



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 :Dhanush Kumar.G

Register Number :240701113

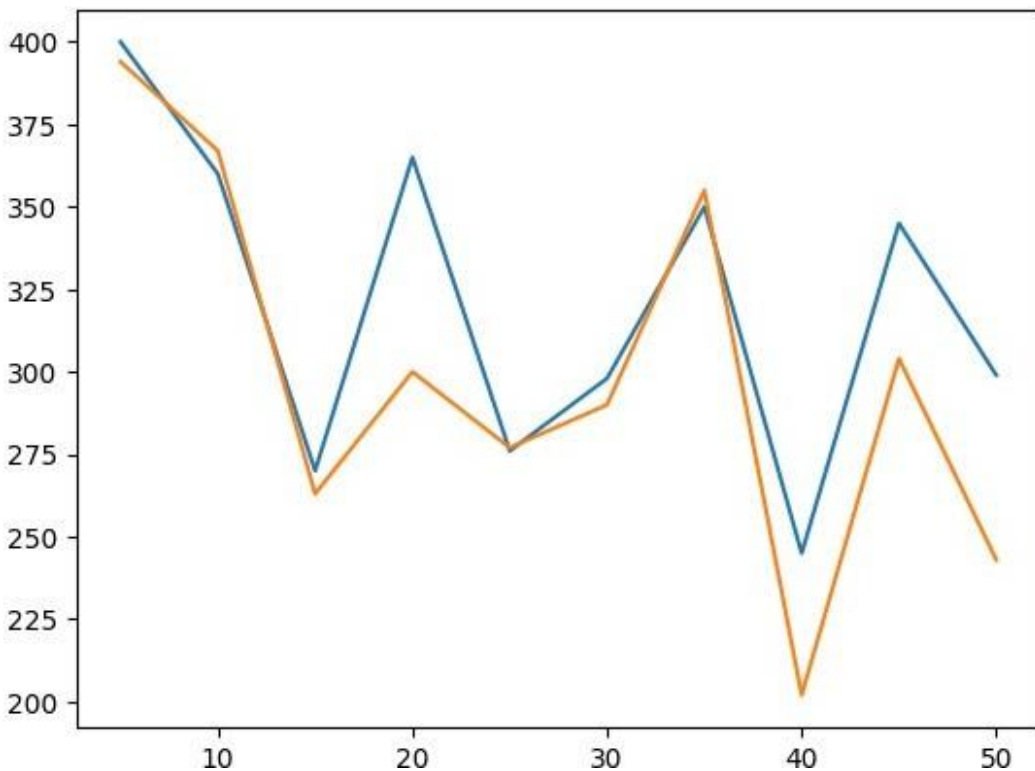
#240701542

#Dhanush

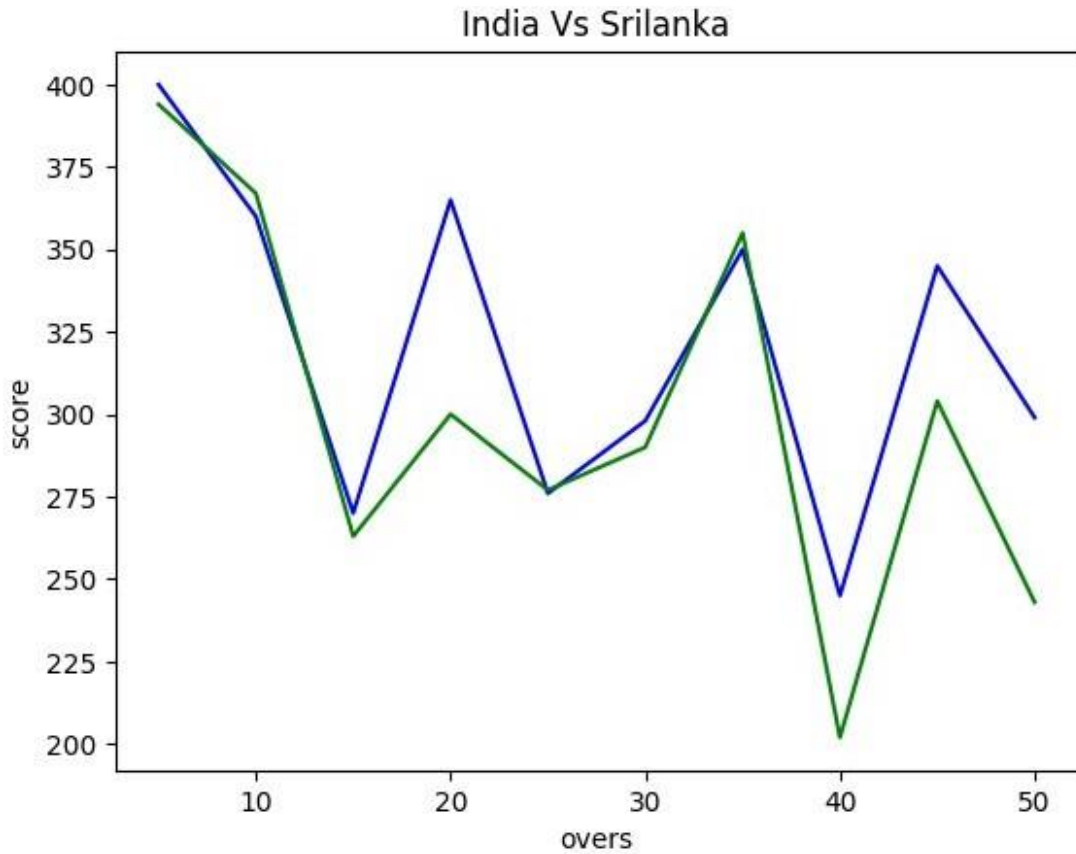
Kumar.G

#17-07-25

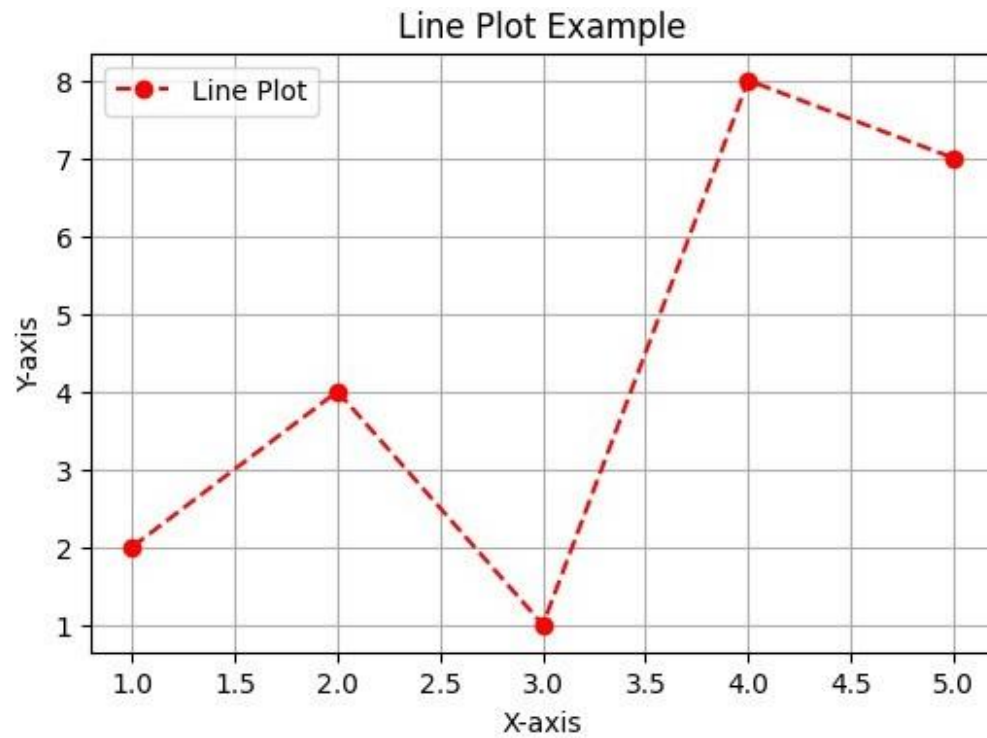
```
import matplotlib.pyplot as plt
overs=list(range(5,51,5))
India_Score=[400,360,270,365,276,298,350,245,345,299]
Srilanka_Score=[394,367,263,300,277,290,355,202,304,243]
plt.plot(overs,India_Score)
plt.plot(overs,Srilanka_Score)
plt.show()
plt.title("India Vs Srilanka")
plt.xlabel("overs")
plt.ylabel("score")
plt.plot(overs,India_Score,color="blue",label="India")
plt.plot(overs,Srilanka_Score,color="green",label="Srilanka")
```



[<matplotlib.lines.Line2D at 0x191bd6fd720>]



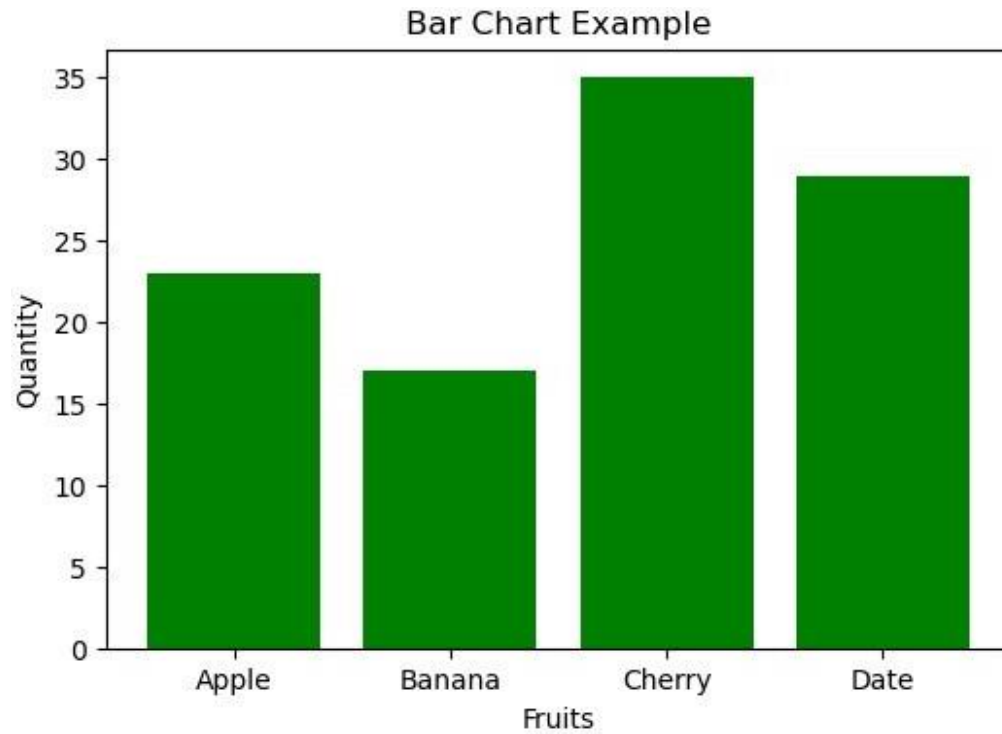
```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [2, 4, 1, 8, 7]
plt.figure(figsize=(6, 4))
plt.plot(x, y, color='red', marker='o', linestyle='--', label='Line Plot')
plt.title("Line Plot Example")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.grid(True)
plt.show()
```



```
categories = ['Apple', 'Banana', 'Cherry', 'Date']
```

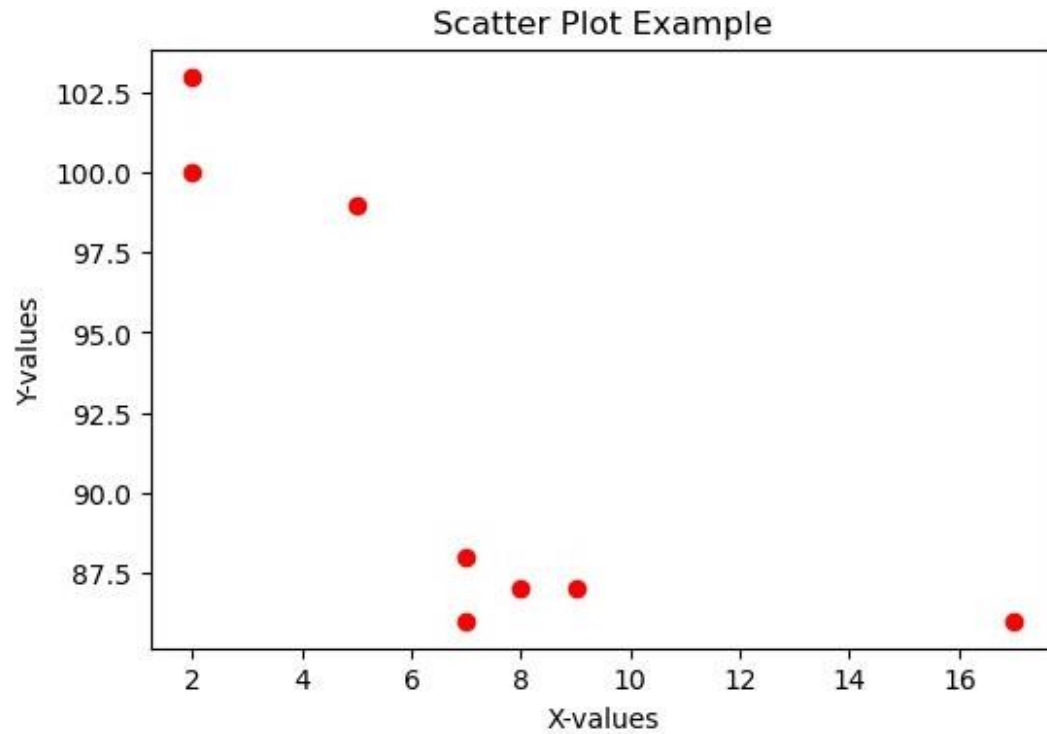
```
values = [23, 17, 35, 29]
```

```
plt.figure(figsize=(6, 4))  
plt.bar(categories, values, color='green')  
plt.title("Bar Chart Example")  
plt.xlabel("Fruits")  
plt.ylabel("Quantity")  
plt.show()
```

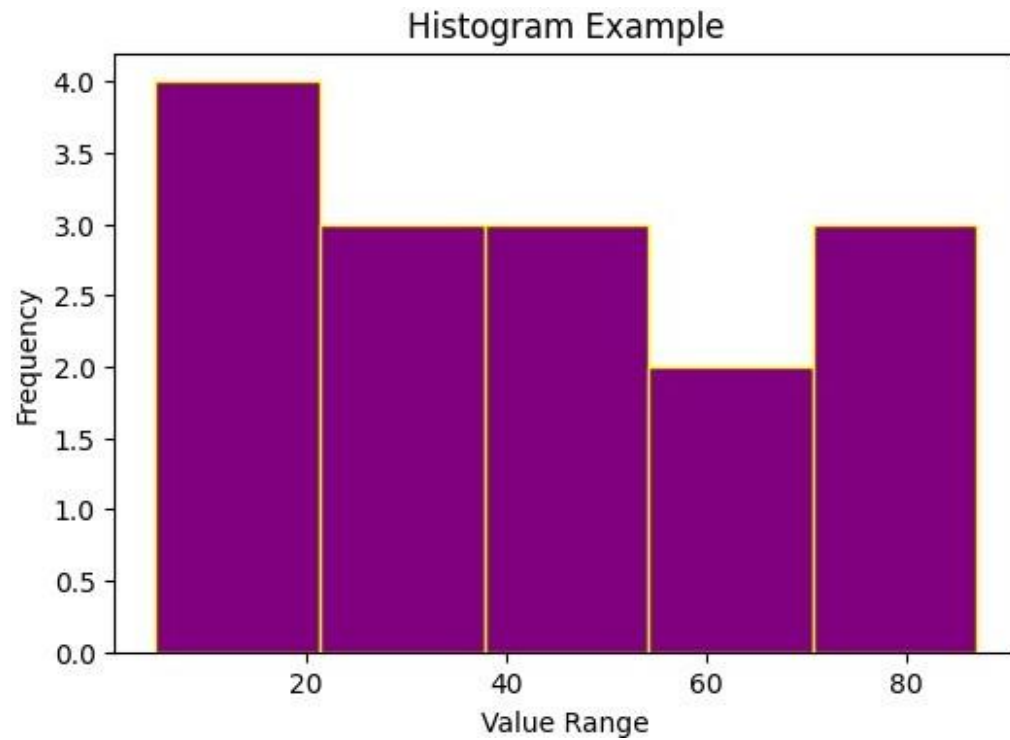


```
x_scatter = [5, 7, 8, 7, 2, 17, 2, 9]
y_scatter = [99, 86, 87, 88, 100, 86, 103, 87]
plt.figure(figsize=(6, 4))
plt.scatter(x_scatter, y_scatter, color='red')
plt.title("Scatter Plot Example")
plt.xlabel("X-values")
plt.ylabel("Y-values")

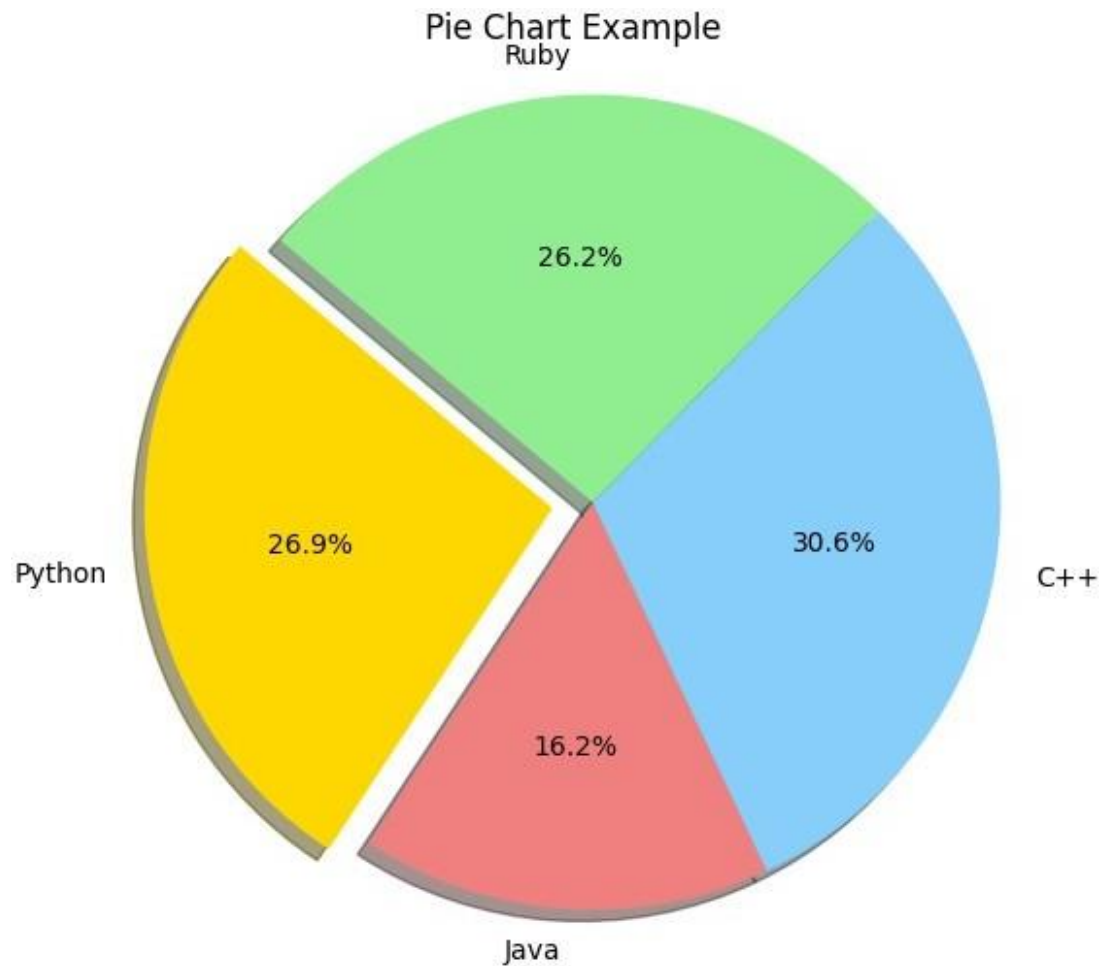
plt.show()
```



```
data = [22, 87, 5, 43, 56, 73, 55, 54, 11, 20, 51, 5, 79, 31, 27]
plt.figure(figsize=(6, 4))
plt.hist(data, bins=5, color='purple', edgecolor='yellow')
plt.title("Histogram Example")
plt.xlabel("Value Range")
plt.ylabel("Frequency")
plt.show()
```



```
labels = ['Python', 'Java', 'C++', 'Ruby']
sizes = [215, 130, 245, 210]
colors = ['gold', 'lightcoral', 'lightskyblue', 'lightgreen']
explode = (0.1, 0, 0, 0)
plt.figure(figsize=(6, 6))
plt.pie(sizes, explode=explode, labels=labels,
        colors=colors, autopct='%1.1f%%',
        shadow=True, startangle=140)
plt.title("Pie Chart Example")
plt.axis('equal')
plt.show()
```



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

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

print(df.head())

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

df['Sales'] = df['Sales'].fillna(df['Sales'].mean())

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

df
```

	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

```

```

      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
5  06-01-2023  Product A    210         5  North
6  07-01-2023  Product C    320         7   East
7  08-01-2023  Product B    160         3  South
8  09-01-2023  Product A    230         6  North
9  10-01-2023  Product C    310         7   East
10 11-01-2023  Product B    190         4   West
11 12-01-2023  Product A    240         6  North
12 13-01-2023  Product C    330         8   East
13 14-01-2023  Product B    170         3  South
14 15-01-2023  Product A    250         7  North
15 16-01-2023  Product C    340         8   East

```

```

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

```

```

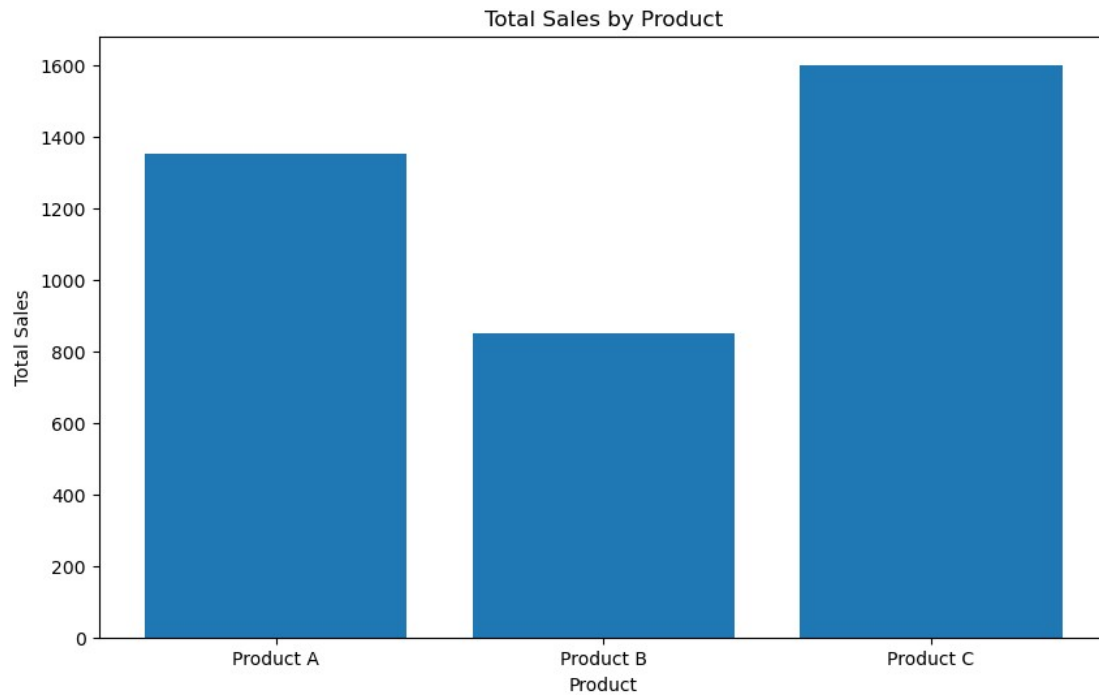
      Product  Sales  Quantity
0  Product A   1350         33
1  Product B    850         17
2  Product C   1600         36

```

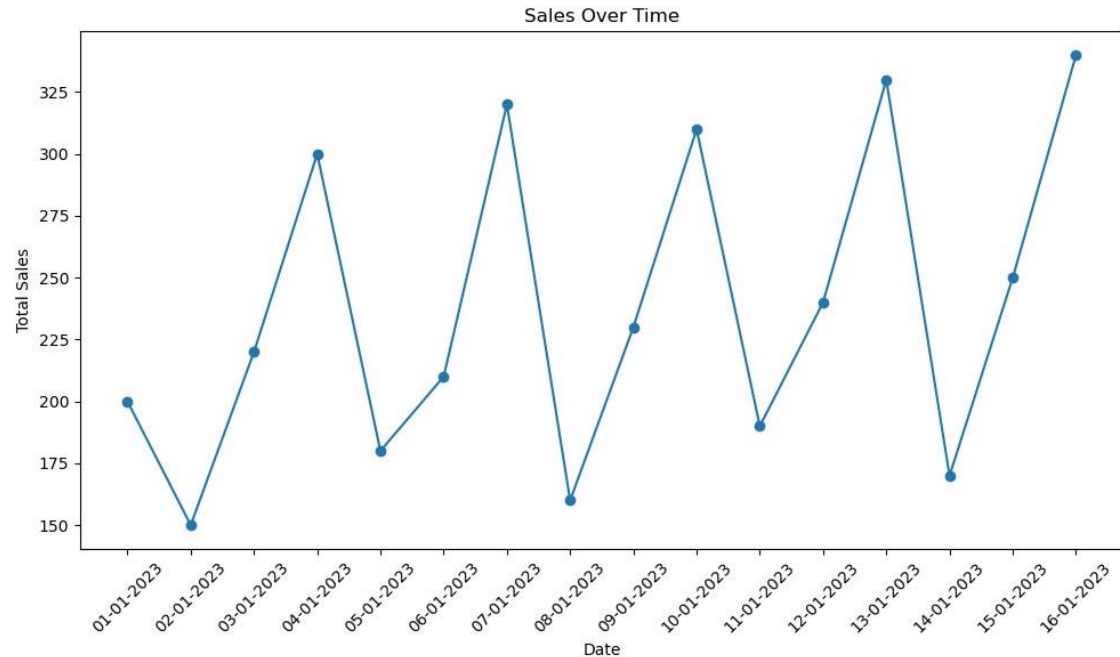
```

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 = df.dropna(subset=['Date'])
sales_over_time = df.groupby('Date', as_index=False)['Sales'].sum()
plt.figure(figsize=(10, 6))
plt.plot(sales_over_time['Date'], sales_over_time['Sales'], marker='o')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.title('Sales Over Time')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



```

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

print(pivot_table)

# Only correlate numeric columns
correlation_matrix = df.select_dtypes(include=['number']).corr()
print(correlation_matrix)

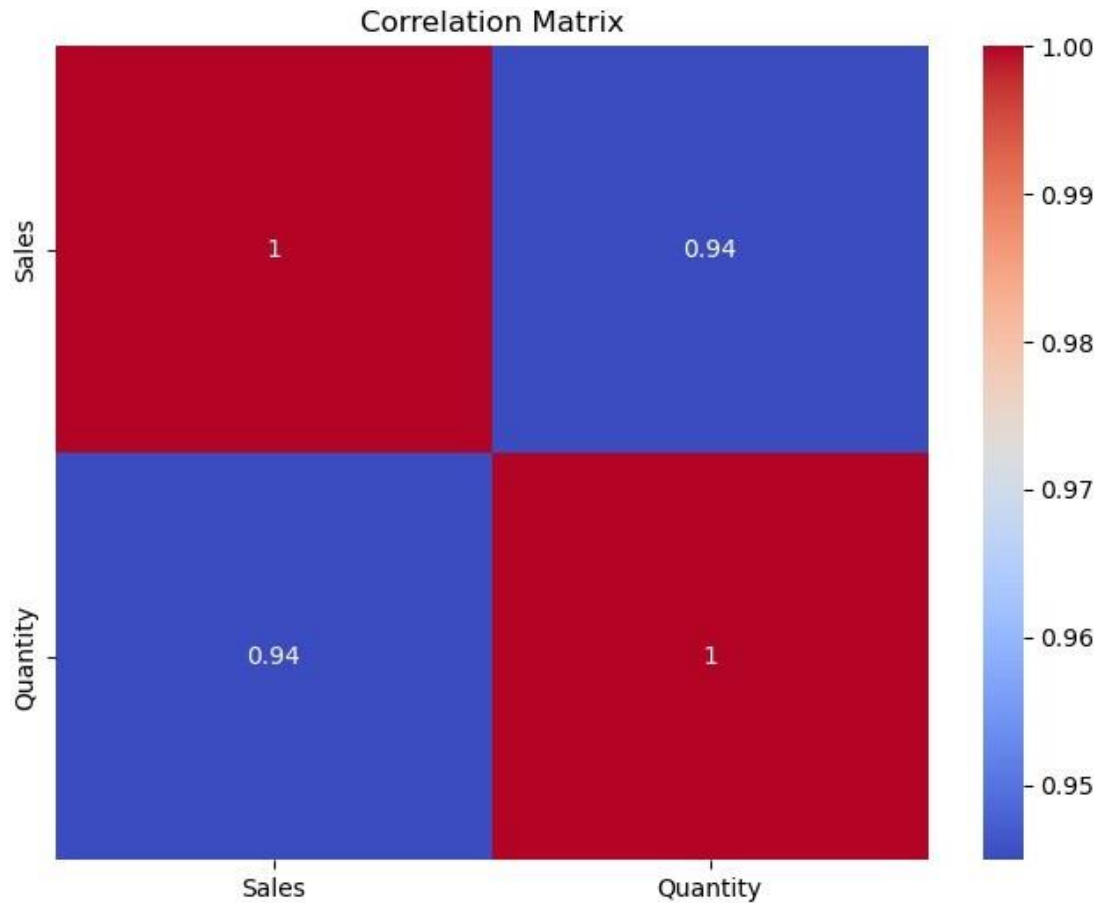
```

	Product A	Product B	Product C
Region			
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	

```

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

```



```
import numpy as np
import pandas as pd
df=pd.read_csv("Hotel_Dataset.csv")
df
```

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	\
0	1	20-25	4	Ibis	veg	1300	
1	2	30-35	5	LemonTree	Non-Veg	2000	
2	3	25-30	6	RedFox	Veg	1322	
3	4	20-25	-1	LemonTree	Veg	1234	
4	5	35+	3	Ibis	Vegetarian	989	
5	6	35+	3	Ibys	Non-Veg	1909	
6	7	35+	4	RedFox	Vegetarian	1000	
7	8	20-25	7	LemonTree	Veg	2999	
8	9	25-30	2	Ibis	Non-Veg	3456	
9	9	25-30	2	Ibis	Non-Veg	3456	
10	10	30-35	5	RedFox	non-Veg	-6755	

	NoOfPax	EstimatedSalary	Age_Group.1
0	2	40000	20-25

1	3	59000	30-35
2	2	30000	25-30
3	2	120000	20-25
4	2	45000	35+
5	2	122220	35+
6	-1	21122	35+
7	-10	345673	20-25
8	3	-99999	25-30
9	3	-99999	25-30
10	4	87777	30-35

```
df.duplicated()
```

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

```
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
```

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

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	\
0	1	20-25	4	Ibis	veg	1300	
1	2	30-35	5	LemonTree	Non-Veg	2000	

2	3	25-30	6	RedFox	Veg	1322
3	4	20-25	-1	LemonTree	Veg	1234
4	5	35+	3	Ibis	Vegetarian	989
5	6	35+	3	Ibys	Non-Veg	1909
6	7	35+	4	RedFox	Vegetarian	1000
7	8	20-25	7	LemonTree	Veg	2999
8	9	25-30	2	Ibis	Non-Veg	3456
10	10	30-35	5	RedFox	non-Veg	-6755

	NoOfPax	EstimatedSalary	Age_Group.1
0	2	40000	20-25
1	3	59000	30-35
2	2	30000	25-30
3	2	120000	20-25
4	2	45000	35+
5	2	122220	35+
6	-1	21122	35+
7	-10	345673	20-25
8	3	-99999	25-30
10	4	87777	30-35

```
len(df)
```

```
10
```

```
index=np.array(list(range(0,len(df))))
df.set_index(index,inplace=True)
index
```

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

```
df
```

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

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

```

4          45000          35+
5          122220          35+
6           21122          35+
7          345673          20-25
8          -99999          25-30
9           87777          30-35

```

```

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

```

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax
\							
0	1	20-25	4	Ibis	veg	1300	2
1	2	30-35	5	LemonTree	Non-Veg	2000	3
2	3	25-30	6	RedFox	Veg	1322	2
3	4	20-25	-1	LemonTree	Veg	1234	2
4	5	35+	3	Ibis	Vegetarian	989	2
5	6	35+	3	Ibys	Non-Veg	1909	2
6	7	35+	4	RedFox	Vegetarian	1000	-1
7	8	20-25	7	LemonTree	Veg	2999	-10
8	9	25-30	2	Ibis	Non-Veg	3456	3
9	10	30-35	5	RedFox	non-Veg	-6755	4

```

EstimatedSalary
0          40000
1          59000
2          30000
3         120000
4          45000
5         122220
6          21122
7         345673
8         -99999
9          87777

```

```

df.loc[df['CustomerID'] < 0, 'CustomerID'] = np.nan
df.loc[df['Bill'] < 0, 'Bill'] = np.nan
df.loc[df['EstimatedSalary'] < 0, 'EstimatedSalary'] = np.nan
df

```

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

	NoOfPax	EstimatedSalary
0	2	40000.0
1	3	59000.0
2	2	30000.0
3	2	120000.0
4	2	45000.0
5	2	122220.0
6	-1	21122.0
7	-10	345673.0
8	3	NaN
9	4	87777.0

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

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

	NoOfPax	EstimatedSalary
0	2.0	40000.0
1	3.0	59000.0
2	2.0	30000.0
3	2.0	120000.0
4	2.0	45000.0
5	2.0	122220.0
6	NaN	21122.0
7	NaN	345673.0
8	3.0	NaN
9	4.0	87777.0

```
df.Age_Group.unique()
```

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

```
df.Hotel.unique()
```

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

```
df['Hotel'] = df['Hotel'].replace(['Ibys'], 'Ibis')
```

```
df.FoodPreference.unique()
```



```
array(['veg', 'Non-Veg', 'Veg', 'Vegetarian', 'non-Veg'], dtype=object)
```

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

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

C:\Users\Dhanush Kumar\AppData\Local\Temp\ipykernel_23860\3377581060.py:1:

FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

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

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

```
df['EstimatedSalary'] =
```

```
df['EstimatedSalary'].fillna(round(df['EstimatedSalary'].mean()))
```

```
df['NoOfPax'] = df['NoOfPax'].fillna(round(df['NoOfPax'].median()))
```

```
df['Bill'] = df['Bill'].fillna(round(df['Bill'].mean()))
```

```
df['Rating(1-5)'] = df['Rating(1-5)'].fillna(round(df['Rating(1-5)'].median()))
```

```
df
```

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	\
0	1.0	20-25	4	Ibis	Veg	1300.0	
1	2.0	30-35	5	LemonTree	Non-Veg	2000.0	
2	3.0	25-30	6	RedFox	Veg	1322.0	
3	4.0	20-25	-1	LemonTree	Veg	1234.0	
4	5.0	35+	3	Ibis	Veg	989.0	
5	6.0	35+	3	Ibis	Non-Veg	1909.0	
6	7.0	35+	4	RedFox	Veg	1000.0	
7	8.0	20-25	7	LemonTree	Veg	2999.0	
8	9.0	25-30	2	Ibis	Non-Veg	3456.0	
9	10.0	30-35	5	RedFox	Non-Veg	1801.0	

	NoOfPax	EstimatedSalary
0	2.0	40000.0
1	3.0	59000.0
2	2.0	30000.0
3	2.0	120000.0
4	2.0	45000.0
5	2.0	122220.0
6	2.0	21122.0
7	2.0	345673.0
8	3.0	96755.0
9	4.0	87777.0

```
import numpy as np
import pandas as pd
df=pd.read_csv("pre_process_datasample.csv")
df
```

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

```
df.info()
```

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

```
RangeIndex: 10 entries, 0 to 9
```

```
Data columns (total 4 columns):
```

```
#   Column   Non-Null Count  Dtype
```

```
---  -----  -
```

```
0  Country   10 non-null   object
1   Age       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
```

```
df.Country.mode()
0    France
Name: Country, dtype: object
df.Country.mode()[0]
'France'
type(df.Country.mode())
pandas.core.series.Series
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
```

C:\Users\Dhanush Kumar\AppData\Local\Temp\ipykernel_19496\1020198583.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the
intermediate object on which we are setting values always behaves as a copy.

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

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

C:\Users\Dhanush Kumar\AppData\Local\Temp\ipykernel_19496\1020198583.py:2:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the
intermediate object on which we are setting values always behaves as a copy.

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

```
df.Age.fillna(df.Age.median(),inplace=True)
```

C:\Users\Dhanush Kumar\AppData\Local\Temp\ipykernel_19496\1020198583.py:3:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

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

```
df.Salary.fillna(round(df.Salary.mean()),inplace=True)
```

```
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  Germany  50.0  83000.0      No
9  France  37.0  67000.0     Yes
pd.get_dummies(df.Country)
```

France Germany Spain

0 True False False

1 False False True

2 False True False

3 False False True

4 False True False

5 True False False

6 False False True

7 True False False

8 False True False

9 True False False

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

updated_dataset

France Germany Spain Age Salary Purchased

0 True False False 44.0 72000.0 No

1 False False True 27.0 48000.0 Yes

2 False True False 30.0 54000.0 No

3 False False True 38.0 61000.0 No

4 False True False 40.0 63778.0 Yes

5 True False False 35.0 58000.0 Yes

6 False False True 38.0 52000.0 No

7 True False False 48.0 79000.0 Yes

8 False True False 50.0 83000.0 No

9 True False False 37.0 67000.0 Yes

df.info()

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

RangeIndex: 10 entries, 0 to 9

Data columns (total 4 columns):

```
# Column Non-Null Count Dtype
```

```
--- ---- -
```

```
0 Country 10 non-null object
```

```
1 Age 10 non-null float64
```

```
2 Salary 10 non-null float64
```

```
3 Purchased 10 non-null object
```

```
dtypes: float64(2), object(2)
```

```
memory usage: 448.0+ bytes
```

```
updated_dataset.Purchased.replace(['No','Yes'],[0,1],inplace=True)
```

```
updated_dataset
```

```
C:\Users\Dhanush Kumar\AppData\Local\Temp\ipykernel_19496\3486364662.py:1:
```

```
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
```

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

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

```
updated_dataset.Purchased.replace(['No','Yes'],[0,1],inplace=True)
```

```
C:\Users\Dhanush Kumar\AppData\Local\Temp\ipykernel_19496\3486364662.py:1:
```

```
FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
```

```
updated_dataset.Purchased.replace(['No','Yes'],[0,1],inplace=True)
```

```
France Germany Spain Age Salary Purchased
```

```
0 True False False 44.0 72000.0 0
```

1	False	False	True	27.0	48000.0	1
2	False	True	False	30.0	54000.0	0
3	False	False	True	38.0	61000.0	0
4	False	True	False	40.0	63778.0	1
5	True	False	False	35.0	58000.0	1
6	False	False	True	38.0	52000.0	0
7	True	False	False	48.0	79000.0	1
8	False	True	False	50.0	83000.0	0
9	True	False	False	37.0	67000.0	1

```
import numpy as np
import pandas as pd
df=pd.read_csv('pre_process_datasample.csv')
df
```

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

```
df.head()
```

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

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

```
features
label
```

```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

```
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]])
```

```
SimpleImputer()
```

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

```
SimpleImputer()
```

```
SimpleImputer()
```

```
SimpleImputer()
```

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

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

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

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

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



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

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit(final_set)
feat_standard_scaler=sc.transform(final_set)
feat_standard_scaler
```

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

```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler(feature_range=(0,1))
mms.fit(final_set)
feat_minmax_scaler=mms.transform(final_set)
feat_minmax_scaler
```

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

```

[1.          , 0.          , 0.          , 0.91304348, 0.88571429],
[0.          , 1.          , 0.          , 1.          , 1.          ],
[1.          , 0.          , 0.          , 0.43478261, 0.54285714]])

```

```

import numpy as np
array=np.random.randint(1,100,16)
array

array([78, 82, 16, 15, 41, 27, 40, 37, 52, 83, 69, 93, 98, 90, 94, 33],
      dtype=int32)

array.mean()

np.float64(59.25)

np.percentile(array,25)

np.float64(36.0)

np.percentile(array,50)

np.float64(60.5)

np.percentile(array,75)

np.float64(84.75)

np.percentile(array,100)

np.float64(98.0)

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

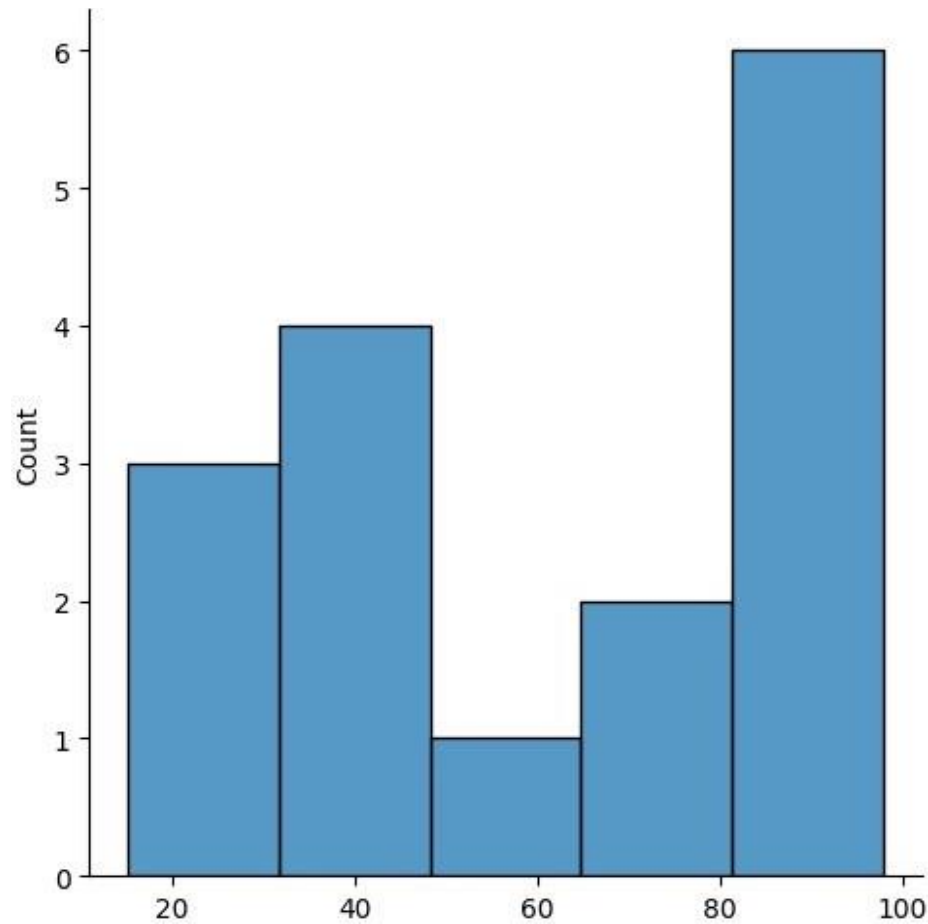
lr,ur=outDetection(array)
lr,ur

(np.float64(-37.125), np.float64(157.875))

sns.displot(array)

<seaborn.axisgrid.FacetGrid at 0x2704bddf7f0>

```



```
sns.distplot(array)
```

C:\Users\Dhanush Kumar\AppData\Local\Temp\ipykernel_24196\1133588802.py:1:
UserWarning:

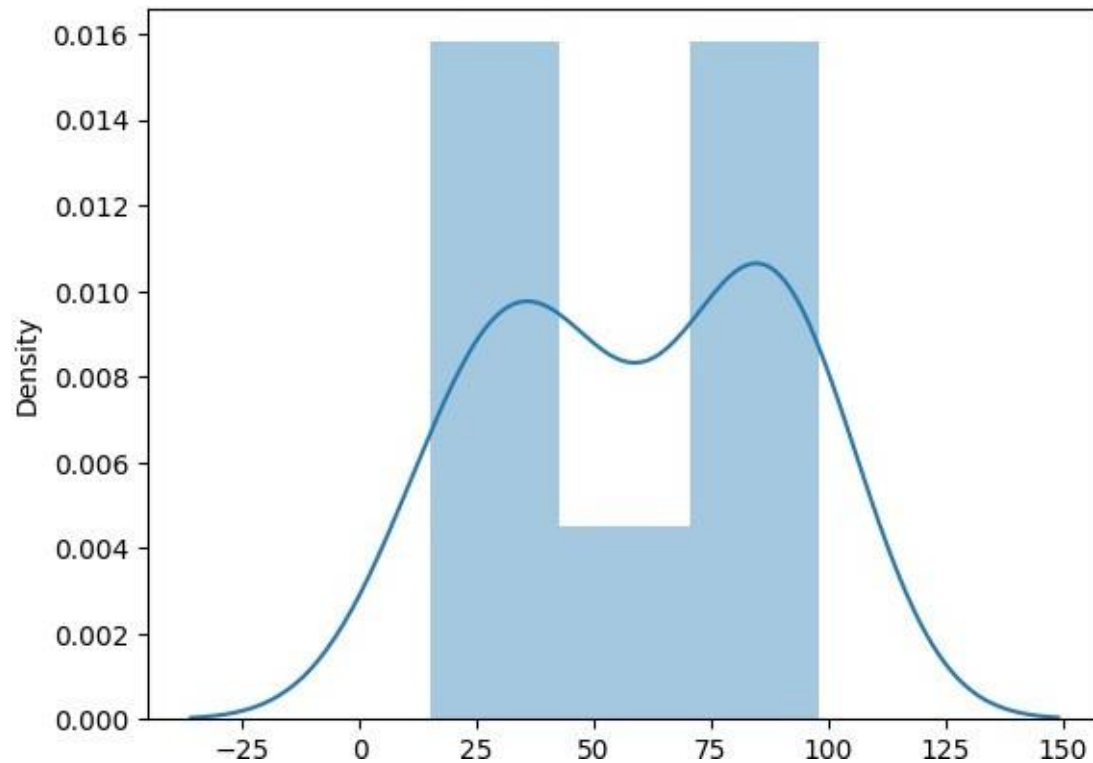
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

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)
```

<Axes: ylabel='Density'>



```
new_array=array[(array>lr) & (array<ur)]
new_array

array([78, 82, 16, 15, 41, 27, 40, 37, 52, 83, 69, 93, 98, 90, 94, 33],
      dtype=int32)

lr1,ur1=outDetection(new_array)
lr1,ur1

(np.float64(-37.125), np.float64(157.875))

final_array=new_array[(new_array>lr1) & (new_array<ur1)]
final_array

array([89,  2, 12, 49, 51, 15, 68, 85, 80, 24, 33, 53, 84, 50, 53,  8])

sns.distplot(final_array)
```

C:\Users\HDC0422095\AppData\Local\Temp\ipykernel_2332\209491988.py:1:
UserWarning:

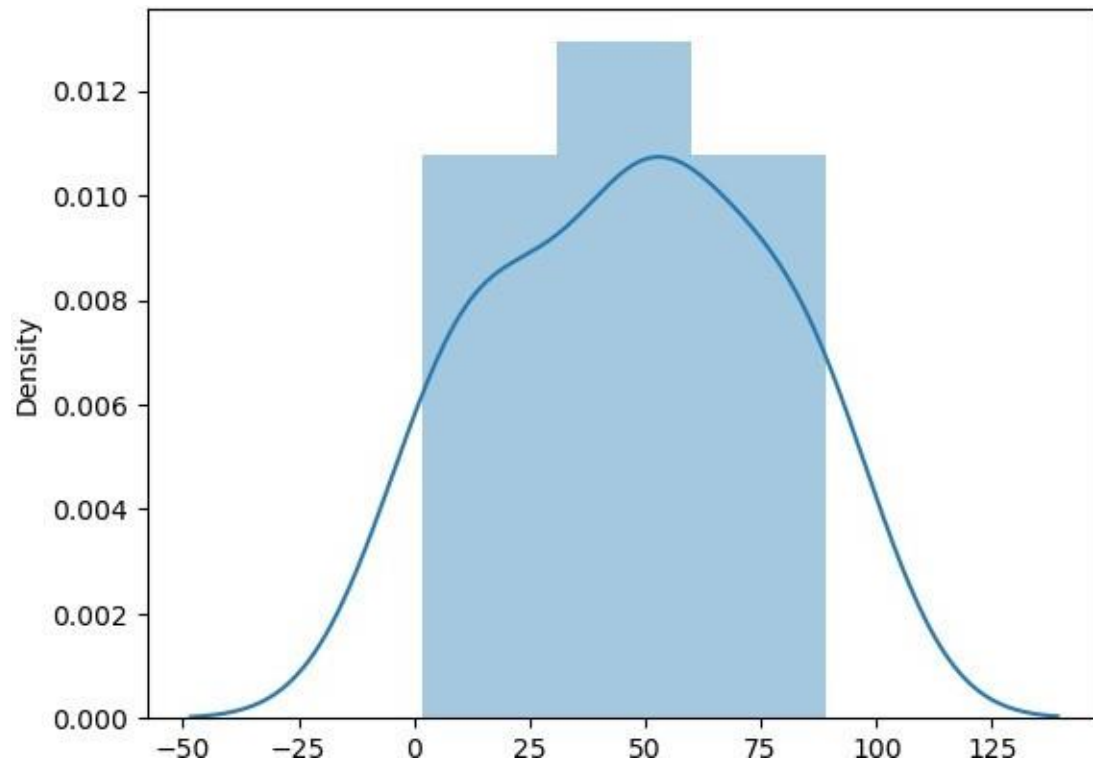
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

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(final_array)
<Axes: ylabel='Density'>
```



```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
tips=sns.load_dataset('tips')
tips.head()

total_bill  tip  sex smoker day  time  size
0    16.99  1.01 Female   No  Sun  Dinner    2
1    10.34  1.66  Male   No  Sun  Dinner    3
```

2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

tips

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
..
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

[244 rows x 7 columns]

```
sns.displot(tips.total_bill,kde=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x1ab9bf48a60>
```

```
sns.displot(tips.total_bill,kde=False)
```

```
<seaborn.axisgrid.FacetGrid at 0x294ad6d7d10>
```

```
sns.jointplot(x=tips.tip,y=tips.total_bill)
```

```
<seaborn.axisgrid.JointGrid at 0x294ad778a70>
```

```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="reg")  
<seaborn.axisgrid.JointGrid at 0x294a8ba4740>
```

```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="hex")  
<seaborn.axisgrid.JointGrid at 0x294ade95220>
```

```
sns.pairplot(tips)  
<seaborn.axisgrid.PairGrid at 0x294a8c7aa20>
```

```
tips.time.value_counts()
```

```
time
```

```
Dinner    176
```

```
Lunch      68
```

```
Name: count, dtype: int64
```

```
sns.pairplot(tips,hue='time')  
<seaborn.axisgrid.PairGrid at 0x294aed013a0>
```

```
sns.pairplot(tips,hue='day')  
<seaborn.axisgrid.PairGrid at 0x294aecf56a0>
```

```
sns.heatmap(tips.corr(numeric_only=True),annot=True)  
<Axes: >
```

```
sns.boxplot(tips.total_bill)  
<Axes: ylabel='total_bill'>
```

```
sns.boxplot(tips.tip)
```

```
<Axes: ylabel='tip'>
```

```
sns.countplot(tips.day)
```

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

```
sns.countplot(tips.sex)
```

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

```
tips.sex.value_counts().plot(kind='pie')
```

```
<Axes: ylabel='count'>
```

```
tips.sex.value_counts().plot(kind='bar')
```

```
<Axes: xlabel='sex'>
```

```
sns.countplot(tips[tips.time=='Dinner']['day'])
```

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

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

	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()
```

	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,random_state=23)
```

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
```

```
LinearRegression()
```

```
print(x_train.shape)
print(y_train.shape)
```

```
(24, 1)
(24, 1)
```

```
model.score(x_train,y_train)
```

```
0.9603182547438908
```

```
model.score(x_test,y_test)
```

```
0.9184170849214232
```

```
model.coef_
```

```
array([[9281.30847068]])
```

```
model.intercept_
```

```
array([27166.73682891])
```

```
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[0]))
```

```
Enter Years of Experience: 41
```

Estimated Salary for 41.0 years of experience is: [407700.38412682]

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

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

[400 rows x 5 columns]

```
df.head()
```

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

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

```
array([[ 19, 19000],
       [ 35, 20000],
       [ 26, 43000],
       [ 27, 57000],
       [ 19, 76000],
       [ 27, 58000],
       [ 27, 84000],
       [ 32, 150000],
       [ 25, 33000],
       [ 35, 65000],
       [ 26, 80000],
       [ 26, 52000],
```

[20, 86000],
[32, 18000],
[18, 82000],
[29, 80000],
[47, 25000],
[45, 26000],
[46, 28000],
[48, 29000],
[45, 22000],
[47, 49000],
[48, 41000],
[45, 22000],
[46, 23000],
[47, 20000],
[49, 28000],
[47, 30000],
[29, 43000],
[31, 18000],
[31, 74000],
[27, 137000],
[21, 16000],
[28, 44000],
[27, 90000],
[35, 27000],
[33, 28000],
[30, 49000],
[26, 72000],
[27, 31000],
[27, 17000],
[33, 51000],
[35, 108000],
[30, 15000],
[28, 84000],
[23, 20000],
[25, 79000],
[27, 54000],
[30, 135000],
[31, 89000],
[24, 32000],
[18, 44000],
[29, 83000],
[35, 23000],
[27, 58000],
[24, 55000],
[23, 48000],
[28, 79000],
[22, 18000],
[32, 117000],
[27, 20000],
[25, 87000],

[23, 66000],
[32, 120000],
[59, 83000],
[24, 58000],
[24, 19000],
[23, 82000],
[22, 63000],
[31, 68000],
[25, 80000],
[24, 27000],
[20, 23000],
[33, 113000],
[32, 18000],
[34, 112000],
[18, 52000],
[22, 27000],
[28, 87000],
[26, 17000],
[30, 80000],
[39, 42000],
[20, 49000],
[35, 88000],
[30, 62000],
[31, 118000],
[24, 55000],
[28, 85000],
[26, 81000],
[35, 50000],
[22, 81000],
[30, 116000],
[26, 15000],
[29, 28000],
[29, 83000],
[35, 44000],
[35, 25000],
[28, 123000],
[35, 73000],
[28, 37000],
[27, 88000],
[28, 59000],
[32, 86000],
[33, 149000],
[19, 21000],
[21, 72000],
[26, 35000],
[27, 89000],
[26, 86000],
[38, 80000],
[39, 71000],
[37, 71000],

[38, 61000],
[37, 55000],
[42, 80000],
[40, 57000],
[35, 75000],
[36, 52000],
[40, 59000],
[41, 59000],
[36, 75000],
[37, 72000],
[40, 75000],
[35, 53000],
[41, 51000],
[39, 61000],
[42, 65000],
[26, 32000],
[30, 17000],
[26, 84000],
[31, 58000],
[33, 31000],
[30, 87000],
[21, 68000],
[28, 55000],
[23, 63000],
[20, 82000],
[30, 107000],
[28, 59000],
[19, 25000],
[19, 85000],
[18, 68000],
[35, 59000],
[30, 89000],
[34, 25000],
[24, 89000],
[27, 96000],
[41, 30000],
[29, 61000],
[20, 74000],
[26, 15000],
[41, 45000],
[31, 76000],
[36, 50000],
[40, 47000],
[31, 15000],
[46, 59000],
[29, 75000],
[26, 30000],
[32, 135000],
[32, 100000],
[25, 90000],

[37, 33000],
[35, 38000],
[33, 69000],
[18, 86000],
[22, 55000],
[35, 71000],
[29, 148000],
[29, 47000],
[21, 88000],
[34, 115000],
[26, 118000],
[34, 43000],
[34, 72000],
[23, 28000],
[35, 47000],
[25, 22000],
[24, 23000],
[31, 34000],
[26, 16000],
[31, 71000],
[32, 117000],
[33, 43000],
[33, 60000],
[31, 66000],
[20, 82000],
[33, 41000],
[35, 72000],
[28, 32000],
[24, 84000],
[19, 26000],
[29, 43000],
[19, 70000],
[28, 89000],
[34, 43000],
[30, 79000],
[20, 36000],
[26, 80000],
[35, 22000],
[35, 39000],
[49, 74000],
[39, 134000],
[41, 71000],
[58, 101000],
[47, 47000],
[55, 130000],
[52, 114000],
[40, 142000],
[46, 22000],
[48, 96000],
[52, 150000],

[59, 42000],
[35, 58000],
[47, 43000],
[60, 108000],
[49, 65000],
[40, 78000],
[46, 96000],
[59, 143000],
[41, 80000],
[35, 91000],
[37, 144000],
[60, 102000],
[35, 60000],
[37, 53000],
[36, 126000],
[56, 133000],
[40, 72000],
[42, 80000],
[35, 147000],
[39, 42000],
[40, 107000],
[49, 86000],
[38, 112000],
[46, 79000],
[40, 57000],
[37, 80000],
[46, 82000],
[53, 143000],
[42, 149000],
[38, 59000],
[50, 88000],
[56, 104000],
[41, 72000],
[51, 146000],
[35, 50000],
[57, 122000],
[41, 52000],
[35, 97000],
[44, 39000],
[37, 52000],
[48, 134000],
[37, 146000],
[50, 44000],
[52, 90000],
[41, 72000],
[40, 57000],
[58, 95000],
[45, 131000],
[35, 77000],
[36, 144000],

[55, 125000],
[35, 72000],
[48, 90000],
[42, 108000],
[40, 75000],
[37, 74000],
[47, 144000],
[40, 61000],
[43, 133000],
[59, 76000],
[60, 42000],
[39, 106000],
[57, 26000],
[57, 74000],
[38, 71000],
[49, 88000],
[52, 38000],
[50, 36000],
[59, 88000],
[35, 61000],
[37, 70000],
[52, 21000],
[48, 141000],
[37, 93000],
[37, 62000],
[48, 138000],
[41, 79000],
[37, 78000],
[39, 134000],
[49, 89000],
[55, 39000],
[37, 77000],
[35, 57000],
[36, 63000],
[42, 73000],
[43, 112000],
[45, 79000],
[46, 117000],
[58, 38000],
[48, 74000],
[37, 137000],
[37, 79000],
[40, 60000],
[42, 54000],
[51, 134000],
[47, 113000],
[36, 125000],
[38, 50000],
[42, 70000],
[39, 96000],

[38, 50000],
[49, 141000],
[39, 79000],
[39, 75000],
[54, 104000],
[35, 55000],
[45, 32000],
[36, 60000],
[52, 138000],
[53, 82000],
[41, 52000],
[48, 30000],
[48, 131000],
[41, 60000],
[41, 72000],
[42, 75000],
[36, 118000],
[47, 107000],
[38, 51000],
[48, 119000],
[42, 65000],
[40, 65000],
[57, 60000],
[36, 54000],
[58, 144000],
[35, 79000],
[38, 55000],
[39, 122000],
[53, 104000],
[35, 75000],
[38, 65000],
[47, 51000],
[47, 105000],
[41, 63000],
[53, 72000],
[54, 108000],
[39, 77000],
[38, 61000],
[38, 113000],
[37, 75000],
[42, 90000],
[37, 57000],
[36, 99000],
[60, 34000],
[54, 70000],
[41, 72000],
[40, 71000],
[42, 54000],
[43, 129000],
[53, 34000],

[47, 50000],
[42, 79000],
[42, 104000],
[59, 29000],
[58, 47000],
[46, 88000],
[38, 71000],
[54, 26000],
[60, 46000],
[60, 83000],
[39, 73000],
[59, 130000],
[37, 80000],
[46, 32000],
[46, 74000],
[42, 53000],
[41, 87000],
[58, 23000],
[42, 64000],
[48, 33000],
[44, 139000],
[49, 28000],
[57, 33000],
[56, 60000],
[49, 39000],
[39, 71000],
[47, 34000],
[48, 35000],
[48, 33000],
[47, 23000],
[45, 45000],
[60, 42000],
[39, 59000],
[46, 41000],
[51, 23000],
[50, 20000],
[36, 33000],
[49, 36000]])

label

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

```

0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
1, 1, 0, 1])

```

```

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

for i in range(1,401):

x_train,x_test,y_train,y_test=train_test_split(features,label,test_size=0.5,r
andom_state=i)
    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 0.845 Train0.84 Random State 3
Test 0.855 Train0.83 Random State 5
Test 0.855 Train0.805 Random State 6
Test 0.865 Train0.815 Random State 7
Test 0.85 Train0.835 Random State 8
Test 0.855 Train0.83 Random State 10
Test 0.84 Train0.835 Random State 13
Test 0.87 Train0.865 Random State 15
Test 0.875 Train0.845 Random State 17
Test 0.87 Train0.835 Random State 18
Test 0.825 Train0.82 Random State 20
Test 0.855 Train0.82 Random State 21
Test 0.865 Train0.84 Random State 22
Test 0.87 Train0.825 Random State 27
Test 0.85 Train0.84 Random State 29
Test 0.85 Train0.835 Random State 37
Test 0.85 Train0.835 Random State 42
Test 0.895 Train0.825 Random State 46
Test 0.85 Train0.815 Random State 47
Test 0.855 Train0.835 Random State 48
Test 0.86 Train0.845 Random State 51
Test 0.845 Train0.84 Random State 52
Test 0.87 Train0.81 Random State 54
Test 0.855 Train0.795 Random State 56

```

Test 0.85 Train0.84 Random State 59
Test 0.875 Train0.855 Random State 61
Test 0.875 Train0.855 Random State 64
Test 0.855 Train0.825 Random State 65
Test 0.87 Train0.815 Random State 68
Test 0.835 Train0.83 Random State 72
Test 0.835 Train0.82 Random State 73
Test 0.875 Train0.84 Random State 74
Test 0.875 Train0.85 Random State 75
Test 0.865 Train0.84 Random State 76
Test 0.855 Train0.83 Random State 77
Test 0.855 Train0.825 Random State 79
Test 0.86 Train0.825 Random State 82
Test 0.865 Train0.835 Random State 84
Test 0.88 Train0.83 Random State 85
Test 0.855 Train0.835 Random State 86
Test 0.87 Train0.815 Random State 88
Test 0.905 Train0.805 Random State 90
Test 0.845 Train0.84 Random State 91
Test 0.87 Train0.84 Random State 95
Test 0.86 Train0.84 Random State 98
Test 0.87 Train0.845 Random State 99
Test 0.85 Train0.84 Random State 100
Test 0.87 Train0.845 Random State 104
Test 0.855 Train0.835 Random State 105
Test 0.865 Train0.83 Random State 106
Test 0.86 Train0.83 Random State 109
Test 0.87 Train0.815 Random State 112
Test 0.855 Train0.825 Random State 115
Test 0.86 Train0.84 Random State 116
Test 0.85 Train0.84 Random State 119
Test 0.88 Train0.83 Random State 120
Test 0.85 Train0.835 Random State 123
Test 0.865 Train0.85 Random State 125
Test 0.855 Train0.845 Random State 127
Test 0.865 Train0.85 Random State 128
Test 0.87 Train0.85 Random State 130
Test 0.865 Train0.825 Random State 133
Test 0.855 Train0.84 Random State 134
Test 0.87 Train0.85 Random State 136
Test 0.85 Train0.825 Random State 141
Test 0.845 Train0.83 Random State 143
Test 0.835 Train0.83 Random State 146
Test 0.85 Train0.845 Random State 147
Test 0.86 Train0.83 Random State 148
Test 0.88 Train0.825 Random State 150
Test 0.895 Train0.835 Random State 152
Test 0.86 Train0.855 Random State 154
Test 0.86 Train0.82 Random State 155
Test 0.87 Train0.86 Random State 156

Test 0.86 Train0.85 Random State 159
Test 0.865 Train0.845 Random State 162
Test 0.85 Train0.8 Random State 163
Test 0.87 Train0.835 Random State 164
Test 0.845 Train0.84 Random State 173
Test 0.855 Train0.845 Random State 174
Test 0.86 Train0.83 Random State 178
Test 0.86 Train0.84 Random State 179
Test 0.86 Train0.825 Random State 180
Test 0.875 Train0.84 Random State 184
Test 0.845 Train0.84 Random State 185
Test 0.86 Train0.85 Random State 186
Test 0.875 Train0.825 Random State 187
Test 0.84 Train0.835 Random State 189
Test 0.86 Train0.84 Random State 192
Test 0.845 Train0.83 Random State 194
Test 0.835 Train0.81 Random State 196
Test 0.85 Train0.835 Random State 200
Test 0.88 Train0.805 Random State 202
Test 0.885 Train0.835 Random State 203
Test 0.865 Train0.845 Random State 206
Test 0.86 Train0.82 Random State 209
Test 0.855 Train0.83 Random State 211
Test 0.855 Train0.835 Random State 212
Test 0.89 Train0.83 Random State 213
Test 0.865 Train0.85 Random State 214
Test 0.865 Train0.83 Random State 217
Test 0.895 Train0.82 Random State 220
Test 0.875 Train0.805 Random State 223
Test 0.855 Train0.845 Random State 228
Test 0.86 Train0.85 Random State 229
Test 0.865 Train0.84 Random State 232
Test 0.845 Train0.84 Random State 235
Test 0.855 Train0.825 Random State 242
Test 0.84 Train0.835 Random State 243
Test 0.87 Train0.835 Random State 245
Test 0.875 Train0.845 Random State 247
Test 0.895 Train0.82 Random State 252
Test 0.855 Train0.85 Random State 254
Test 0.875 Train0.825 Random State 256
Test 0.86 Train0.835 Random State 260
Test 0.87 Train0.815 Random State 266
Test 0.855 Train0.835 Random State 268
Test 0.85 Train0.83 Random State 269
Test 0.885 Train0.82 Random State 270
Test 0.87 Train0.84 Random State 277
Test 0.875 Train0.815 Random State 285
Test 0.87 Train0.835 Random State 287
Test 0.855 Train0.835 Random State 289
Test 0.87 Train0.815 Random State 290

```
Test 0.83 Train0.82 Random State 296
Test 0.855 Train0.825 Random State 297
Test 0.855 Train0.85 Random State 302
Test 0.855 Train0.85 Random State 303
Test 0.865 Train0.845 Random State 304
Test 0.885 Train0.835 Random State 306
Test 0.89 Train0.84 Random State 308
Test 0.87 Train0.835 Random State 314
Test 0.875 Train0.845 Random State 317
Test 0.855 Train0.85 Random State 318
Test 0.86 Train0.83 Random State 319
Test 0.86 Train0.83 Random State 321
Test 0.855 Train0.85 Random State 326
Test 0.835 Train0.825 Random State 331
Test 0.84 Train0.825 Random State 334
Test 0.865 Train0.835 Random State 336
Test 0.865 Train0.81 Random State 337
Test 0.875 Train0.81 Random State 339
Test 0.865 Train0.86 Random State 344
Test 0.875 Train0.855 Random State 347
Test 0.87 Train0.85 Random State 351
Test 0.865 Train0.82 Random State 354
Test 0.87 Train0.82 Random State 358
Test 0.865 Train0.855 Random State 361
Test 0.87 Train0.84 Random State 362
Test 0.89 Train0.83 Random State 363
Test 0.865 Train0.86 Random State 364
Test 0.86 Train0.835 Random State 366
Test 0.875 Train0.825 Random State 369
Test 0.865 Train0.845 Random State 370
Test 0.87 Train0.855 Random State 371
Test 0.865 Train0.815 Random State 376
Test 0.885 Train0.83 Random State 378
Test 0.865 Train0.85 Random State 379
Test 0.865 Train0.825 Random State 383
Test 0.865 Train0.84 Random State 386
Test 0.865 Train0.855 Random State 390
Test 0.875 Train0.835 Random State 393
Test 0.855 Train0.83 Random State 394
Test 0.89 Train0.84 Random State 399
```

```
x_train,x_test,y_train,y_test=train_test_split(features,label,test_size=0.2,r
andom_state=3)
finalModel=LogisticRegression()
finalModel.fit(x_train,y_train)

LogisticRegression()

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

```
0.85625
0.8375
```

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

	precision	recall	f1-score	support
0	0.86	0.91	0.89	257
1	0.83	0.74	0.78	143
accuracy			0.85	400
macro avg	0.85	0.83	0.84	400
weighted avg	0.85	0.85	0.85	400

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

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

	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
..
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

```
[200 rows x 5 columns]
```

```
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

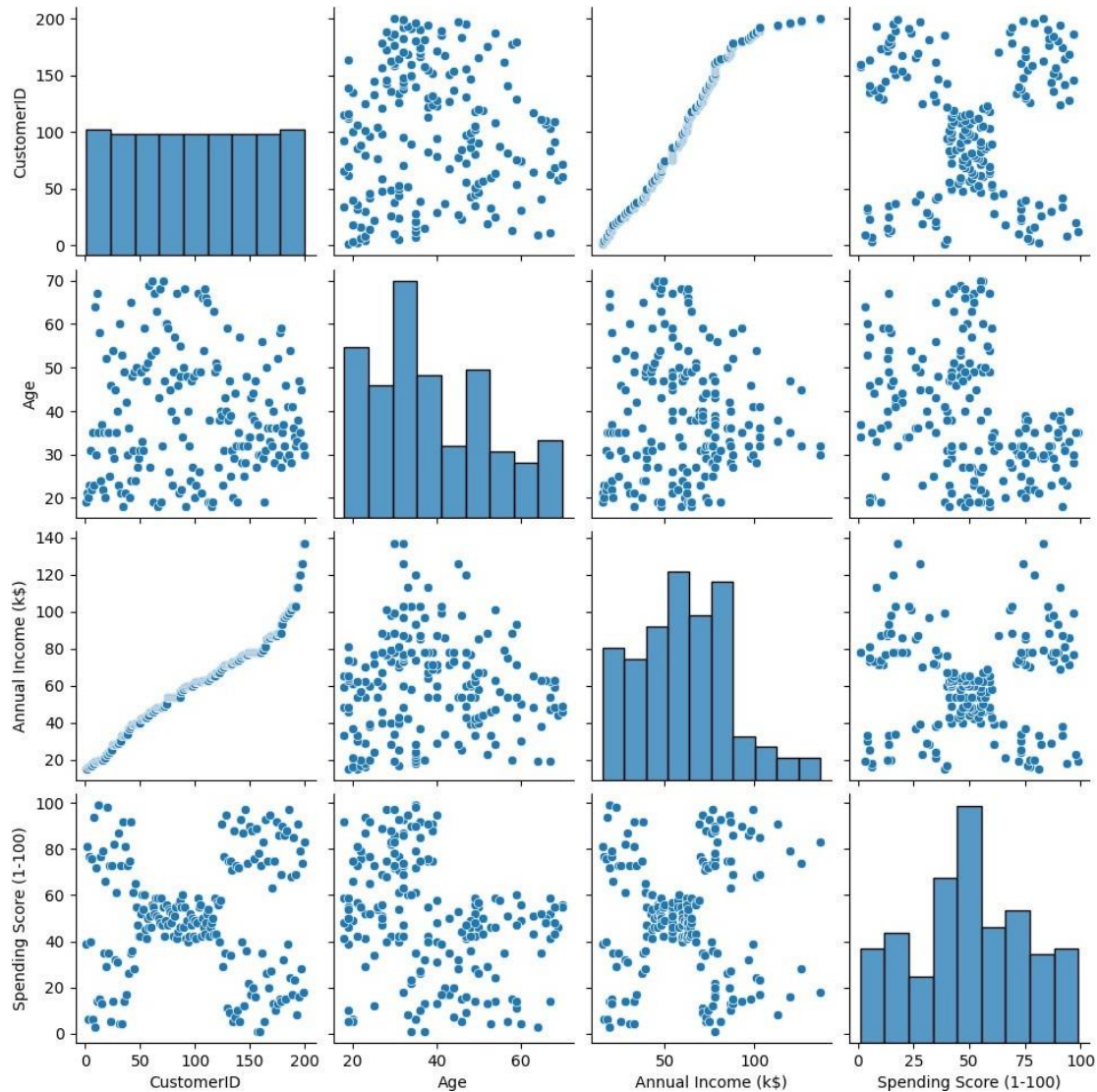

```
2   Age                200 non-null    int64
3   Annual Income (k$)  200 non-null    int64
4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

```
df.head()
```

	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

```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x179eb0db0a0>
```



```
features=df.iloc[:,[3,4]].values
```

```
from sklearn.cluster import KMeans
```

```
model=KMeans(n_clusters=5)
```

```
model.fit(features)
```

```
KMeans(n_clusters=5)
```

```
KMeans(n_clusters=5)
```

```
Final = df.iloc[:, [3, 4]].copy()
```

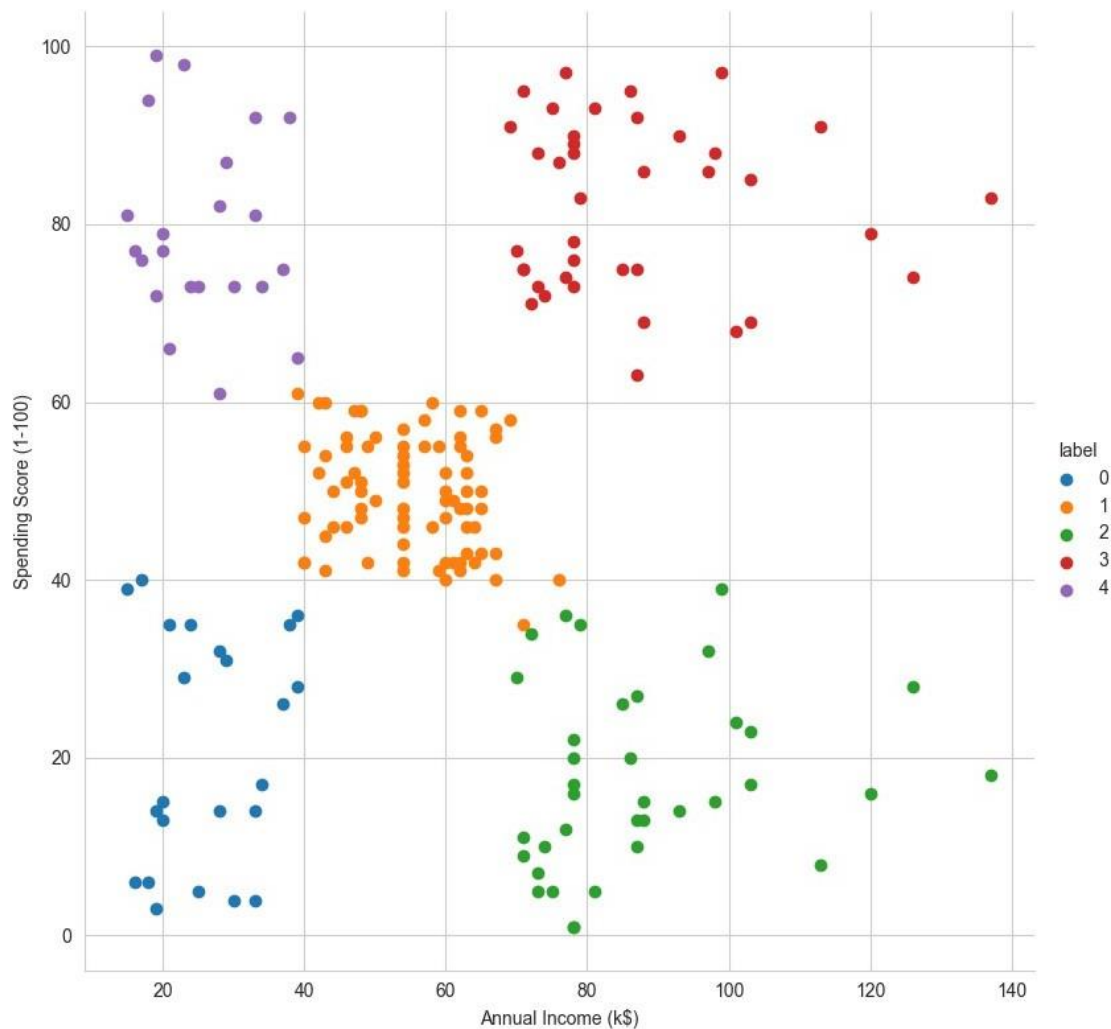
```
Final['label'] = model.predict(features)
```

```
Final.head()
```

	Annual Income (k\$)	Spending Score (1-100)	label
0	15	39	0
1	15	81	4
2	16	6	0

3	16	77	4
4	17	40	0

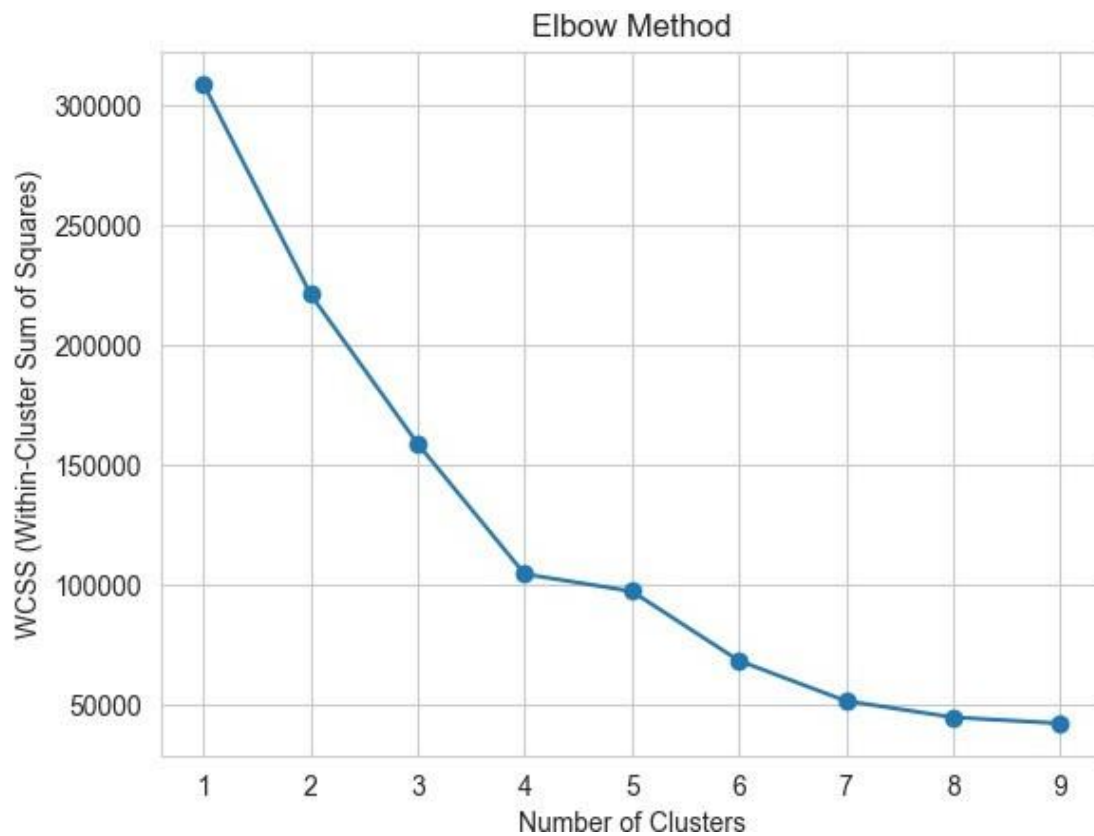
```
sns.set_style("whitegrid")
sns.FacetGrid(Final,hue="label",height=8) \
.map(plt.scatter,"Annual Income (k$)", "Spending Score (1-100)") \
.add_legend();
plt.show()
```



```
features_el=df.iloc[:,[2,3,4]].values
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 10):
    model = KMeans(n_clusters=i, random_state=42)
    model.fit(features_el)
    wcss.append(model.inertia_)

plt.plot(range(1, 10), wcss, marker='o')
plt.title("Elbow Method")
```

```
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS (Within-Cluster Sum of Squares)")
plt.show()
```



```
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
```

```
df.variety.value_counts()
```

```
variety
Setosa      50
Versicolor  50
Virginica   50
Name: count, dtype: int64
```

```
df.head()
```

```
   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
```

```
features=df.iloc[:, :-1].values
label=df.iloc[:, 4].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,random
_state=3)
```

```
model_KNN=KNeighborsClassifier(n_neighbors=5)
```

```
model_KNN.fit(xtrain,ytrain)
```

```
KNeighborsClassifier()
```

```
print(model_KNN.score(xtrain,ytrain))
```

```
print(model_KNN.score(xtest,ytest))
```

```
0.9666666666666667
```

```
0.9666666666666667
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(label,model_KNN.predict(features))
```

```
array([[50,  0,  0],
       [ 0, 46,  4],
       [ 0,  1, 49]], dtype=int64)
```

```
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            0.98      0.92      0.95         50
   Virginica            0.92      0.98      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
--------------	------	------	------	-----

#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_1samp(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.

```
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("Z-statistic: {z_stat:.3f}")
print("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: {z_stat:.3f}

P-value: {p_value:.4f}

Reject Null Hypothesis → Mean is significantly different from 50 g.

```
#Anova test
```

```
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.
```