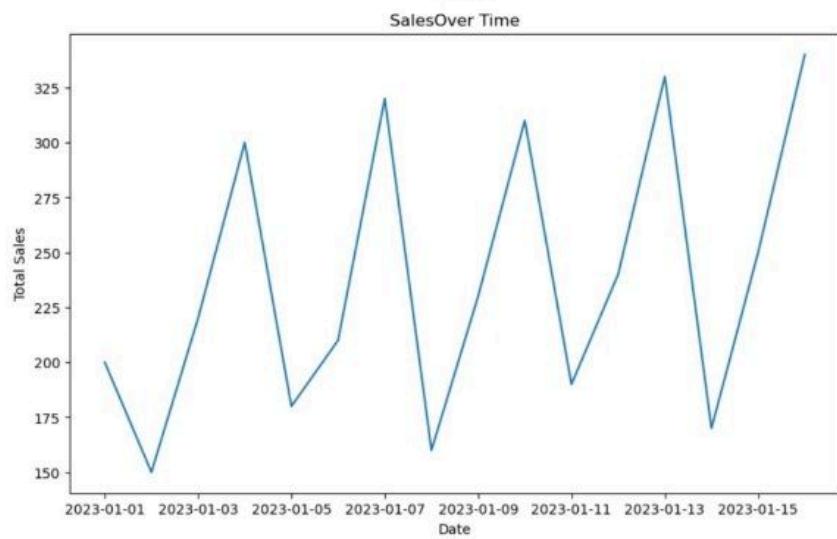
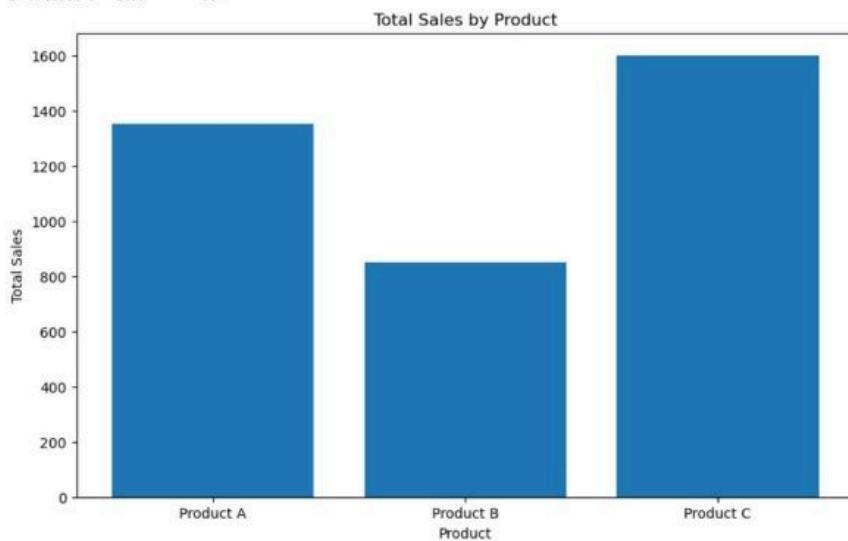
import seaborn as sns
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

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

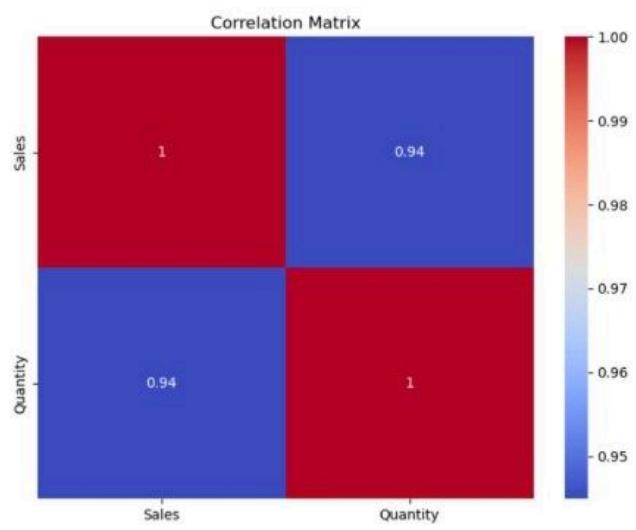
Date 0
Product 0
Sales 0
Quantity 0
Region 0
dtype: int64

	Sales	Quantity
count	16.000000	16.000000
mean	237.500000	5.375000
std	64.031242	1.746425
min	150.000000	3.000000
25%	187.500000	4.000000
50%	225.000000	5.500000
75%	302.500000	7.000000

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



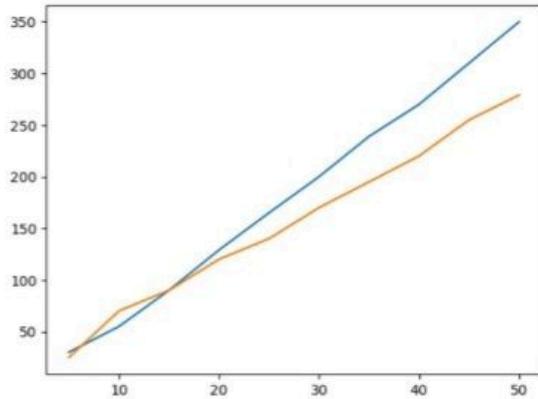
Region	Product A	Product B	Product C
East	0	0	1600
North	1350	0	0
South	0	480	0
West	0	370	0
	Sales	Quantity	
Sales	1.000000	0.944922	
Quantity	0.944922	1.000000	



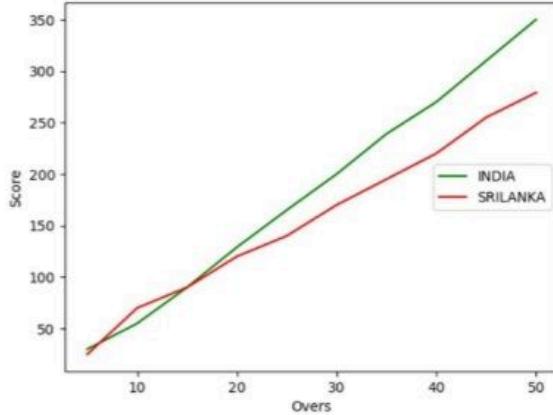
Exp No:3. a Conduct an Experiment to show data visualization using line plot

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

```
[1]: import matplotlib.pyplot as cricket
Overs=list(range(5,51,5))
Indian_Scores=[30,55,98,129,165,200,239,270,310,350]
Srilankan_Scores=[25,70,90,120,140,170,195,220,255,279]
cricket.plot(Overs,Indian_Score)
cricket.plot(Overs,Srilankan_Score)
cricket.show()
cricket.title("INDIA Vs SRILANKA")
cricket.xlabel("Overs")
cricket.ylabel("Score")
cricket.legend()
cricket.plot(Overs,Indian_Score,color="green",label="INDIA")
cricket.plot(Overs,Srilankan_Score,color="red",label="SRILANKA")
cricket.legend(loc="center right")
```



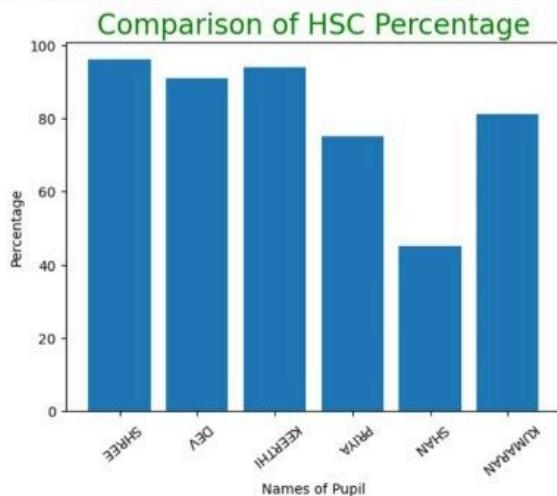
```
[10]: <matplotlib.legend.Legend at 0x1ee61a97dc0>
INDIA Vs SRILANKA
```



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.

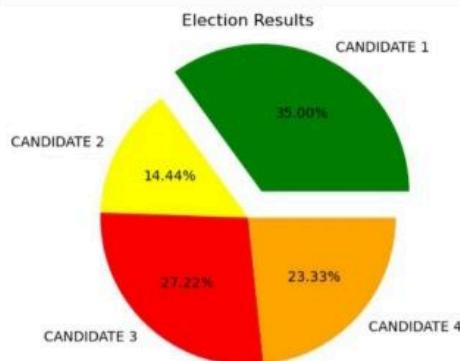
```
[5]: import matplotlib.pyplot as hscmark
import numpy as np
Names = ['SHREE', 'DEV', 'KEERTHI', 'PRIYA', 'SHAN', 'KUMARAN']
xaxis = np.arange(len(Names))
Percentage_hsc = [96, 91, 94, 75, 45, 81]
hscmark.bar(Names, Percentage_hsc)
hscmark.xticks(xaxis, Names, rotation=220)
hscmark.xlabel("Names of Pupil")
hscmark.ylabel("Percentage")
hscmark.title("Comparison of HSC Percentage", fontsize=20, color="green")
hscmark.show()
```



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.

```
[3]: import matplotlib.pyplot as election
labels = ['CANDIDATE 1', 'CANDIDATE 2', 'CANDIDATE 3', 'CANDIDATE 4']
Votes = [315, 130, 245, 210]
colors = ['green', 'yellow', 'red', 'orange']
explode = (0.2, 0, 0, 0)
election.pie(Votes, labels=labels, colors=colors, explode=explode, autopct="%.2f%%")
election.title('Election Results')
election.show()
```



Exp No:4 To Count the frequency of occurrence of a word in a body of text is often needed during text processing.

Description: Import the word_tokenize function and gutenberg.

```
[2]: import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import gutenberg
nltk.download('gutenberg')
nltk.download('punkt')
nltk.download('punkt_tab')
sample = gutenberg.raw("austen-emma.txt")
token = word_tokenize(sample)
wlist = []
for i in range(50):
    wlist.append(token[i])
wordfreq = [wlist.count(w) for w in wlist]
print("Pairs in " + str(list(zip(wlist, wordfreq))))
```

```
[nltk_data] Downloading package gutenberg to
[nltk_data]     C:/Users/merly/AppData/Roaming/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:/Users/merly/AppData/Roaming/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to
[nltk_data]     C:/Users/merly/AppData/Roaming/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
Pairs
[('I', 1), ('Emma', 2), ('by', 1), ('Jane', 1), ('Austen', 1), ('1816', 1), (']', 1), ('VOLUME', 1), ('I', 2), ('CHAPTER', 1), ('I', 2), ('Emma', 2), ('Woodhouse', 1), ('.', 5), ('handsome', 1), ('.', 5), ('clever', 1), ('.', 5), ('and', 3), ('rich', 1), ('.', 5), ('with', 2), ('a', 1), ('comfortable', 1), ('home', 1), ('and', 3), ('happy', 1), ('disposition', 1), ('.', 5), ('seemed', 1), ('to', 1), ('unite', 1), ('some', 1), ('of', 2), ('the', 2), ('best', 1), ('blessings', 1), ('of', 2), ('existence', 1), ('i', 1), ('and', 3), ('had', 1), ('lived', 1), ('nearly', 1), ('twenty-one', 1), ('years', 1), ('in', 1), ('the', 2), ('world', 1), ('with', 2)]
```

Exp No:5 Data Collection and Initial Exploration

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

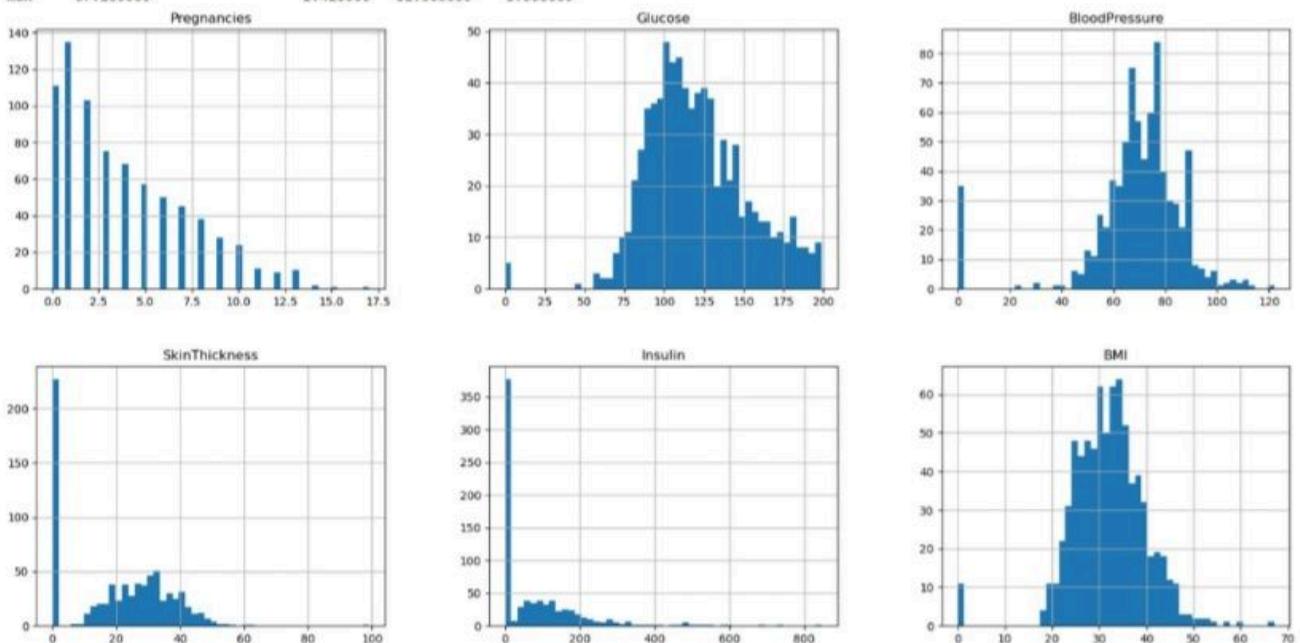
```
[ ]: import pandas as pd
df=pd.read_csv("diabetes.csv")
print(df.info())
print(df.describe())
import matplotlib.pyplot as plt
import seaborn as sns

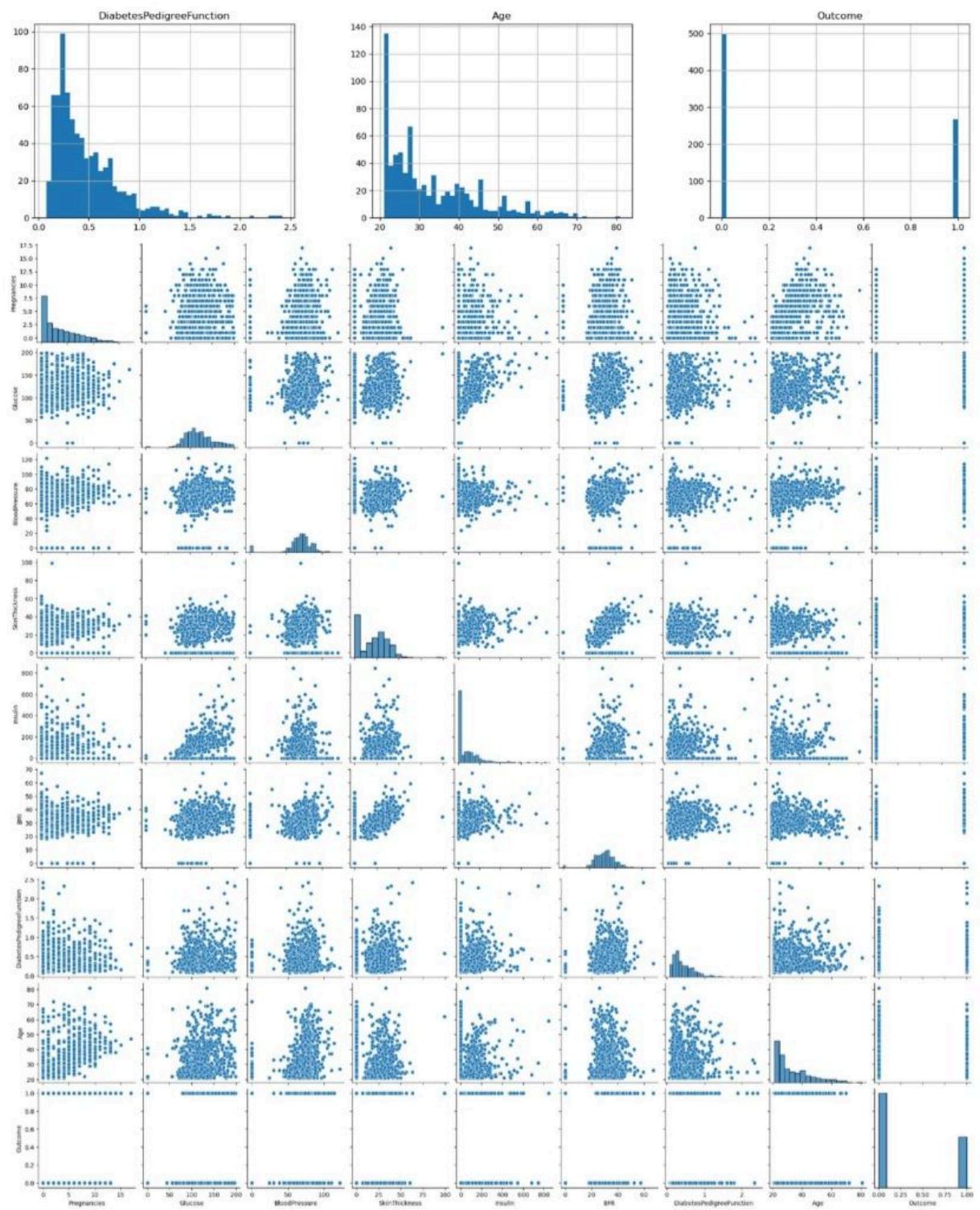
df.hist(bins=50,figsize=(20,15))
plt.show()

sns.pairplot(df)
plt.show()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Pregnancies      768 non-null    int64  
 1   Glucose          768 non-null    int64  
 2   BloodPressure    768 non-null    int64  
 3   SkinThickness    768 non-null    int64  
 4   Insulin          768 non-null    int64  
 5   BMI              768 non-null    float64 
 6   DiabetesPedigreeFunction 768 non-null    float64 
 7   Age              768 non-null    int64  
 8   Outcome          768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479			
std	3.369578	31.972618	19.355807	15.952218	115.244002			
min	0.000000	0.000000	0.000000	0.000000	0.000000			
25%	1.000000	99.000000	62.000000	0.000000	0.000000			
50%	3.000000	117.000000	72.000000	23.000000	30.500000			
75%	6.000000	140.250000	80.000000	32.000000	127.250000			
max	17.000000	199.000000	122.000000	99.000000	846.000000			





Exp:6 Handling Missing and Inappropriate Data in a Dataset

Aim: Demonstrate an Experiment to handle missing data and inappropriate data in a Data set using Python Pandas Library for Data Preprocessing.

```
[1]: import numpy as np
import pandas as pd
df=pd.read_csv("Hotel.csv")
```

	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

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

```
[26]: 0    False
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8    False
9    True
10   False
dtype: bool
```

```
[3]: 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: 920.0+ bytes
```

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

	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

```
[1]: len(df)
[29]: 10
[1]: index=np.array(list(range(0,len(df))))
df.set_index(index,inplace=True)
index
```

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

```
[1]: df
```

[31]:

	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	10	30-35	5	RedFox	non-Veg	6755	4	87777	30-35

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

[32]:

	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

```
[1]: df.CustomerID.loc[df.CustomerID<0]=np.nan
df.Bill.loc[df.Bill<0]=np.nan
df.EstimatedSalary.loc[df.EstimatedSalary<0]=np.nan
df
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df.CustomerID.loc[df.CustomerID<0]=np.nan
C:\Users\REC\AppData\Local\Temp\ipykernel_9864\2080958306.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df.Bill.loc[df.Bill<0]=np.nan
C:\Users\REC\AppData\Local\Temp\ipykernel_9864\2080958306.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df.EstimatedSalary.loc[df.EstimatedSalary<0]=np.nan

[33]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary
0	1.0	20-25	4	Ibis	veg	1300.0	2	40000.0
1	2.0	30-35	5	LemonTree	Non-Veg	2000.0	3	59000.0
2	3.0	25-30	6	RedFox	Veg	1322.0	2	30000.0
3	4.0	20-25	-1	LemonTree	Veg	1234.0	2	120000.0
4	5.0	35+	3	Ibis	Vegetarian	989.0	2	45000.0
5	6.0	35+	3	Ibys	Non-Veg	1909.0	2	122220.0
6	7.0	35+	4	RedFox	Vegetarian	1000.0	-1	21122.0
7	8.0	20-25	7	LemonTree	Veg	2999.0	-10	345673.0
8	9.0	25-30	2	Ibis	Non-Veg	3456.0	3	NaN
9	10.0	30-35	5	RedFox	non-Veg	6755.0	4	87777.0

```
[33]: df['NoOfPax'].loc[(df['NoOfPax']<1) | (df['NoOfPax']>20)] = np.nan
```

C:\Users\REC\AppData\Local\Temp\ipykernel_9864\2129877948.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
[34]:
```

	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	6755.0	4.0	87777.0

```
[35]: df.Age_Group.unique()
```

```
[35]: array(['20-25', '30-35', '25-30', '35+'], dtype=object)
```

```
[36]: df.Hotel.unique()
```

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

```
[37]: df.Hotel.replace(['Ibys'], 'Ibis', inplace=True)
```

```
[37]: df.FoodPreference.unique()
```

```
[37]: <bound method Series.unique of 0>
```

1	Non-Veg
2	Veg
3	Veg
4	Vegetarian
5	Non-Veg
6	Vegetarian
7	Veg
8	Non-Veg
9	non-Veg

Name: FoodPreference, dtype: object

```
[38]: df.FoodPreference.replace(['Vegetarian', 'veg'], 'Veg', inplace=True)
```

```
[38]: df.FoodPreference.replace(['non-Veg'], 'Non-Veg', inplace=True)
```

```
[39]: df.EstimatedSalary.fillna(round(df.EstimatedSalary.mean(), inplace=True))
```

```
[39]: df.NoOfPax.fillna(round(df.NoOfPax.median(), inplace=True))
```

```
[39]: df['Rating(1-5)'].fillna(round(df['Rating(1-5)'].median(), inplace=True))
```

```
[39]: df.Bill.fillna(round(df.Bill.mean(), inplace=True))
```

```
[39]:
```

	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	Veg	989.0	2.0	45000.0
5	6.0	35+	3	Ibis	Non-Veg	1909.0	2.0	122220.0
6	7.0	35+	4	RedFox	Veg	1000.0	2.0	21122.0
7	8.0	20-25	7	LemonTree	Veg	2999.0	2.0	345673.0
8	9.0	25-30	2	Ibis	Non-Veg	3456.0	3.0	96755.0
9	10.0	30-35	5	RedFox	Non-Veg	6755.0	4.0	87777.0

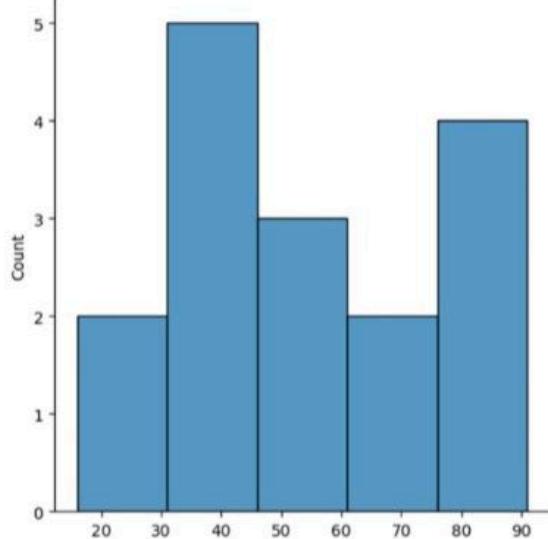
Exp:7 Experiment to detect outliers in a given data set.

Description: Understand the procedure to identify the outliers in a given dataset

```
[1]: import numpy as np  
array=np.random.randint(1,100,16)  
array  
[40]: array([38, 16, 55, 79, 35, 60, 47, 78, 18, 42, 89, 91, 43, 70, 38, 64])  
[1]: array.mean()  
[41]: 53.9375  
[1]: np.percentile(array,25)  
[42]: 38.0  
[1]: np.percentile(array,50)  
[43]: 51.0  
[1]: np.percentile(array,75)  
[44]: 72.0  
[1]: np.percentile(array,100)  
[45]: 91.0  
[1]: def outDetection(array):  
    sorted(array)  
    Q1,Q3=np.percentile(array,[25,75])  
    IQR=Q3-Q1  
    lr=Q1-(1.5*IQR)  
    ur=Q3+(1.5*IQR)  
    return lr,ur  
lr,ur=outDetection(array)  
lr,ur  
[46]: (-13.0, 123.0)
```

```
[1]: import seaborn as sns  
%matplotlib inline  
sns.displot(array)
```

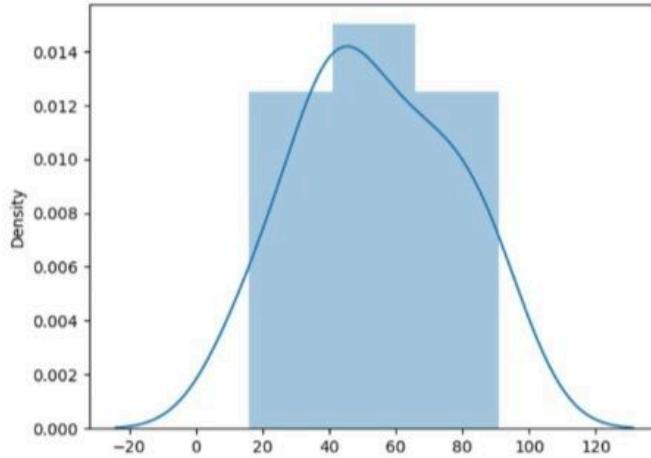
```
[47]: <seaborn.axisgrid.FacetGrid at 0x1ee591f9360>
```



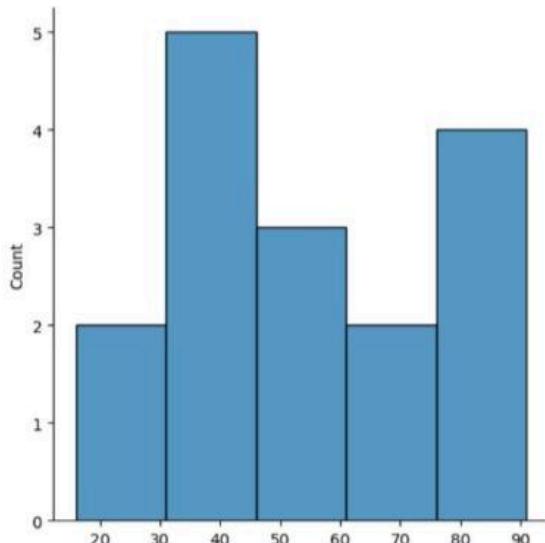
```
[1]: sns.distplot(array)
C:\Users\REC\AppData\Local\Temp\ipykernel_9864\1133588802.py:1: UserWarning:
'distplot' is a deprecated function and will be removed in seaborn v0.14.0.
Please adapt your code to use either 'displot' (a figure-level function with
similar flexibility) or 'histplot' (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

sns.distplot(array)
[48]: <Axes: ylabel='Density'>
```



```
[49]: new_array=array((array>lr) & (array<ur))
new_array
[49]: array([38, 16, 55, 79, 35, 60, 47, 78, 18, 42, 89, 91, 43, 70, 38, 64])
[50]: sns.distplot(new_array)
[50]: <seaborn.axisgrid.FacetGrid at 0x1ee66838d00>
```



```
[51]: lr1,ur1=outDetection(new_array)
lr1,ur1
[51]: (-13.0, 123.0)
[52]: final_array=new_array[(new_array>lr1) & (new_array<ur1)]
final_array
[52]: array([38, 16, 55, 79, 35, 60, 47, 78, 18, 42, 89, 91, 43, 70, 38, 64])
```

Exp:8. a Experiment to understand feature scaling.

Description: Understand the importance of feature scaling

```
[ ]: import numpy as np
import pandas as pd
df= pd.read_csv('pre-process_datasample.csv')
df
```

```
[55]:   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 NaN     50.0  83000.0      No
  9 France  37.0  67000.0     Yes
```

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

```
[56]:   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
```

```
[ ]: labels=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]])
```

```
[60]: * SimpleImputer
SimpleImputer()
```

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

```
[61]: * SimpleImputer
SimpleImputer()
```

```
[ ]: SimpleImputer()
[62]: * SimpleImputer
SimpleImputer()

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

[63]: 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],
           ['France', 50.0, 83000.0],
           ['France', 37.0, 67000.0]], dtype=object)

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

[64]: 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.],
           [1., 0., 0.],
           [1., 0., 0.]))

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

[65]: 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],
           [1.0, 0.0, 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

[66]: array([[ 1.00000000e+00, -5.00000000e-01, -6.54653671e-01,
             7.58874362e-01, 7.49473254e-01],
            [-1.00000000e+00, -5.00000000e-01, 1.52752523e+00,
             -1.71150388e+00, -1.43817841e+00],
            [-1.00000000e+00, 2.00000000e+00, -6.54653671e-01,
             -1.27555478e+00, -8.91265492e-01],
            [-1.00000000e+00, -5.00000000e-01, 1.52752523e+00,
             -1.13023841e-01, -2.53200424e-01],
            [-1.00000000e+00, 2.00000000e+00, -6.54653671e-01,
             1.77608893e-01, 6.63219199e-16],
            [ 1.00000000e+00, -5.00000000e-01, -6.54653671e-01,
             -5.48972942e-01, -5.26656882e-01],
            [-1.00000000e+00, -5.00000000e-01, 1.52752523e+00,
             0.00000000e+00, -1.07356980e+00],
            [ 1.00000000e+00, -5.00000000e-01, -6.54653671e-01,
             1.34013983e+00, 1.38753832e+00],
            [ 1.00000000e+00, -5.00000000e-01, -6.54653671e-01,
             1.63077256e+00, 1.75214693e+00],
            [ 1.00000000e+00, -5.00000000e-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

[67]: 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, 0.45879365],
           [1.          , 0.          , 0.          , 0.34782609, 0.28571429],
           [0.          , 0.          , 1.          , 0.51207729, 0.11428571],
           [1.          , 0.          , 0.          , 0.91304348, 0.88571429],
           [1.          , 0.          , 0.          , 1.          , 1.          ],
           ...])

```

Exp:8. b Experiment to understand the data preprocessing in Data science

Description: Understand the importance of Data preprocessing in data science

```
[1]: import numpy as np
import pandas as pd
df=pd.read_csv("pre-process_datasample.csv")
df
```

```
[69]:
```

	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		50.0	83000.0	No
9	France	37.0	67000.0	Yes

```
[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     9 non-null      object 
 1   Age         9 non-null      float64 
 2   Salary       9 non-null      float64 
 3   Purchased    10 non-null    object  
dtypes: float64(2), object(2)
memory usage: 448.0+ bytes
```

```
[1]: df.Country.mode()
```

```
[71]: 0    France
Name: Country, dtype: object
```

```
[1]: df.Country.mode()[0]
```

```
[72]: 'France'
```

```
[1]: type(df.Country.mode())
```

```
[73]: pandas.core.series.Series
```

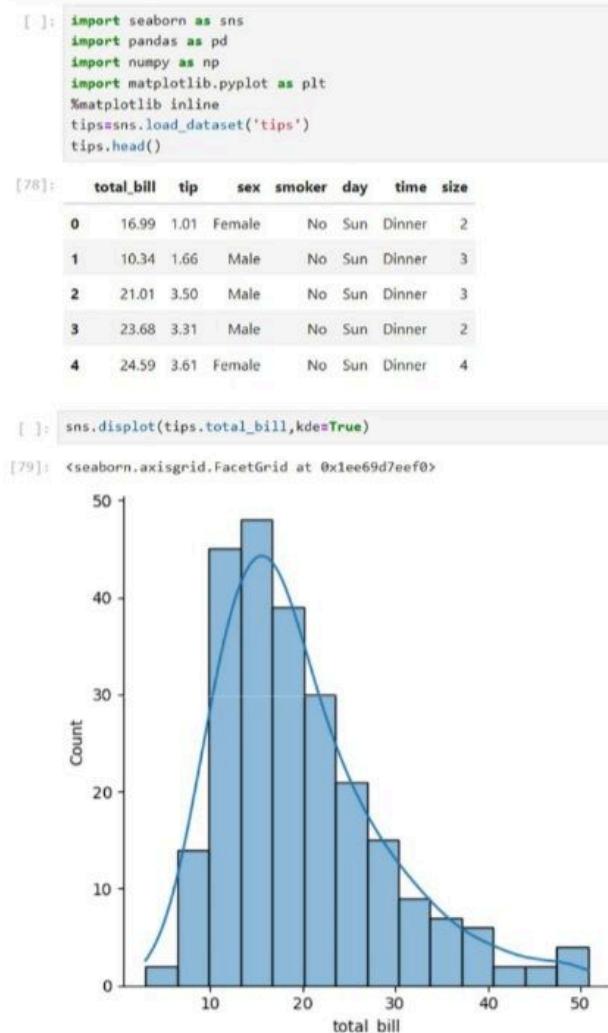
```
[1]: df.Country.fillna(df.Country.mode()[0],inplace=True)
df.Age.fillna(df.Age.median(),inplace=True)
df.Salary.fillna(round(df.Salary.mean()),inplace=True)
df
```

```
[74]:
```

	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	63778.0	Yes
5	France	35.0	58000.0	Yes
6	Spain	38.0	52000.0	No
7	France	48.0	79000.0	Yes
8	France	50.0	83000.0	No
9	France	37.0	67000.0	Yes

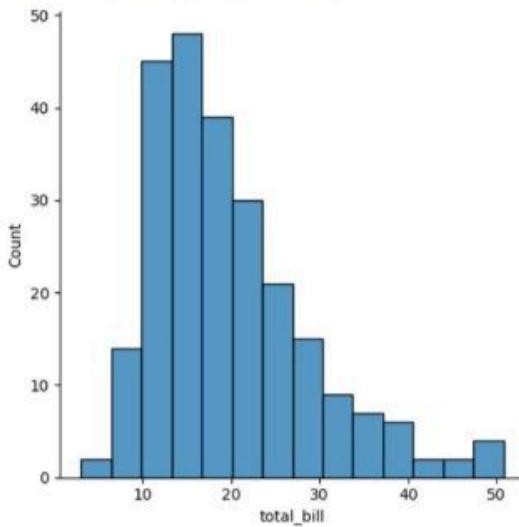
Exp No:9 Experiment to understand EDA-Quantitative and Qualitative analysis.

Description: Understand the importance of EDA-Quantitative and Qualitative analysis.



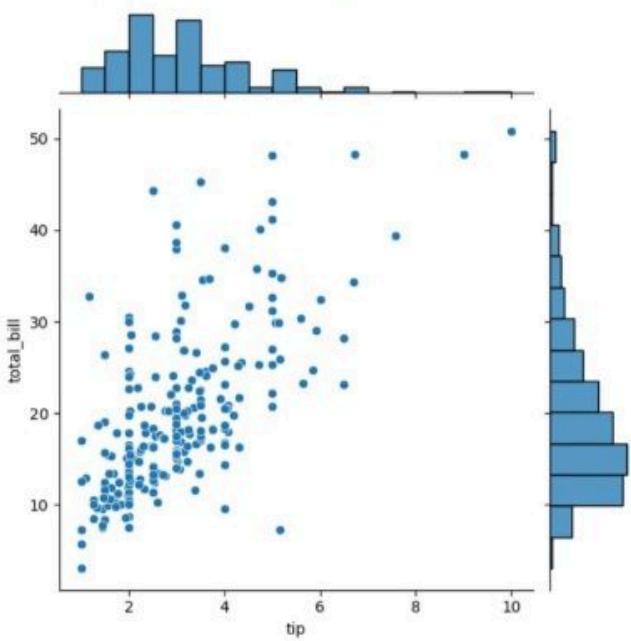
```
[1]: sns.distplot(tips.total_bill,kde=False)
```

```
[80]: <seaborn.axisgrid.FacetGrid at 0x1ee6adfb00>
```



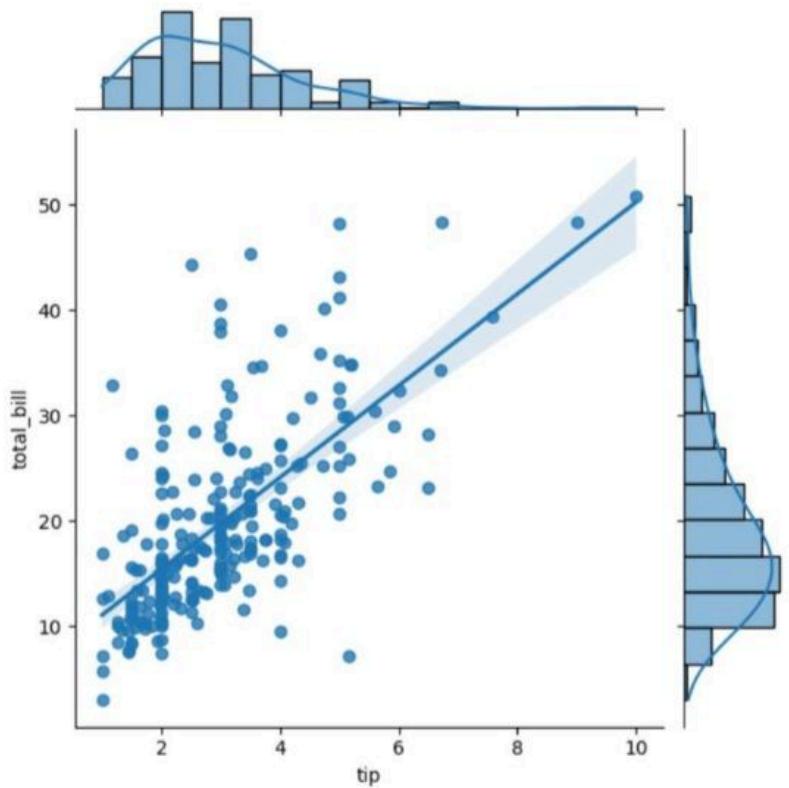
```
[1]: sns.jointplot(x=tips.tip,y=tips.total_bill)
```

```
[81]: <seaborn.axisgrid.JointGrid at 0x1ee6b2ed210>
```



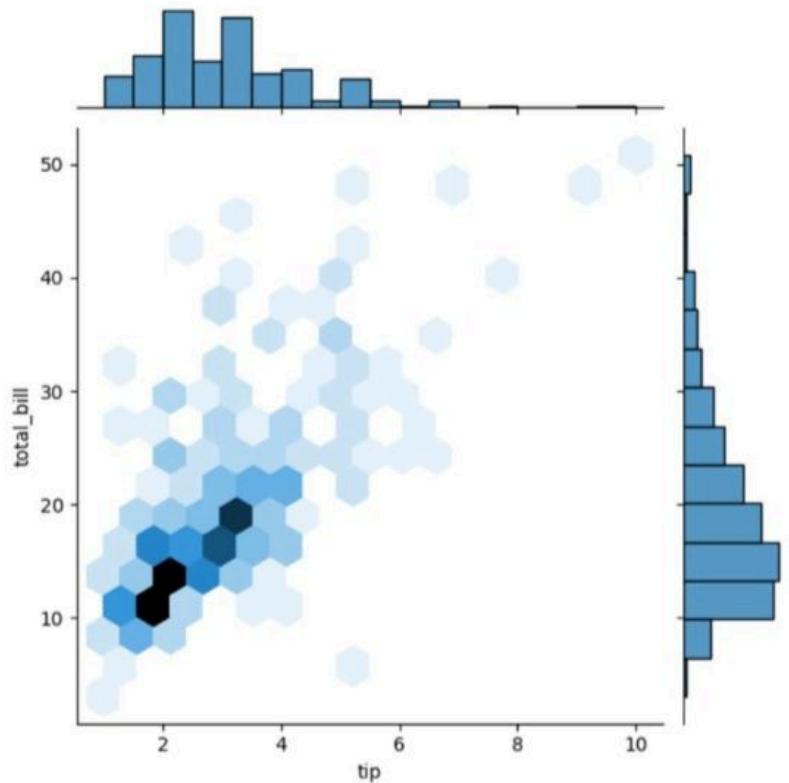
```
[ ]: sns.jointplot(x=tips.tip,y=tips.total_bill,kind="reg")
```

```
[82]: <seaborn.axisgrid.JointGrid at 0x1ee6b3f1120>
```



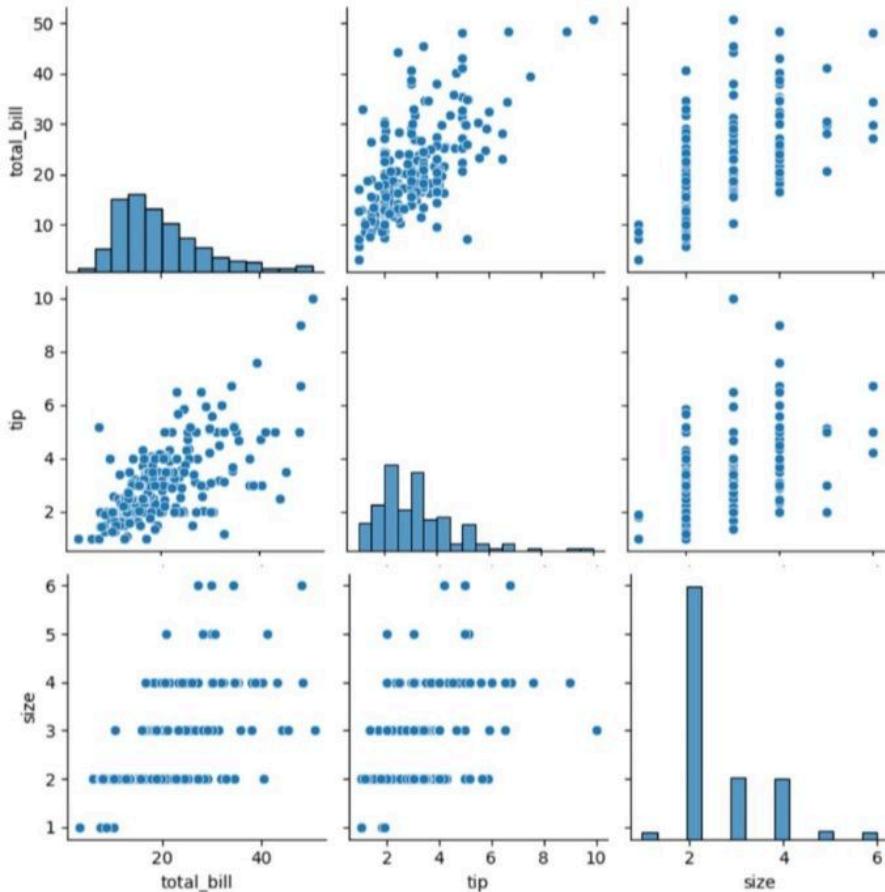
```
[ ]: sns.jointplot(x=tips.tip,y=tips.total_bill,kind="hex")
```

```
[83]: <seaborn.axisgrid.JointGrid at 0x1ee6b809600>
```



```
[ ]: sns.pairplot(tips)
```

```
[84]: <seaborn.axisgrid.PairGrid at 0x1ee6b4060e0>
```

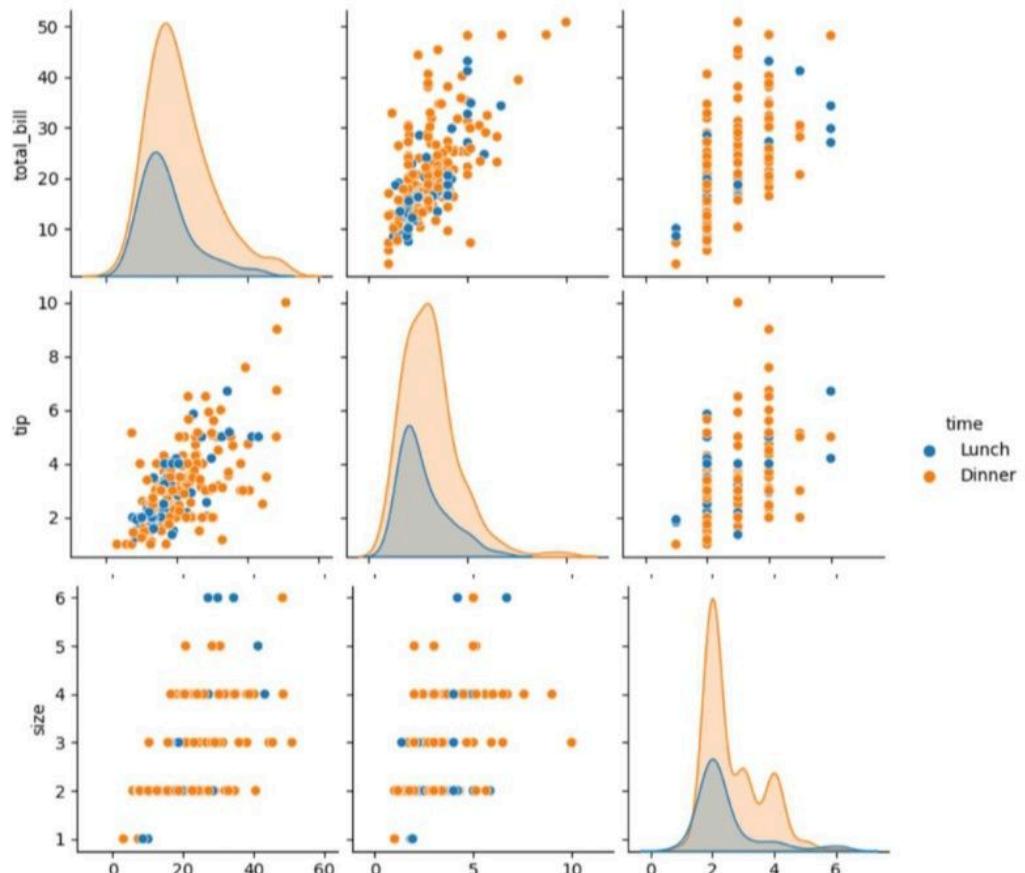


```
[ ]: tips.time.value_counts()
```

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

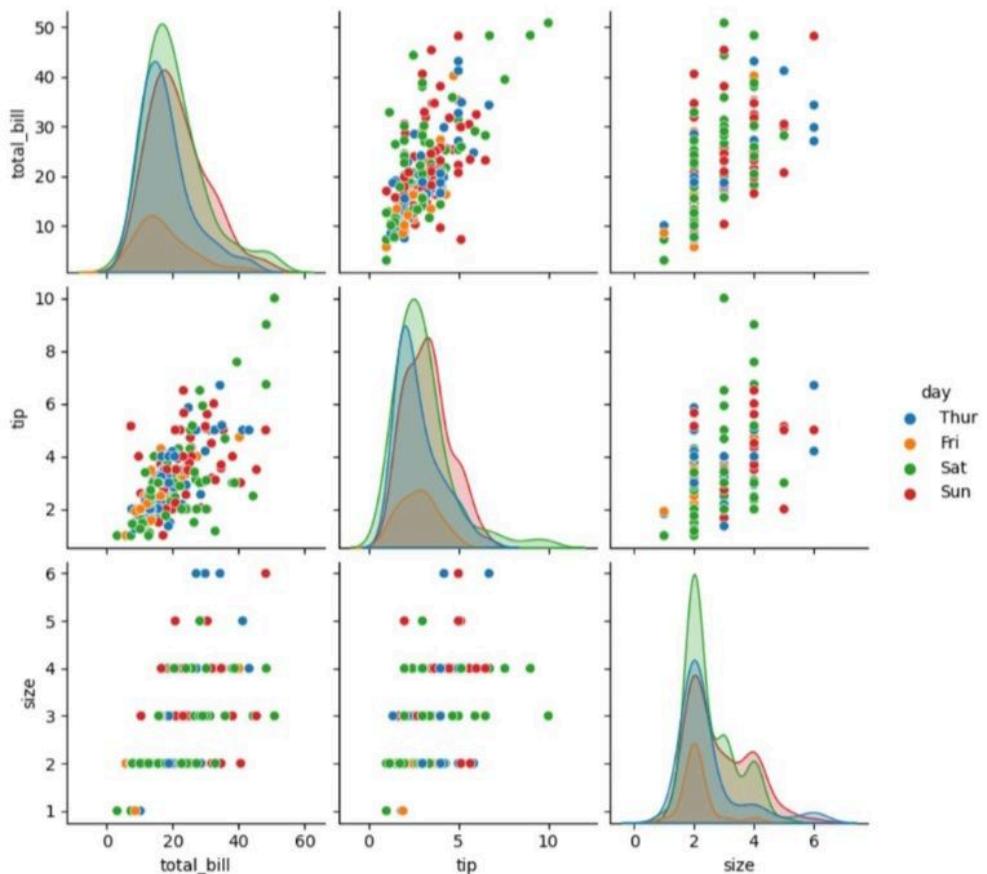
```
[ ]: sns.pairplot(tips,hue='time')
```

```
[86]: <seaborn.axisgrid.PairGrid at 0x1ee6c671870>
```



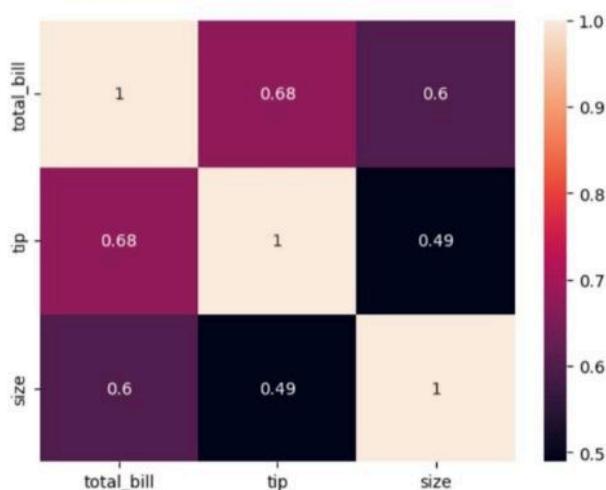
```
[1]: sns.pairplot(tips,hue='day')
```

```
[87]: <seaborn.axisgrid.PairGrid at 0x1ee6cf26ce0>
```

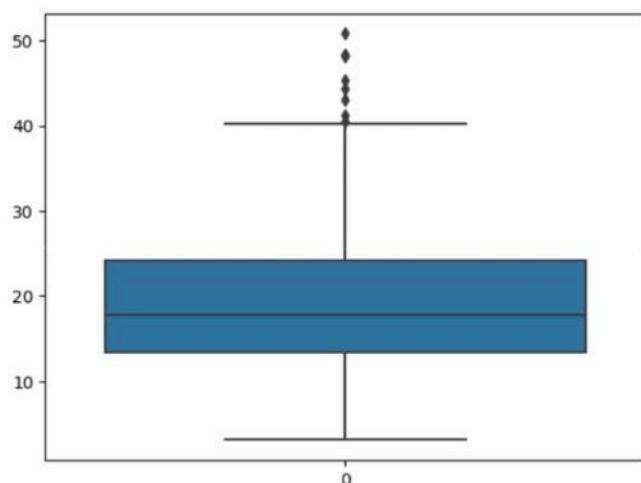


```
[1]: sns.heatmap(tips.corr(numeric_only=True),annot=True)
```

```
[88]: <Axes: >
```

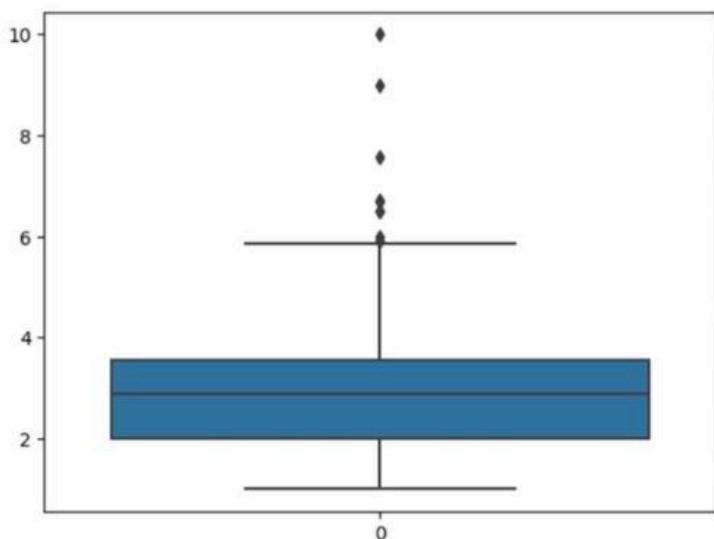


```
[1]: sns.boxplot(tips.total_bill)
```



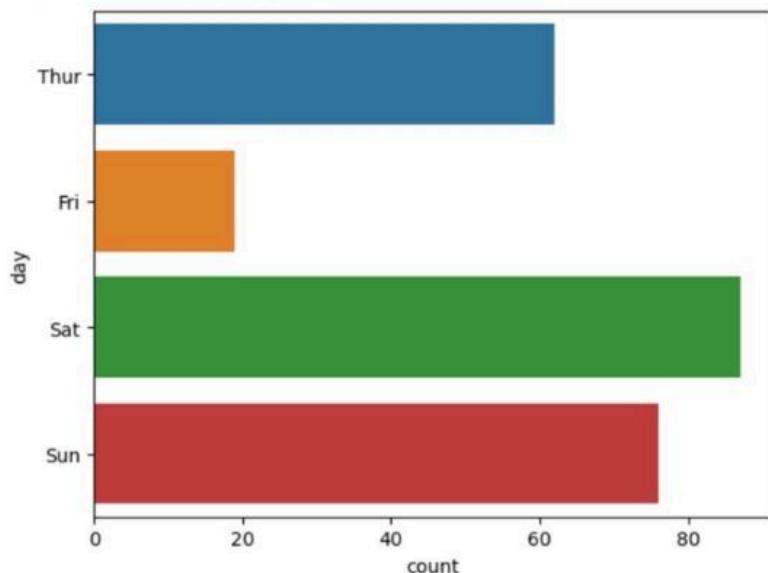
```
[ ]: sns.boxplot(tips.tip)
```

```
[90]: <Axes: >
```



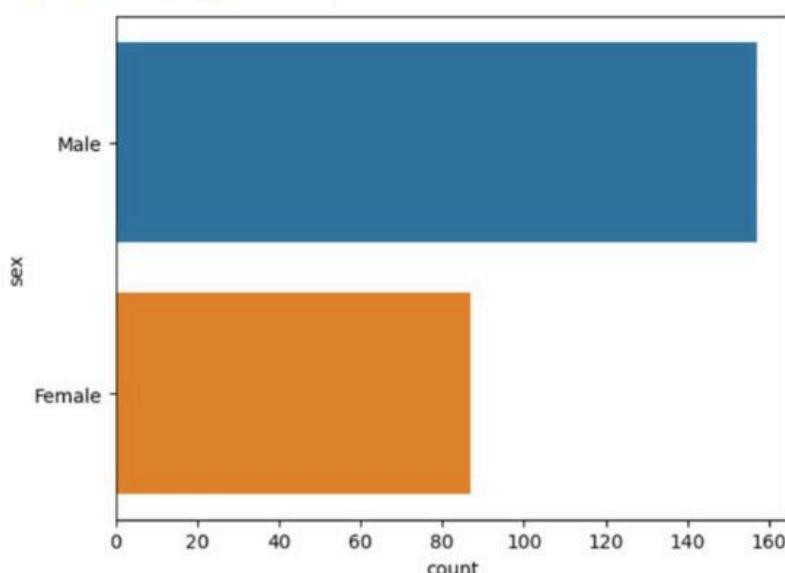
```
[ ]: sns.countplot(data=tips,y="day")
```

```
[91]: <Axes: xlabel='count', ylabel='day'>
```



```
[ ]: sns.countplot(data=tips,y="sex")
```

```
[92]: <Axes: xlabel='count', ylabel='sex'>
```



Exp:10 Regression

```
[ ]: import numpy as np
      import pandas as pd
      df=pd.read_csv('Salary_data.csv')
      df
```

```
[97]:
```

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891
5	2.9	56642
6	3.0	60150
7	3.2	54445
8	3.2	64445
9	3.7	57189
10	3.9	63218
11	4.0	55794
12	4.0	56957
13	4.1	57081
14	4.5	61111
15	4.9	67938
16	5.1	66029
17	5.3	83088
18	5.9	81363
19	6.0	93940
20	6.8	91738
21	7.1	98273
22	7.9	101302
23	8.2	113812
24	8.7	109431
25	9.0	105582
26	9.5	116969
27	9.6	112635
28	10.3	122391
29	10.5	121872

```
[ ]: 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: 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   int64   
dtypes: float64(1), int64(1)
memory usage: 608.0 bytes
```

```
[ ]: df.describe()

[101]:
```

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

[ ]: from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)

[104]: * LinearRegression
        LinearRegression()

[ ]: model.score(x_train,y_train)

[105]: 0.9608732656731791

[ ]: model.score(x_test,y_test)

[106]: 0.9387922566107753

[ ]: model.coef_

[107]: array([[9373.93124487]])

[ ]: model.intercept_

[108]: array([26116.42617326])

[ ]: 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 [[438569.40094767]]:
```

Exp:11 Logistic Regression

```
[ ]: import numpy as np
import pandas as pd
df=pd.read_csv('Social_Network_Ads.csv')
df
```

```
[114]:
```

	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()
```

```
[115]:
```

	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
```

```
[116]: 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],
```

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

(): label

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

```
[1]: for i in range(1,401):
    x_train,x_test,y_train,y_test=train_test_split(features,label,test_size=0.1)
    model=LogisticRegression()
    model.fit(x_train,y_train)
    train_score=model.score(x_train,y_train)
    test_score=model.score(x_test,y_test)
    if test_score>train_score:
        print("Test () Train Random State {}".format(test_score,train_score,i))
```

```
Test 0.675 Train0.6388888888888888 Random State 1
Test 0.775 Train0.6277777777777778 Random State 2
Test 0.725 Train0.6333333333333333 Random State 4
Test 0.725 Train0.6333333333333333 Random State 7
Test 0.65 Train0.64166666666666667 Random State 8
Test 0.7 Train0.6361111111111111 Random State 9
Test 0.725 Train0.6333333333333333 Random State 10
Test 0.675 Train0.6388888888888888 Random State 12
Test 0.675 Train0.6388888888888888 Random State 13
Test 0.7 Train0.6361111111111111 Random State 16
Test 0.675 Train0.6388888888888888 Random State 18
Test 0.675 Train0.6388888888888888 Random State 19
Test 0.8 Train0.625 Random State 20
Test 0.675 Train0.6388888888888888 Random State 21
Test 0.65 Train0.64166666666666667 Random State 22
Test 0.675 Train0.6388888888888888 Random State 24
Test 0.65 Train0.64166666666666667 Random State 26
Test 0.65 Train0.64166666666666667 Random State 27
Test 0.7 Train0.6361111111111111 Random State 31
Test 0.725 Train0.6333333333333333 Random State 34
Test 0.725 Train0.6333333333333333 Random State 36
Test 0.75 Train0.6305555555555555 Random State 38
Test 0.65 Train0.64166666666666667 Random State 42
Test 0.75 Train0.6305555555555555 Random State 44
Test 0.675 Train0.6388888888888888 Random State 45
Test 0.65 Train0.64166666666666667 Random State 48
Test 0.725 Train0.6333333333333333 Random State 49
Test 0.775 Train0.6277777777777778 Random State 50
Test 0.8 Train0.625 Random State 51
Test 0.725 Train0.6333333333333333 Random State 56
Test 0.675 Train0.6388888888888888 Random State 57
Test 0.65 Train0.64166666666666667 Random State 58
Test 0.65 Train0.64166666666666667 Random State 59
Test 0.7 Train0.6361111111111111 Random State 64
Test 0.675 Train0.6388888888888888 Random State 67
Test 0.775 Train0.6277777777777778 Random State 68
Test 0.675 Train0.6388888888888888 Random State 71
Test 0.65 Train0.64166666666666667 Random State 72
Test 0.65 Train0.64166666666666667 Random State 76
Test 0.65 Train0.64166666666666667 Random State 77
Test 0.7 Train0.6361111111111111 Random State 82
Test 0.65 Train0.64166666666666667 Random State 83
```

```
Test 0.675 Train0.6388888888888888 Random State 373
Test 0.675 Train0.6388888888888888 Random State 377
Test 0.675 Train0.6388888888888888 Random State 378
Test 0.65 Train0.6416666666666667 Random State 382
Test 0.7 Train0.6361111111111111 Random State 383
Test 0.65 Train0.6416666666666667 Random State 384
Test 0.725 Train0.6333333333333333 Random State 386
Test 0.75 Train0.6305555555555555 Random State 388
Test 0.65 Train0.6416666666666667 Random State 390
Test 0.7 Train0.6361111111111111 Random State 391
Test 0.7 Train0.6361111111111111 Random State 394
Test 0.7 Train0.6361111111111111 Random State 397
```

```
[ ]: 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)
```

```
120]: LogisticRegression
LogisticRegression()
```

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

```
0.628125
0.7
```

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

	precision	recall	f1-score	support
0	0.64	1.00	0.78	257
1	0.00	0.00	0.00	143
accuracy			0.64	400
macro avg	0.32	0.50	0.39	400
weighted avg	0.41	0.64	0.50	400

Exp:12 KNN

```
[1]: import numpy as np
import pandas as pd
df=pd.read_csv('Iris.csv')
df.info()

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

[2]: df.variety.value_counts()

[125]: Setosa      50
Versicolor  50
Virginica   50
Name: variety, dtype: int64

[3]: df.head()

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

[4]: features=df.iloc[:, :-1].values
label=df.iloc[:, -1].values
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
xtrain,xtest,ytrain,ytest=train_test_split(features,label,test_size=.2)
model_KNN=KNeighborsClassifier(n_neighbors=5)
model_KNN.fit(xtrain,ytrain)

[127]: * KNeighborsClassifier
KNeighborsClassifier()

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

0.9666666666666667
0.9666666666666667

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

[129]: array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  2, 48]], dtype=int64)

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

precision    recall  f1-score   support

 Setosa      1.00      1.00      1.00      50
 Versicolor   0.96      0.94      0.95      50
 Virginica   0.94      0.96      0.95      50

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

Exp:13 K-Means

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

[ 2]: df=pd.read_csv('Mall_Customers.csv')

[ 3]: df.info()

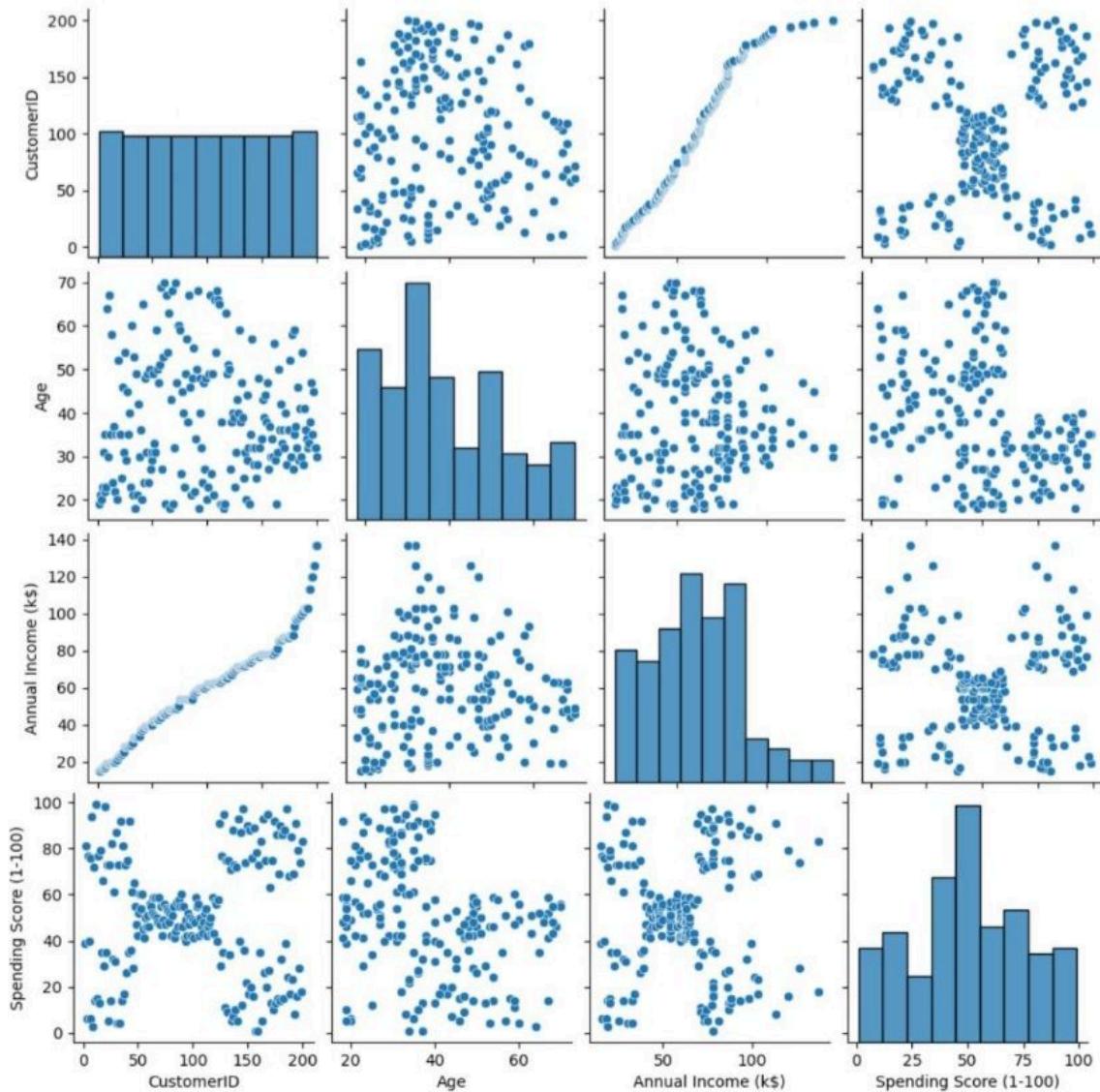
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   CustomerID  200 non-null    int64  
 1   Gender       200 non-null    object  
 2   Age          200 non-null    int64  
 3   Annual Income (k$) 200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

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

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

```
[ 5]: sns.pairplot(df)
```

```
[136]: <seaborn.axisgrid.PairGrid at 0x1ee6f13f2b0>
```



Exp:14 Testing

```
[145]: #T-TEST
import numpy as np
from scipy import stats
marks = np.array([72, 68, 75, 70, 74, 69, 71, 73, 70, 72])
mu_0 = 70
t_stat, p_value = stats.ttest_isamp(marks, mu_0)
print(f'T-statistic: {t_stat:.3f}')
print(f'P-value: {p_value:.4f}')
alpha = 0.05
if p_value < alpha:
    print('Reject Null Hypothesis. Mean is significantly different from 70.')
else:
    print('Fail to Reject Null Hypothesis. No significant difference')

T-statistic: 1.993
P-value: 0.0774
Fail to Reject Null Hypothesis. No significant difference
```

```
[146]: #Z-TEST
import numpy as np
from math import sqrt
from scipy.stats import norm
x_bar = 51.2
mu_0 = 50
sigma = 3
n = 36
z_stat = (x_bar - mu_0) / (sigma / sqrt(n))
p_value = 2 * (1 - norm.cdf(abs(z_stat)))
print(f'Z-statistic: {z_stat:.3f}')
print(f'P-value: {p_value:.4f}')
alpha = 0.05
if p_value < alpha:
    print('Reject Null Hypothesis. Mean is significantly different from 50 g.')
else:
    print('Fail to Reject Null Hypothesis. No significant difference.')

Z-statistic: 2.400
P-value: 0.0164
Reject Null Hypothesis. Mean is significantly different from 50 g.
```

```
[147]: #anova test
[148]: import numpy as np
from scipy import stats
A = [20, 22, 23]
B = [19, 20, 18]
C = [25, 27, 26]
f_stat, p_value = stats.f_oneway(A, B, C)
print(f'F-statistic: {f_stat:.3f}')
print(f'P-value: {p_value:.4f}')
alpha = 0.05
if p_value < alpha:
    print('Reject Null Hypothesis. Means are significantly different.')
else:
    print('Fail to Reject Null Hypothesis. No significant difference')

F-statistic: 25.923
P-value: 0.0011
Reject Null Hypothesis. Means are significantly different.
```