```
In [7]: import pandas as pd
        df = pd.read_csv("sales_data.csv")
```

```
In [8]: df
```

Out[8]:

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [9]:

```
#1.data exploration.
#1.load the given dataset into a dataframe.
dm=pd.DataFrame(df)
dm
```

Out[9]:

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [10]:

```
#2.Display the first 5 and last 5 rows os the dataset
dm.head()
```

Out[10]:

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
4	05-01-2025	SmartBottle X100	30	100	3000	No	West

In [11]:

```
dm.tail()
```

Out[11]:

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [12]:

```
#3.Check the shape of the dataset(rows and colums)
dm.shape
```

Out[12]:

```
(20, 7)
```

In [13]:

```
print(df.dtypes)
```

```
Date              object
Product           object
Units_Sold         int64
Price_per_Unit     int64
Total_Sales        int64
Promotion         object
Region            object
dtype: object
```

In [14]:

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

```
Date              0
Product           0
Units_Sold        0
Price_per_Unit    0
Total_Sales       0
Promotion         0
Region            0
dtype: int64
```

In [15]:

```
# 1.5 Display summary statistics
print(df.describe())
```

```
       Units_Sold  Price_per_Unit  Total_Sales
count   20.000000       20.000000    20.000000
mean    53.750000       99.000000  5299.250000
std     17.414075        2.051957  1644.051698
min     30.000000       95.000000  3000.000000
25%     37.250000      100.000000  3725.000000
50%     52.500000      100.000000  5250.000000
75%     68.500000      100.000000  6725.000000
max     80.000000      100.000000  7600.000000
```

```
In [16]: # 1.6 Identify duplicate rows and remove them
         duplicates = df.duplicated().sum()
         df = df.drop_duplicates()
         print("Duplicate Rows Removed:", duplicates)
```

```
Duplicate Rows Removed: 0
```

```
In [17]: # 1.7 Get the number of unique values in each column
         print(df.nunique())
```

```
Date              20
Product            1
Units_Sold        19
Price_per_Unit     2
Total_Sales       19
Promotion          2
Region             5
dtype: int64
```

```
In [18]: #2. DATA CLEANING
         # 2.1 Fill missing numerical values with the mean
         df = df.fillna(df.mean(numeric_only=True))
```

```
In [19]: df
```

Out[19]:

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [21]:

```
# 2.2 Drop rows with any missing values
df = df.dropna()
```

In [22]:

```
df
```

Out[22]:

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [24]:

```
# 2.3. Replace all occurrences of a specific value in a column (e.g., replace 'N
dm3= dm["Total_Sales"].replace({5000:1500})
dm3
```

Out[24]:

```
0      1500
1      4800
2      6650
3      7410
4      3000
5      4000
6      6500
7      6800
8      7200
9      3800
10     3500
11     3300
12     6000
13     7125
14     7600
15     5500
16     4900
17     3200
18     3000
19     6700
Name: Total_Sales, dtype: int64
```

In [25]:

```
# 4. Remove a specific column from the dataset.
dm4= dm.drop(columns=["Date"])
dm4
```

Out[25]:

	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	SmartBottle X100	50	100	5000	No	North
1	SmartBottle X100	48	100	4800	No	North
2	SmartBottle X100	70	95	6650	Yes	East
3	SmartBottle X100	78	95	7410	Yes	East
4	SmartBottle X100	30	100	3000	No	West
5	SmartBottle X100	40	100	4000	No	West
6	SmartBottle X100	65	100	6500	No	South
7	SmartBottle X100	68	100	6800	No	South
8	SmartBottle X100	72	100	7200	No	North
9	SmartBottle X100	38	100	3800	No	East
10	SmartBottle X100	35	100	3500	No	West
11	SmartBottle X100	33	100	3300	No	North
12	SmartBottle X100	60	100	6000	No	South
13	SmartBottle X100	75	95	7125	Yes	East
14	SmartBottle X100	80	95	7600	Yes	West
15	SmartBottle X100	55	100	5500	No	South
16	SmartBottle X100	49	100	4900	No	North
17	SmartBottle X100	32	100	3200	No	North
18	SmartBottle X100	30	100	3000	No	West
19	SmartBottle X100	67	100	6700	No	Eas

In [26]:

```
# 5. Rename a column from old_name to new_name.
df.rename(columns= {"Product":"Product Name"})
```

Out[26]:

	Date	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

	Date	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [28]:

```
#3. DATA SELECTION AND FILTERING
# 3.1 Select the Units_sold column
print(df["Units_Sold"].head())
```

```
0    50
1    48
2    70
3    78
4    30
Name: Units_Sold, dtype: int64
```

In [29]:

```
# 2. Select the Reviewer and Review columns from the DataFrame.
dm[["Product","Promotion"]]
```

Out[29]:

	Product	Promotion
0	SmartBottle X100	No
1	SmartBottle X100	No
2	SmartBottle X100	Yes
3	SmartBottle X100	Yes
4	SmartBottle X100	No
5	SmartBottle X100	No
6	SmartBottle X100	No
7	SmartBottle X100	No
8	SmartBottle X100	No
9	SmartBottle X100	No
10	SmartBottle X100	No
11	SmartBottle X100	No
12	SmartBottle X100	No
13	SmartBottle X100	Yes
14	SmartBottle X100	Yes
15	SmartBottle X100	No
16	SmartBottle X100	No
17	SmartBottle X100	No
18	SmartBottle X100	No
19	SmartBottle X100	No

In [31]:

```
# 3. Filter the DataFrame to show rows where the total sales is greater than 300
dm[dm["Total_Sales"]>3000]
```

Out[31]:

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West
15	16-01-2025	SmartBottle X100	55	100	5500	No	South

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

```
In [34]: # 4. Filter the dataset to display only rows where the Product is"SmartBottleX10
         dm[dm["Product"]=="SmartBottle X100"]
```

Out[34]:

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [35]:

```
#                         Sorting and Ordering: (All)
# 1. Sort the DataFrame by Total sales in ascending order.
dm7= dm.sort_values("Total_Sales", ascending = True)
dm7
```

Out[35]:

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

In [37]:

```
# 2. Sort the column alphabetically (for example, product).
dm8= dm.sort_values(by=["Product","Total_Sales"], ascending= [True,True])
dm8
```

Out[37]:

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

In [38]:

```
# 3. Sort the column in descending order (for example, Product).
dm= dm.sort_values(by= "Product", ascending = False)
dm
```

Out[38]:

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
6	07-01-2025	SmartBottle X100	65	100	6500	No	South

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [39]:

```python
# 4. Sort the DataFrame by Units_Sold first and then by total sales.
dm= dm.sort_values(by= ["Units_Sold","Total_Sales"], ascending= [True,True])
dm
```

Out[39]:

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas
7	08-01-2025	SmartBottle X100	68	100	6800	No	South

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

In [44]:

```
# 5. Sort the DataFrame by total sales and keep only the Product 3 rows.
dm= dm.sort_values(by= "Total_Sales", ascending = False).head(3)
dm
```

Out[44]:

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
8	09-01-2025	SmartBottle X100	72	100	7200	No	North

In [46]:

```
#                 Renaming Columns: (All)
# 1. Rename the Time column to Product and Product Name.
df.rename(columns= {"Product":"Product Name"})
```

Out[46]:

	Date	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

	Date	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [47]:

```
# 2. Rename the date column to DATE.
dm.rename(columns= {"Date":"DATE"})
df
```

Out[47]:

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

	Date	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [52]:

```
# 3. Rename multiple columns at once (e.g., date to DATE and Product to Product
dm = df
df= df.rename(columns= {"Date":"DATE","Product":"Product Name"})
df
```

Out[52]:

	DATE	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

	DATE	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [53]:

```
# 4. Change the column name to a simpler one (e.g., Product Name to product).
df.rename(columns= {"Product Name":"Product"})
```

Out[53]:

	DATE	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

	DATE	Product	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [55]:

```
#                  Data Aggregation: (Any 5)
# 1. Find the average of the Pictures column.
pd.to_numeric(dm["Units_Sold"]).mean()
```

Out[55]:

```
53.75
```

In [56]:

```
# 2. Find the total sum of the Pictures column.
dm["Units_Sold"].sum()
```

Out[56]:

```
1075
```

In [57]:

```
# 3. Count how many times each value appears in the Restaurant column.
dm["Total_Sales"].value_counts()
```

Out[57]:

```
Total_Sales
3000    2
5000    1
3500    1
3200    1
4900    1
5500    1
7600    1
7125    1
6000    1
3300    1
3800    1
4800    1
7200    1
6800    1
6500    1
4000    1
7410    1
6650    1
6700    1
Name: count, dtype: int64
```

In [58]:
```
# 4. Calculate the total sum of total sales for each unique value in the product
dm.groupby("Product")["Total_Sales"].sum()
```

Out[58]:
```
Product
SmartBottle X100    105985
Name: Total_Sales, dtype: int64
```

In [59]:
```
# 5. Find the highest Quantity in the dataset.
dm["Units_Sold"].max()
```

Out[59]:
```
80
```

In [60]:
```
# 6. Calculate the minimum quantity in the dataset.
df["Units_Sold"].min()
```

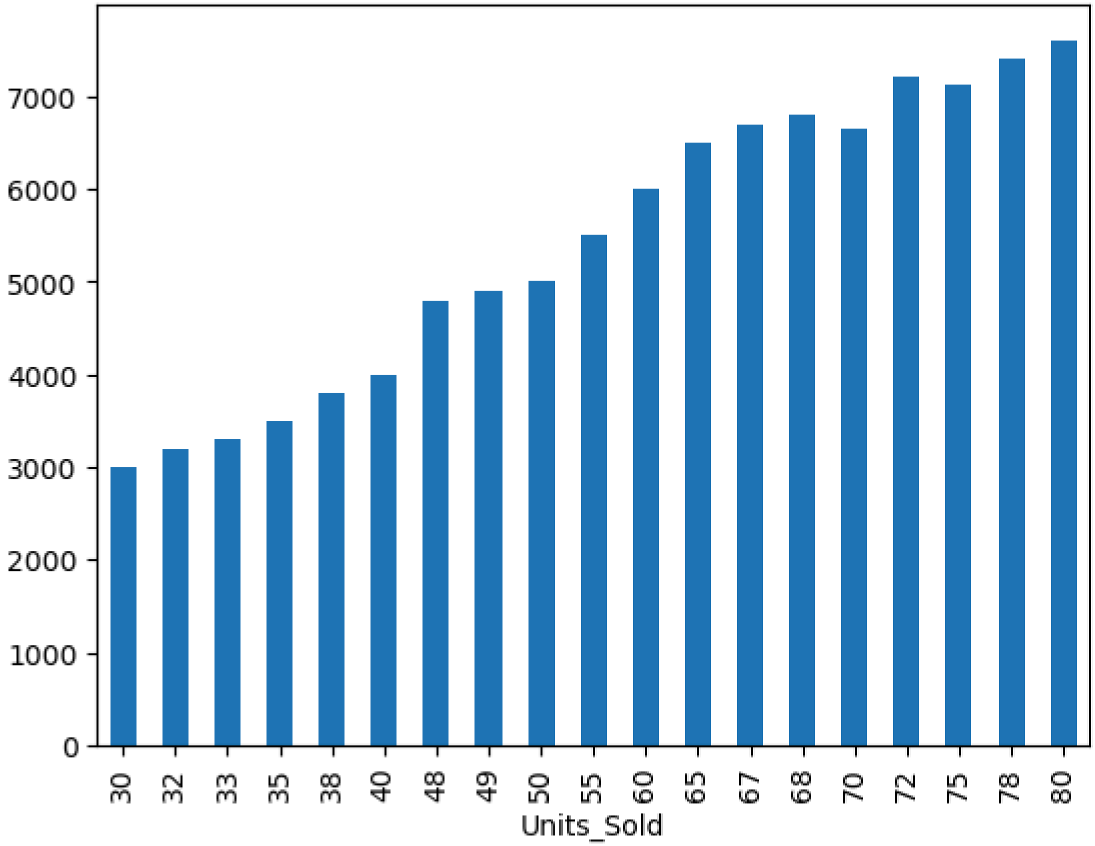
Out[60]:
```
30
```

In [61]:
```
df["Total_Sales"].unique()[0:5]
```

Out[61]:
```
array([5000, 4800, 6650, 7410, 3000])
```

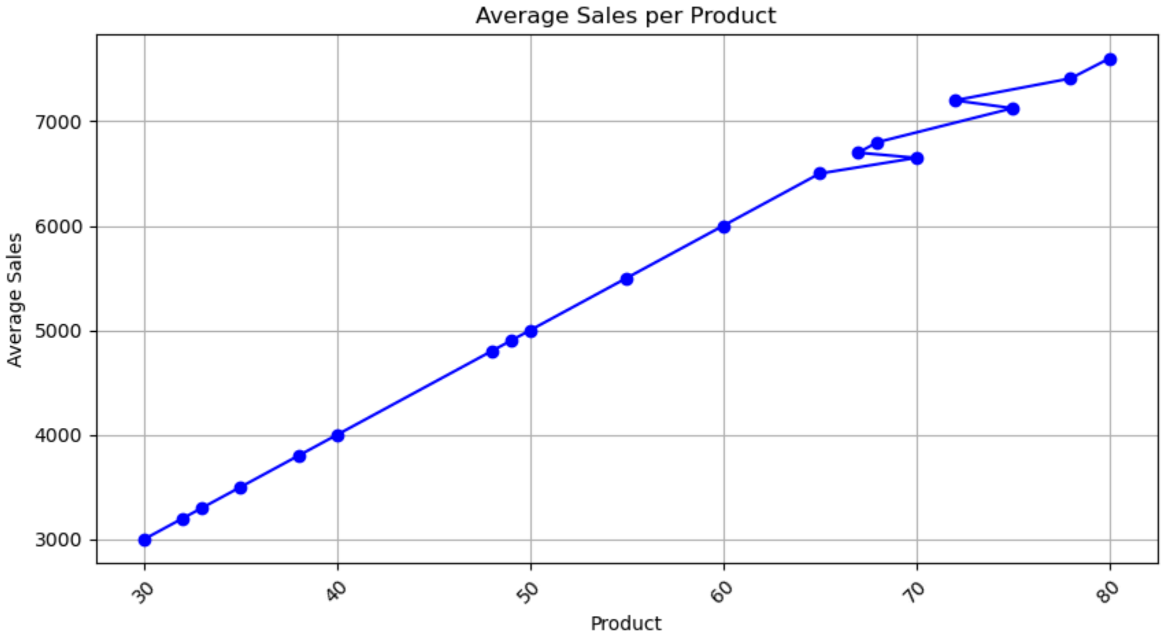
In [63]:
```
#7. SIMPLE DATA VISUALIZATION
import matplotlib.pyplot as plt
import seaborn as sns
```

In [64]:
```
# 7.1 Bar plot of Units_Sold vs Total_Sales
df.groupby("Units_Sold")["Total_Sales"].mean().plot(kind='bar')
plt.show()
```



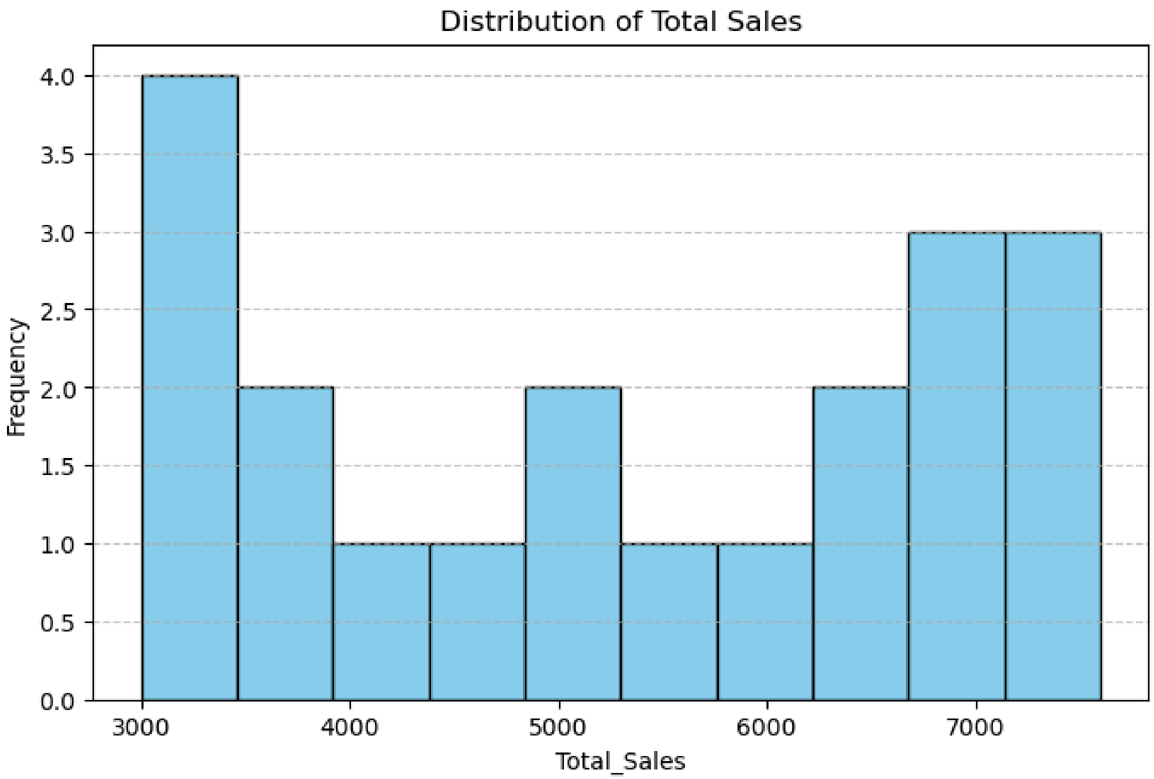
In [70]:
```
avg_sales = df.groupby("Units_Sold")["Total_Sales"].mean()

# Sort values for a better line plot
```

```
avg_sales = avg_sales.sort_values()

# Plot line chart
plt.figure(figsize=(10, 5))
plt.plot(avg_sales.index, avg_sales.values, marker="o", linestyle="-", color="bl

# Labels and title
plt.xlabel("Product")
plt.ylabel("Average Sales")
plt.title("Average Sales per Product")
plt.xticks(rotation=45)
plt.grid(True)
```

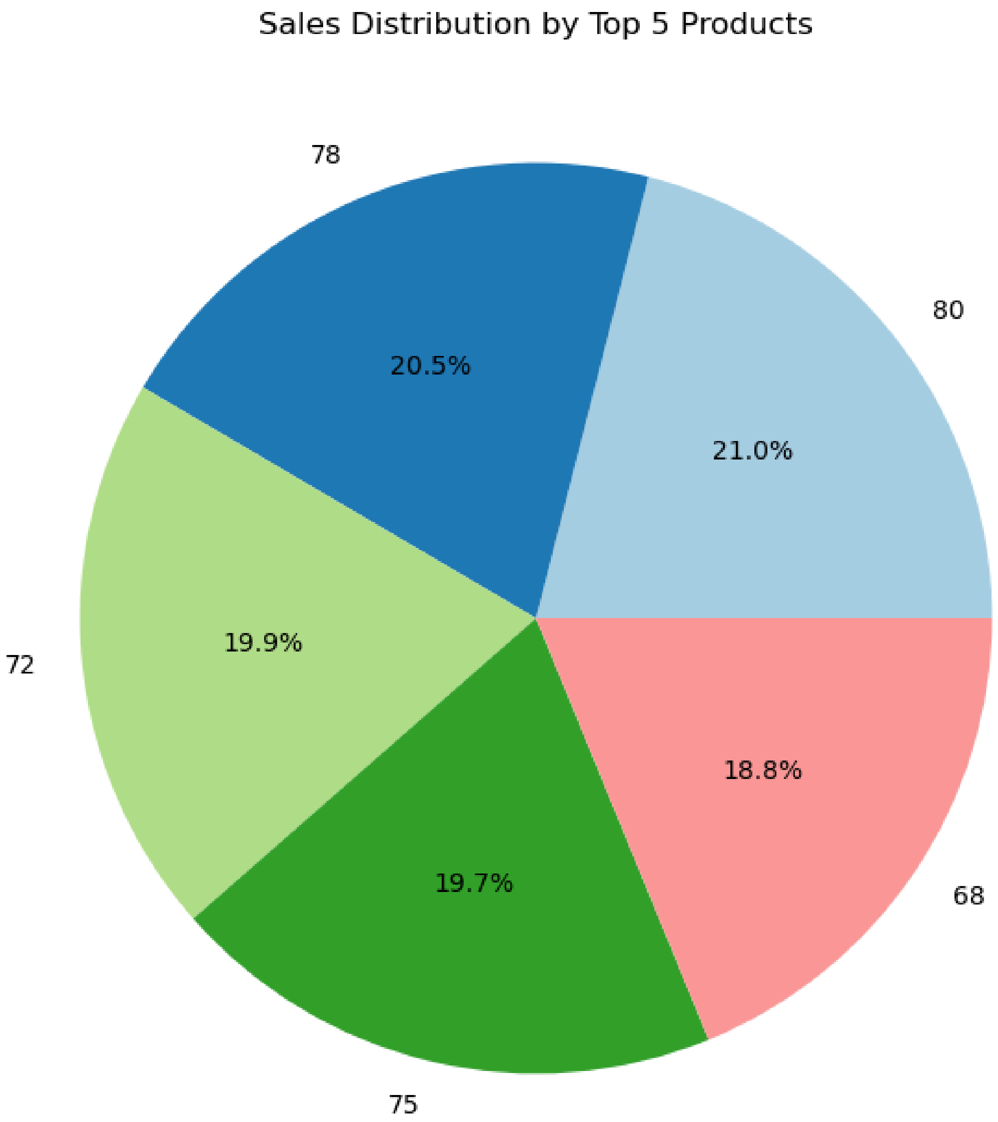


In [71]:

```
plt.figure(figsize=(8, 5))
plt.hist(df["Total_Sales"], bins=10, color="skyblue", edgecolor="black")
plt.xlabel("Total_Sales")
plt.ylabel("Frequency")
plt.title("Distribution of Total Sales")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

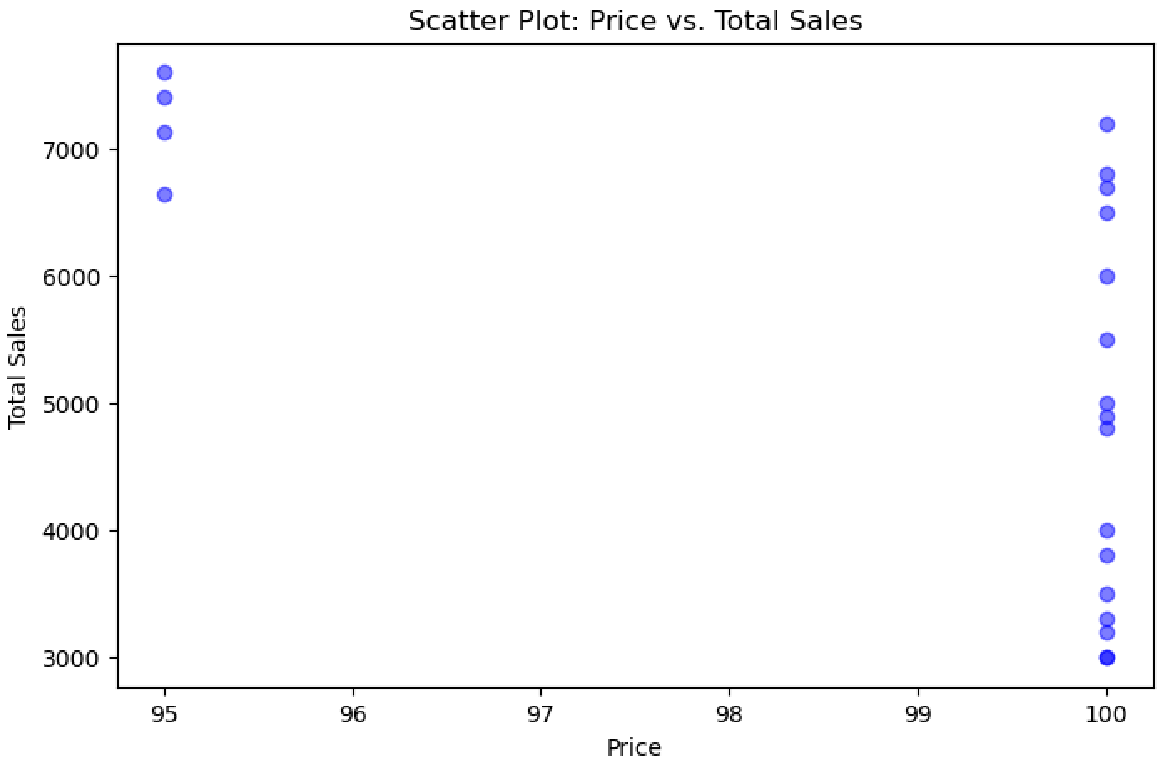


```
In [73]: sales_distribution = df.groupby("Units_Sold")["Total_Sales"].sum().nlargest(5)
         plt.figure(figsize=(8, 8))
         plt.pie(sales_distribution, labels=sales_distribution.index, autopct="%1.1f%%",
         plt.title("Sales Distribution by Top 5 Products")
         plt.show()
```

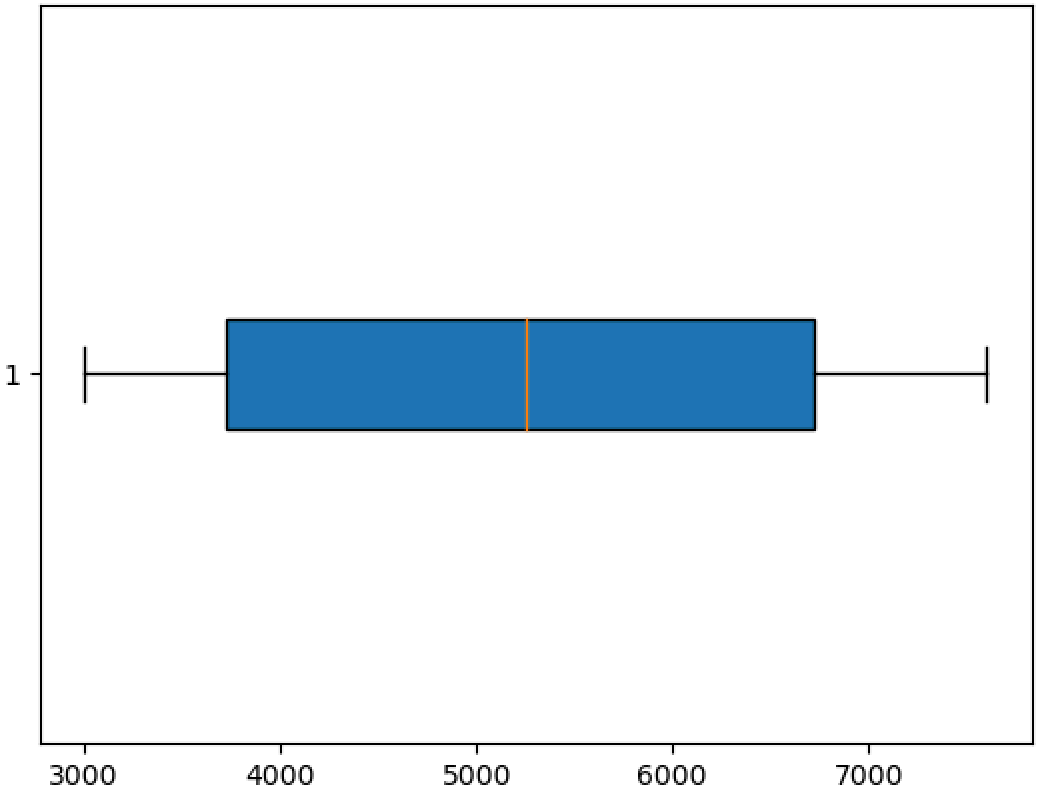


In [75]:

```
plt.figure(figsize=(8, 5))
plt.scatter(df["Price_per_Unit"], df["Total_Sales"], color="blue", alpha=0.5)
plt.xlabel("Price")
plt.ylabel("Total Sales")
plt.title("Scatter Plot: Price vs. Total Sales")
plt.show()
```



```
In [77]: plt.boxplot(dm["Total_Sales"], vert=False, patch_artist=True)
         plt.show()
```



```
In [78]: df
```

Out[78]:

	DATE	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

	DATE	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [79]:

```
#                  Basic String Manipulation: (All)
# 1. Convert the Product column values to lowercase.
dm["Product"].str.lower()
```

Out[79]:

```
0     smartbottle x100
1     smartbottle x100
2     smartbottle x100
3     smartbottle x100
4     smartbottle x100
5     smartbottle x100
6     smartbottle x100
7     smartbottle x100
8     smartbottle x100
9     smartbottle x100
10    smartbottle x100
11    smartbottle x100
12    smartbottle x100
13    smartbottle x100
14    smartbottle x100
15    smartbottle x100
16    smartbottle x100
17    smartbottle x100
18    smartbottle x100
19    smartbottle x100
Name: Product, dtype: object
```

In [80]:

```
# 2. Extract the first 4 characters of the Product column.
dm["Product"].str[:4]
```

Out[80]:

```
0      Smar
1      Smar
2      Smar
3      Smar
4      Smar
5      Smar
6      Smar
7      Smar
8      Smar
9      Smar
10     Smar
11     Smar
12     Smar
13     Smar
14     Smar
15     Smar
16     Smar
17     Smar
18     Smar
19     Smar
Name: Product, dtype: object
```

In [83]:

```
dfcopy = df.copy()
dfcopy[["half1","half2"]]=dfcopy["Product Name"].str.split(" ",n=1,expand = True
dfcopy["half1"] = dfcopy["half2"].fillna("")
dfcopy = dfcopy.drop(columns=["Product Name"])
dfcopy
```

Out[83]:

	DATE	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region	half1	half2
0	01-01-2025	50	100	5000	No	North	X100	X100
1	02-01-2025	48	100	4800	No	North	X100	X100
2	03-01-2025	70	95	6650	Yes	East	X100	X100
3	04-01-2025	78	95	7410	Yes	East	X100	X100
4	05-01-2025	30	100	3000	No	West	X100	X100
5	06-01-2025	40	100	4000	No	West	X100	X100
6	07-01-2025	65	100	6500	No	South	X100	X100
7	08-01-2025	68	100	6800	No	South	X100	X100
8	09-01-2025	72	100	7200	No	North	X100	X100
9	10-01-2025	38	100	3800	No	East	X100	X100
10	11-01-2025	35	100	3500	No	West	X100	X100
11	12-01-2025	33	100	3300	No	North	X100	X100
12	13-01-2025	60	100	6000	No	South	X100	X100
13	14-01-2025	75	95	7125	Yes	East	X100	X100
14	15-01-2025	80	95	7600	Yes	West	X100	X100

	DATE	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region	half1	half2
15	16-01-2025	55	100	5500	No	South	X100	X100
16	17-01-2025	49	100	4900	No	North	X100	X100
17	18-01-2025	32	100	3200	No	North	X100	X100
18	19-01-2025	30	100	3000	No	West	X100	X100
19	20-01-2025	67	100	6700	No	Eas	X100	X100

In [84]:

```
#8. 4. Concatenate the half1 and half2 columns into a new column Clothinh_Item.
dfcopy["Product Name"] = dfcopy["half1"]+ " "+ dfcopy["half2"]
dfcopy = dfcopy.drop(columns=['half1','half2'])
dfcopy
```

Out[84]:

	DATE	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region	Product Name
0	01-01-2025	50	100	5000	No	North	X100 X100
1	02-01-2025	48	100	4800	No	North	X100 X100
2	03-01-2025	70	95	6650	Yes	East	X100 X100
3	04-01-2025	78	95	7410	Yes	East	X100 X100
4	05-01-2025	30	100	3000	No	West	X100 X100
5	06-01-2025	40	100	4000	No	West	X100 X100
6	07-01-2025	65	100	6500	No	South	X100 X100
7	08-01-2025	68	100	6800	No	South	X100 X100
8	09-01-2025	72	100	7200	No	North	X100 X100
9	10-01-2025	38	100	3800	No	East	X100 X100
10	11-01-2025	35	100	3500	No	West	X100 X100
11	12-01-2025	33	100	3300	No	North	X100 X100
12	13-01-2025	60	100	6000	No	South	X100 X100
13	14-01-2025	75	95	7125	Yes	East	X100 X100
14	15-01-2025	80	95	7600	Yes	West	X100 X100
15	16-01-2025	55	100	5500	No	South	X100 X100
16	17-01-2025	49	100	4900	No	North	X100 X100
17	18-01-2025	32	100	3200	No	North	X100 X100
18	19-01-2025	30	100	3000	No	West	X100 X100
19	20-01-2025	67	100	6700	No	Eas	X100 X100

In [86]:

```
#8. 5. Replace all spaces in the region column with underscores.
df["Region"].str.replace(" ", "_")
```

Out[86]:

```
0      North
1      North
2       East
3       East
4       West
5       West
6      South
7      South
8      North
9       East
10      West
11     North
12     South
13      East
14      West
15     South
16     North
17     North
18      West
19       Eas
Name: Region, dtype: object
```

In [87]:

```
#9 iloc - Integer-location based indexing: (Any 5)
# 9.1. Select the first row of the dataset using iloc.
df.iloc[:1]
```

Out[87]:

	DATE	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North

In [88]:

```
# 9.2. Select the last 3 rows of the dataset using iloc.
df.iloc[-3:]
```

Out[88]:

	DATE	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [89]:

```
# 9.3. Select the first 5 rows and the first 3 columns using iloc.
df.iloc[:5,:3]
```

Out[89]:

	DATE	Product Name	Units_Sold
0	01-01-2025	SmartBottle X100	50
1	02-01-2025	SmartBottle X100	48
2	03-01-2025	SmartBottle X100	70
3	04-01-2025	SmartBottle X100	78
4	05-01-2025	SmartBottle X100	30

In [90]:

```
#9. 4. Select all rows of the dataset but only the 2nd and 3rd columns using ilo
df.iloc[:,[1,2]]
```

Out[90]:

	Product Name	Units_Sold
0	SmartBottle X100	50
1	SmartBottle X100	48
2	SmartBottle X100	70
3	SmartBottle X100	78
4	SmartBottle X100	30
5	SmartBottle X100	40
6	SmartBottle X100	65
7	SmartBottle X100	68
8	SmartBottle X100	72
9	SmartBottle X100	38
10	SmartBottle X100	35
11	SmartBottle X100	33
12	SmartBottle X100	60
13	SmartBottle X100	75
14	SmartBottle X100	80
15	SmartBottle X100	55
16	SmartBottle X100	49
17	SmartBottle X100	32
18	SmartBottle X100	30
19	SmartBottle X100	67

In [91]:

```
#9.5. Select the value at the 4th row and 2nd column using iloc.
df.iloc[3,1]
```

Out[91]:

```
'SmartBottle X100'
```

In [92]:

```
# 6. Use iloc to select rows 2 to 5 and columns 1 to 4.
df.iloc[1:5,:4]
```

Out[92]:

	DATE	Product Name	Units_Sold	Price_per_Unit
1	02-01-2025	SmartBottle X100	48	100
2	03-01-2025	SmartBottle X100	70	95
3	04-01-2025	SmartBottle X100	78	95
4	05-01-2025	SmartBottle X100	30	100

In [94]:

```
#   10              loc - Label-based indexing: (Any 5)
#10. 1. Select the row where the Reviewer is "Dileep" using loc.
df.loc[df["Region"]=="North"]
```

Out[94]:

	DATE	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North

In [96]:

```
#10.2. Select rows where the Rating is greater than 4.5 using loc.
df.loc[df["Units_Sold"]>4.5]
```

Out[96]:

	DATE	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
0	01-01-2025	SmartBottle X100	50	100	5000	No	North
1	02-01-2025	SmartBottle X100	48	100	4800	No	North
2	03-01-2025	SmartBottle X100	70	95	6650	Yes	East
3	04-01-2025	SmartBottle X100	78	95	7410	Yes	East
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
6	07-01-2025	SmartBottle X100	65	100	6500	No	South
7	08-01-2025	SmartBottle X100	68	100	6800	No	South
8	09-01-2025	SmartBottle X100	72	100	7200	No	North
9	10-01-2025	SmartBottle X100	38	100	3800	No	East
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
11	12-01-2025	SmartBottle X100	33	100	3300	No	North
12	13-01-2025	SmartBottle X100	60	100	6000	No	South
13	14-01-2025	SmartBottle X100	75	95	7125	Yes	East
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West

	DATE	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
15	16-01-2025	SmartBottle X100	55	100	5500	No	South
16	17-01-2025	SmartBottle X100	49	100	4900	No	North
17	18-01-2025	SmartBottle X100	32	100	3200	No	North
18	19-01-2025	SmartBottle X100	30	100	3000	No	West
19	20-01-2025	SmartBottle X100	67	100	6700	No	Eas

In [97]:

```
#10.3. Select the City sold for rows where sold is between 3 and 4 using loc.
df.loc[(df["Units_Sold"]>3) & (df["Units_Sold"]<4)]
```

Out[97]:

DATE	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region

In [98]:

```
#10.4. Select all rows and columns where Region is "West" using loc.
df.loc[df["Region"]=="West"]
```

Out[98]:

	DATE	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Region
4	05-01-2025	SmartBottle X100	30	100	3000	No	West
5	06-01-2025	SmartBottle X100	40	100	4000	No	West
10	11-01-2025	SmartBottle X100	35	100	3500	No	West
14	15-01-2025	SmartBottle X100	80	95	7600	Yes	West
18	19-01-2025	SmartBottle X100	30	100	3000	No	West

In [99]:

```
#10.5. Select multiple columns (Reviewer, Rating) for a specific row with loc.
df.loc[:0,["Region","Units_Sold"]]
```

Out[99]:

	Region	Units_Sold
0	North	50

In [100…

```
#11. replace - Replacing values in a DataFrame: (Any 5)
#11. 1. Replace all occurrences of the value "Dileep" with "Nitin" in the Review
df["Region"].replace("West","West_side")
```

Out[100…

```
0         North
1         North
2          East
3          East
4     West_side
5     West_side
6         South
7         South
8         North
9          East
10    West_side
11        North
12        South
13         East
14    West_side
15        South
16        North
17        North
18    West_side
19          Eas
Name: Region, dtype: object
```

In [101…

```
#11. 2. Replace the value 0 with NaN in a specific numerical column.
df["Units_Sold"].replace(0,"...")
```

Out[101…

```
0     50
1     48
2     70
3     78
4     30
5     40
6     65
7     68
8     72
9     38
10    35
11    33
12    60
13    75
14    80
15    55
16    49
17    32
18    30
19    67
Name: Units_Sold, dtype: int64
```

In [102…

```
# 3. Replace a value in the entire DataFrame (e.g., replace all 100s with 50s).
df["Units_Sold"].replace(2.5,2.0)
```

Out[102…

```
0     50
1     48
2     70
3     78
4     30
5     40
6     65
7     68
8     72
9     38
10    35
11    33
12    60
13    75
14    80
15    55
16    49
17    32
18    30
19    67
Name: Units_Sold, dtype: int64
```

In [103…

```python
# 11.4.
df["Product Name"].replace({"Promotion":1})
```

Out[103…

```
0     SmartBottle X100
1     SmartBottle X100
2     SmartBottle X100
3     SmartBottle X100
4     SmartBottle X100
5     SmartBottle X100
6     SmartBottle X100
7     SmartBottle X100
8     SmartBottle X100
9     SmartBottle X100
10    SmartBottle X100
11    SmartBottle X100
12    SmartBottle X100
13    SmartBottle X100
14    SmartBottle X100
15    SmartBottle X100
16    SmartBottle X100
17    SmartBottle X100
18    SmartBottle X100
19    SmartBottle X100
Name: Product Name, dtype: object
```

In [104…

```python
#11. 6. Replace the value of "NA" in the Rating column with the mean of the colu
df["Units_Sold"].replace("np.nan", df["Units_Sold"].mean())
```

```
Out[104…  0     50
          1     48
          2     70
          3     78
          4     30
          5     40
          6     65
          7     68
          8     72
          9     38
          10    35
          11    33
          12    60
          13    75
          14    80
          15    55
          16    49
          17    32
          18    30
          19    67
          Name: Units_Sold, dtype: int64
```

```
In [105…  #    12.         index - Indexing and resetting index: (Any 5)
          # 12.1. Set a specific column (e.g., Name) as the index of the DataFrame.
          df.set_index("Region")
```

Out[105…

	DATE	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion
Region						
North	01-01-2025	SmartBottle X100	50	100	5000	No
North	02-01-2025	SmartBottle X100	48	100	4800	No
East	03-01-2025	SmartBottle X100	70	95	6650	Yes
East	04-01-2025	SmartBottle X100	78	95	7410	Yes
West	05-01-2025	SmartBottle X100	30	100	3000	No
West	06-01-2025	SmartBottle X100	40	100	4000	No
South	07-01-2025	SmartBottle X100	65	100	6500	No
South	08-01-2025	SmartBottle X100	68	100	6800	No
North	09-01-2025	SmartBottle X100	72	100	7200	No
East	10-01-2025	SmartBottle X100	38	100	3800	No
West	11-01-2025	SmartBottle X100	35	100	3500	No
North	12-01-2025	SmartBottle X100	33	100	3300	No
South	13-01-2025	SmartBottle X100	60	100	6000	No
East	14-01-2025	SmartBottle X100	75	95	7125	Yes
West	15-01-2025	SmartBottle X100	80	95	7600	Yes
South	16-01-2025	SmartBottle X100	55	100	5500	No
North	17-01-2025	SmartBottle X100	49	100	4900	No
North	18-01-2025	SmartBottle X100	32	100	3200	No
West	19-01-2025	SmartBottle X100	30	100	3000	No
Eas	20-01-2025	SmartBottle X100	67	100	6700	No

```
In [106…  #12. 2. Reset the index of the DataFrame after setting a new column as the index
          df.reset_index()
```

Out[106…

	index	DATE	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Regio
0	0	01-01-2025	SmartBottle X100	50	100	5000	No	Nort
1	1	02-01-2025	SmartBottle X100	48	100	4800	No	Nort
2	2	03-01-2025	SmartBottle X100	70	95	6650	Yes	Ea
3	3	04-01-2025	SmartBottle X100	78	95	7410	Yes	Ea
4	4	05-01-2025	SmartBottle X100	30	100	3000	No	We
5	5	06-01-2025	SmartBottle X100	40	100	4000	No	We
6	6	07-01-2025	SmartBottle X100	65	100	6500	No	Sout
7	7	08-01-2025	SmartBottle X100	68	100	6800	No	Sout
8	8	09-01-2025	SmartBottle X100	72	100	7200	No	Nort
9	9	10-01-2025	SmartBottle X100	38	100	3800	No	Ea
10	10	11-01-2025	SmartBottle X100	35	100	3500	No	We
11	11	12-01-2025	SmartBottle X100	33	100	3300	No	Nort
12	12	13-01-2025	SmartBottle X100	60	100	6000	No	Sout
13	13	14-01-2025	SmartBottle X100	75	95	7125	Yes	Ea
14	14	15-01-2025	SmartBottle X100	80	95	7600	Yes	We

	index	DATE	Product Name	Units_Sold	Price_per_Unit	Total_Sales	Promotion	Regio
15	15	16-01-2025	SmartBottle X100	55	100	5500	No	Sout
16	16	17-01-2025	SmartBottle X100	49	100	4900	No	Nort
17	17	18-01-2025	SmartBottle X100	32	100	3200	No	Nort
18	18	19-01-2025	SmartBottle X100	30	100	3000	No	We
19	19	20-01-2025	SmartBottle X100	67	100	6700	No	Ea

In [108…

```
#12. 3. Set a multi-level index using seasont and sold columns.
dfindex= df.set_index(["Region","Units_Sold"])
dfindex
```

Out[108…

		DATE	Product Name	Price_per_Unit	Total_Sales	Promotion
Region	**Units_Sold**					
North	**50**	01-01-2025	SmartBottle X100	100	5000	No
	48	02-01-2025	SmartBottle X100	100	4800	No
East	**70**	03-01-2025	SmartBottle X100	95	6650	Yes
	78	04-01-2025	SmartBottle X100	95	7410	Yes
West	**30**	05-01-2025	SmartBottle X100	100	3000	No
	40	06-01-2025	SmartBottle X100	100	4000	No
South	**65**	07-01-2025	SmartBottle X100	100	6500	No
	68	08-01-2025	SmartBottle X100	100	6800	No
North	**72**	09-01-2025	SmartBottle X100	100	7200	No
East	**38**	10-01-2025	SmartBottle X100	100	3800	No
West	**35**	11-01-2025	SmartBottle X100	100	3500	No
North	**33**	12-01-2025	SmartBottle X100	100	3300	No
South	**60**	13-01-2025	SmartBottle X100	100	6000	No
East	**75**	14-01-2025	SmartBottle X100	95	7125	Yes
West	**80**	15-01-2025	SmartBottle X100	95	7600	Yes
South	**55**	16-01-2025	SmartBottle X100	100	5500	No
North	**49**	17-01-2025	SmartBottle X100	100	4900	No
	32	18-01-2025	SmartBottle X100	100	3200	No
West	**30**	19-01-2025	SmartBottle X100	100	3000	No
Eas	**67**	20-01-2025	SmartBottle X100	100	6700	No

In [110…

```
# 12.4. Access a specific row by using the index after setting the index to City
dfindex.loc["South"]
```

Out[110…

	DATE	Product Name	Price_per_Unit	Total_Sales	Promotion
Units_Sold					
65	07-01-2025	SmartBottle X100	100	6500	No
68	08-01-2025	SmartBottle X100	100	6800	No
60	13-01-2025	SmartBottle X100	100	6000	No
55	16-01-2025	SmartBottle X100	100	5500	No

In [111…

```
#12. 5. Check the index type and whether it’s unique.
type(dfindex.index)
dfindex.index.is_unique
```

Out[111…

```
False
```

In [112…

```
# 13.                 groupby - Grouping data: (Any 5)
#13. 1. Group the dataset by season and find the average Rating for each sold.
df.groupby("Region")["Units_Sold"].mean()
```

Out[112…

```
Region
Eas      67.000000
East     65.250000
North    47.333333
South    62.000000
West     43.000000
Name: Units_Sold, dtype: float64
```

In [113…

```
#13. 2. Group by Rating and get the count of each group in the season column.
df.groupby("Units_Sold")["Region"].count()
```

Out[113…

```
Units_Sold
30    2
32    1
33    1
35    1
38    1
40    1
48    1
49    1
50    1
55    1
60    1
65    1
67    1
68    1
70    1
72    1
75    1
78    1
80    1
Name: Region, dtype: int64
```

In [114…

```
# 13.3. Group the data by Restaurant and calculate the total Pictures for each s
df.groupby("Region")["Units_Sold"].sum()
```

Out[114…

```
Region
Eas       67
East     261
North    284
South    248
West     215
Name: Units_Sold, dtype: int64
```

In [115…

```
#13. 5. Group by sold and find the number of unique entries in the seson column.
df.groupby("Region")["Units_Sold"].nunique()
```

Out[115…

```
Region
Eas      1
East     4
North    6
South    4
West     4
Name: Units_Sold, dtype: int64
```

In [116…

```
#13. 6. Group the dataset by season and find the maximum value of sold for each
df.groupby("Region")["Units_Sold"].max()
```

Out[116…

```
Region
Eas      67
East     78
North    72
South    68
West     80
Name: Units_Sold, dtype: int64
```

In [117…

```
#  14              aggregation - Performing multiple aggregations: (Any 5)
# 14.1. Group by seaso and calculate both the mean and median of the sold column
df.groupby("Region")["Units_Sold"].mean()
```

Out[117…

```
Region
Eas      67.000000
East     65.250000
North    47.333333
South    62.000000
West     43.000000
Name: Units_Sold, dtype: float64
```

In [118…

```
df.groupby("Region")["Units_Sold"].median()
```

Out[118…

```
Region
Eas      67.0
East     72.5
North    48.5
South    62.5
West     35.0
Name: Units_Sold, dtype: float64
```

In [119…

```
# 14.2. Perform multiple aggregations: find the sum, mean, and count of the seas
# grouped by Restaurant.
df.groupby("Region")["Units_Sold"].agg(["sum","mean","count"]).reset_index()
```

Out[119…

	Region	sum	mean	count
0	Eas	67	67.000000	1
1	East	261	65.250000	4
2	North	284	47.333333	6
3	South	248	62.000000	4
4	West	215	43.000000	5

In [120…

```
# 14.3. Group by season and apply min() and max() functions to the sold column.
df.groupby("Region").agg({
    "Units_Sold":["min","max"]
}).reset_index()
```

Out[120…

	Region	Units_Sold	
		min	max
0	Eas	67	67
1	East	38	78
2	North	32	72
3	South	55	68
4	West	30	80

In [121…

```
#14. 4. Group by seasonand apply a custom aggregation function (e.g., find the r
# within each sold).
df.groupby("Region")["Units_Sold"].agg(lambda x: x.max()-x.min())
```

Out[121…

```
Region
Eas       0
East     40
North    40
South    13
West     50
Name: Units_Sold, dtype: int64
```

In [122…

```
#14. 5. Group by season and calculate the average season, along with the total
# group.
df.groupby("Region").agg({
    "Units_Sold":"mean",
    "Units_Sold":"sum"
}).reset_index()
```

Out[122…

	Region	Units_Sold
0	Eas	67
1	East	261
2	North	284
3	South	248
4	West	215

In [244…

```
#   15.              faker - Generating Fake Data for Testing: (All)
# 1. Generate a fake dataset with 100 rows of random names, addresses, and dates
# Faker library.
import faker
fake= faker.Faker('en_IN')
data=[]
for _ in range(100):
    record={
        "Name": fake.name(),
        "Address": fake.address(),
        "Date": fake.date()
    }
    data.append(record)
fds= pd.DataFrame(data)
fds
```

Out[244…

	Name	Address	Date
0	Leela Balasubramanian	91, Raval\nKamarhati 575769	1974-05-09
1	Shaurya Chandran	235, Datta\nEluru 511856	1970-05-01
2	Azaan Raval	64\nBarman Nagar, Jalgaon-652982	1979-04-16
3	Maanas Gera	25/794\nShenoy\nRajahmundry 334093	1988-08-29
4	Saksham Ramanathan	H.No. 482, Ben Ganj, Sikar 986088	1981-10-12
...	...	...	...
95	Oni Peri	08, Sur\nUdupi-339334	2005-10-15
96	Suhani Din	322, Chokshi Zila\nTadipatri 035748	1997-02-08
97	Tripti Buch	36/84, Kala, Bidar-222616	1998-05-16
98	Odika Prabhu	834, Chaudhary Marg, Indore-072505	2023-08-02
99	Kashvi Contractor	38/884, Chada Nagar\nOrai-150819	1973-09-02

100 rows × 3 columns

In [246…

```
# 15.2. Create a list of 20 fake names and store them in a Name column.
list= {'Name': [fake.name() for _ in range(20)]}
nameds= pd.DataFrame(list)
nameds
```

Out[246…

	Name
0	Alka Sethi
1	Tamanna Bhargava
2	Samaksh Bandi
3	Aahana Deshmukh
4	Rudra Raghavan
5	Teerth Basak
6	Nicholas Viswanathan
7	Triya Goda
8	Vivaan Puri
9	Shravya Vohra
10	Charita Sampath
11	Libni Dhingra
12	Urvashi Chandra
13	Harini Iyer
14	Hema Dyal
15	Qasim Bumb
16	Amol Pingle
17	Samesh Bora
18	Triveni Mall
19	Tanmayi Rege

In [248…

```
# 15.3. Generate fake email addresses and store them in a column called Email.
list= {'Name': [fake.address() for _ in range(20)]}
addressds= pd.DataFrame(list)
addressds
```

Out[248…

	Name
0	H.No. 92, Borde Ganj\nVijayanagaram-952947
1	H.No. 37\nCheema Circle\nBhavnagar 719584
2	22/743, Bhatia Ganj\nPali-844546
3	H.No. 73\nKothari Street, Mumbai-357657
4	H.No. 065\nEdwin Marg\nKarnal 403718
5	H.No. 19\nSani Zila\nDavanagere 704456
6	67/14\nChokshi Ganj, Gwalior-111927
7	13/92, Shetty Marg, Bally-659436
8	95\nPrasad Ganj\nSrikakulam 556809
9	97, Dewan Circle, Madhyamgram 833137
10	H.No. 260\nManne, Sambalpur-508257
11	H.No. 456, Badami\nAlappuzha 002403
12	20\nGoda Path\nPanipat 088316
13	10\nBansal Road\nDindigul-910326
14	H.No. 75, Oommen Circle, Rewa 863242
15	924, Sridhar Ganj\nMumbai-751474
16	99/89\nBasak Nagar\nAsansol 553036
17	H.No. 74\nVenkatesh Nagar, Aurangabad-817505
18	604\nSathe Marg\nRaebareli 607887
19	97\nDhingra Nagar, Ongole-929100

In [250…

```
# 15.4. Generate fake dates of birth and convert them into an Age column using F
from datetime import datetime
dob_list = [fake.date_of_birth(minimum_age=18, maximum_age=80) for _ in range(20
age_data = {'Date_of_Birth': dob_list, 'Age': [(datetime.today().year - dob.year
dobds = pd.DataFrame(age_data)
dobds
```

Out[250…

	Date_of_Birth	Age
0	2004-12-10	21
1	1986-08-21	39
2	1997-06-07	28
3	1958-03-25	67
4	2002-03-14	23
5	2002-12-12	23
6	1958-06-28	67
7	1985-08-20	40
8	1984-09-01	41
9	1949-01-08	76
10	1992-11-18	33
11	2000-06-01	25
12	1981-10-02	44
13	1968-03-05	57
14	1986-09-09	39
15	1993-02-21	32
16	1966-10-06	59
17	1963-10-03	62
18	1999-01-23	26
19	1986-11-28	39

In [252…

```python
# 15.5. Generate fake product names, and then create a DataFrame with columns fo
# Product_Name and Price.
import random
product_data = {
    'Product_Name': [fake.word().capitalize() for _ in range(20)],
    'Price': [round(random.uniform(10, 500), 2) for _ in range(20)]
}
prodds = pd.DataFrame(product_data)
prodds
```

Out[252…

	Product_Name	Price
0	Quibusdam	302.21
1	Reprehenderit	163.57
2	Quod	161.31
3	Veritatis	223.99
4	Fugiat	165.38
5	Impedit	455.15
6	Dolore	376.98
7	Reiciendis	32.15
8	Ut	428.95
9	Quod	360.71
10	Dignissimos	318.10
11	A	337.64
12	Officiis	273.71
13	Facere	83.88
14	Aut	316.48
15	Iusto	140.54
16	Consequuntur	67.57
17	Aliquid	386.83
18	Culpa	327.74
19	Ea	337.85

In [254…

```
# 15.6. Use the Faker library to create a DataFrame with fake user information,
# City, and Phone Number.
user_data = {
    'Name': [fake.name() for _ in range(20)],
    'City': [fake.city() for _ in range(20)],
    'Phone_Number': [fake.phone_number() for _ in range(20)]
}
userds = pd.DataFrame(user_data)
userds
```

Out[254…

	Name	City	Phone_Number
0	Dhruv Nagy	Sambalpur	09954161613
1	Sai Kalla	Khandwa	04249654250
2	Kabir Dada	Guntur	8268620333
3	Vinaya Ramakrishnan	Satara	+913208219245
4	Imaran Basak	Berhampur	3297046521
5	Urishilla Palla	Singrauli	07553805729
6	Jagat Halder	Nagaon	+917795121702
7	Advaith Boase	Gwalior	+913170332067
8	Ranbir Chakraborty	Munger	+917111943819
9	Jai Jayaraman	Orai	+916307621196
10	Anmol Sinha	Ajmer	08242773702
11	Jhalak Lanka	Bilaspur	7067704113
12	Arin Bakshi	Visakhapatnam	05977629057
13	Ryan Ganguly	Panihati	9927555396
14	Zaid Din	Bhusawal	+919212919448
15	Atharv Bahl	Secunderabad	06094133601
16	Advik Kara	Gaya	+915672684056
17	Benjamin Gulati	Raipur	2348116796
18	Ayushman Sundaram	Bhavnagar	+915767887153
19	Lakshmi Raman	Shivpuri	1026932157

In [256…

```
# 16.                merge - Merging DataFrames: (All)
# 16.1. Merge two DataFrames on a common column Product_ID.
df1 = pd.DataFrame({
    'Product_ID': [101, 102, 103, 104],
    'Product_Name': ['Laptop', 'Phone', 'Tablet', 'Monitor'],
    'Price': [1000, 500, 300, 200]
})

df2 = pd.DataFrame({
    'Product_ID': [102, 103, 104, 105],
    'Category': ['Electronics', 'Electronics', 'Accessories', 'Gaming'],
    'Stock': [50, 30, 20, 10]
})

pd.merge(df1, df2, on='Product_ID')
```

Out[256…

	Product_ID	Product_Name	Price	Category	Stock
0	102	Phone	500	Electronics	50
1	103	Tablet	300	Electronics	30
2	104	Monitor	200	Accessories	20

In [258…

```
#16. 2. Perform a left join between two DataFrames based on Employee_ID.
df_emp1 = pd.DataFrame({
    'Employee_ID': [1, 2, 3, 4],
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Department': ['HR', 'IT', 'Finance', 'Marketing']
})

df_emp2 = pd.DataFrame({
    'Employee_ID': [2, 3, 5, 6],
    'Salary': [60000, 70000, 80000, 90000]
})

pd.merge(df_emp1, df_emp2, on='Employee_ID', how='left')
```

Out[258…

	Employee_ID	Name	Department	Salary
0	1	Alice	HR	NaN
1	2	Bob	IT	60000.0
2	3	Charlie	Finance	70000.0
3	4	David	Marketing	NaN

In [260…

```
#16. 3. Merge two DataFrames, keeping all rows from the left DataFrame and match
# from the right.
pd.merge(df1, df2, on='Product_ID', how='left')
```

Out[260…

	Product_ID	Product_Name	Price	Category	Stock
0	101	Laptop	1000	NaN	NaN
1	102	Phone	500	Electronics	50.0
2	103	Tablet	300	Electronics	30.0
3	104	Monitor	200	Accessories	20.0

In [262…

```
#16. 4. Merge two DataFrames on multiple columns (e.g., City and Age).
df_city1 = pd.DataFrame({
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston'],
    'Age': [25, 30, 35, 40],
    'Population': [8000000, 4000000, 2700000, 2300000]
})

df_city2 = pd.DataFrame({
    'City': ['New York', 'Los Angeles', 'Chicago', 'San Francisco'],
    'Age': [25, 30, 35, 45],
    'Income': [50000, 60000, 55000, 70000]
})
```

```
pd.merge(df_city1, df_city2, on=['City', 'Age'])
```

Out[262…

	City	Age	Population	Income
0	New York	25	8000000	50000
1	Los Angeles	30	4000000	60000
2	Chicago	35	2700000	55000

In [264…

```
# 16.5. Perform an inner join on two DataFrames with a common column Country.
df_country1 = pd.DataFrame({
    'Country': ['USA', 'Canada', 'Germany', 'France'],
    'Capital': ['Washington', 'Ottawa', 'Berlin', 'Paris']
})

df_country2 = pd.DataFrame({
    'Country': ['USA', 'Canada', 'UK', 'Australia'],
    'Currency': ['USD', 'CAD', 'GBP', 'AUD']
})

pd.merge(df_country1, df_country2, on='Country', how='inner')
```

Out[264…

	Country	Capital	Currency
0	USA	Washington	USD
1	Canada	Ottawa	CAD

In [266…

```
# 16.6. Merge two DataFrames and keep all rows, even if there’s no match in the
# DataFrame.
pd.merge(df1, df2, on='Product_ID', how='outer')
```

Out[266…

	Product_ID	Product_Name	Price	Category	Stock
0	101	Laptop	1000.0	NaN	NaN
1	102	Phone	500.0	Electronics	50.0
2	103	Tablet	300.0	Electronics	30.0
3	104	Monitor	200.0	Accessories	20.0
4	105	NaN	NaN	Gaming	10.0

In [268…

```
# 17             Saving and Exporting Data: (All)
#17. 1. Save the DataFrame to a new CSV file.
userds.to_csv("userData.csv", index=False)
```

In [270…

```
# 17.2. Export the DataFrame to an Excel file.
userds.to_excel("userData.xlsx", index=False)
```

In [272…

```
df
```

Out[272…

	Season	Clothing_Item	Sold
0	Summer	T-Shirts	1622
1	Autumn	T-Shirts	2182
2	Winter	woolen	1315
3	Summer	Shorts	707
4	Summer	Formal Shirts	1654
...	...	...	...
145	Summer	Shorts	2205
146	Summer	Shorts	1258
147	Summer	Jeans	1037
148	Winter	Sweatpants	2390
149	Autumn	Sweatpants	2341

150 rows × 3 columns

In [280…

```
#                          COMPULSORY QUESTION:
# 1. Complex Filtering and Aggregation:
# ● Filter the dataset for all rows where the sold is between 0 and 1600 and the
# ● Then, group the filtered data by Restaurant and calculate both the mean and
# ● After that, sort the result in descending order by the sold column.

dataset= df[((df["Sold"]>0)&(df["Sold"]<1200))& (df["Sold"]<1600)]
dsGroup= dataset.groupby("Season")["Sold"].agg(["mean","median"]).reset_index()
dsGroup.sort_values(by="mean", ascending=False)
```

Out[280…

	Season	mean	median
2	Summer	934.250000	1013.0
1	Spring	883.214286	918.5
0	Autumn	846.000000	927.5
3	Winter	750.416667	694.5

In [282…

```
# 2. Advanced String Manipulation and Grouping:
# ● Create a new column season by concatenating the First and Last columns with
# ● Then, split the Clothing_Item column into two separate columns: First and La
# ● Finally, group the dataset by Restaurant and calculate the count of unique R
dfcopy[["First","Last"]]=dfcopy["Clothing_Item"].str.split(" ",n=1,expand=True)
dfcopy
dfcopy.drop(columns="Clothing_Item")
```

Out[282…

	Season	Sold	First	Last
0	Summer	1622	NaN	NaN
1	Autumn	2182	NaN	NaN
2	Winter	1315	NaN	NaN
3	Summer	707	NaN	NaN
4	Summer	1654	Shirts	Shirts
...	...	...	...	...
145	Summer	2205	NaN	NaN
146	Summer	1258	NaN	NaN
147	Summer	1037	NaN	NaN
148	Winter	2390	NaN	NaN
149	Autumn	2341	NaN	NaN

150 rows × 4 columns

In [284…

```
dfcopy["Clothing_Item"]= dfcopy["First"]+" "+dfcopy["Last"]
dfcopy
```

Out[284…

	Season	Sold	Clothing_Item	First	Last
0	Summer	1622	NaN	NaN	NaN
1	Autumn	2182	NaN	NaN	NaN
2	Winter	1315	NaN	NaN	NaN
3	Summer	707	NaN	NaN	NaN
4	Summer	1654	Shirts Shirts	Shirts	Shirts
...	...	...	...	...	...
145	Summer	2205	NaN	NaN	NaN
146	Summer	1258	NaN	NaN	NaN
147	Summer	1037	NaN	NaN	NaN
148	Winter	2390	NaN	NaN	NaN
149	Autumn	2341	NaN	NaN	NaN

150 rows × 5 columns

In [289…

```
dfcopy.groupby(by="Season")["Clothing_Item"].nunique()
```

Out[289…

```
Season
Autumn    2
Spring    2
Summer    2
Winter    2
Name: Clothing_Item, dtype: int64
```

In [ ]:

In [299…

```
# 3. Multiple Data Transformations and Merging:
# • Normalize the sold column to scale the values between 0 and 1.
# • Replace any negative values in the Pictures column with the median of the co
# • Merge the cleaned dataset with another DataFrame (say, product_data) based o
# Reviewer column and display the first 5 rows of the merged dataset.

import pandas as pd

# Create a sample dataset with reviewer ages (renaming columns to match your dat
data = {
    "Clothing_Item": ["T-Shirts", "T-Shirts", "Jackets", "Shorts", "Formal Shirt
    "Avg_Age": [34, 32, 32, 31, 28, 26, 29, 22]  # Changed "Age" to "Avg_Age"
}
ageds = pd.DataFrame(data)

# Copy original dataset
copydf = df.copy()

# Normalize the Units_Sold column
copydf["Normalized  Sold"] = (copydf["Sold"] - copydf["Sold"].min()) / (copydf["

# Merge with the age dataset based on Clothing_Item
mergedds = pd.merge(copydf, ageds, on="Clothing_Item", how="left")

# Display first 5 rows of the merged dataset
print(mergedds.head())
```

```
  Season Clothing_Item  Sold  Normalized  Sold  Avg_Age
0  Summer      T-Shirts  1622          0.560000     34.0
1  Summer      T-Shirts  1622          0.560000     32.0
2  Autumn      T-Shirts  2182          0.843544     34.0
3  Autumn      T-Shirts  2182          0.843544     32.0
4  Winter        woolen  1315          0.404557      NaN
```

In [301…

```
# 1. Create a DataFrame with 'Product ID', 'Product Name', 'Quantity', and 'Pric
# rows of data for different products. Perform an operation to calculate the 'To
# (Quantity * Price) and add it as a new column. Export the modified DataFrame t
# Excel file and compare the original and modified versions. What changes can yo
# observe in the data?
product_data = {
    "Product ID": [fake.unique.random_int(min=1000, max=9999) for _ in range(10)
    "Product Name": [fake.word().capitalize() for _ in range(10)],
    "Quantity": [random.randint(1, 50) for _ in range(10)],
    "Price": [round(random.uniform(10, 1000), 2) for _ in range(10)]
}

Oproddf = pd.DataFrame(product_data)
Mproddf = Oproddf.copy()
Mproddf["Total Value"] = Mproddf["Quantity"] * Mproddf["Price"]

with pd.ExcelWriter("product.xlsx") as writer:
    Oproddf.to_excel(writer, sheet_name="Original Data", index=False)
    Mproddf.to_excel(writer, sheet_name="Modified Data", index=False)

print("Original DataFrame:\n", Oproddf)
print("\nModified DataFrame with 'Total Value':\n", Mproddf)
```

```
Original DataFrame:
    Product ID Product Name  Quantity   Price
0        7144         Dicta        19  868.36
1        9186      Possimus        17  834.33
2        8571     Explicabo         7  836.83
3        5462         Animi         2  603.75
4        1144        Facere        39  828.50
5        9805         Optio         8  870.10
6        2425          Quis        20  114.62
7        7842      Eligendi         3  602.16
8        1767     Occaecati        31   75.55
9        4476          Odit        12  309.61

Modified DataFrame with 'Total Value':
    Product ID Product Name  Quantity   Price  Total Value
0        7144         Dicta        19  868.36     16498.84
1        9186      Possimus        17  834.33     14183.61
2        8571     Explicabo         7  836.83      5857.81
3        5462         Animi         2  603.75      1207.50
4        1144        Facere        39  828.50     32311.50
5        9805         Optio         8  870.10      6960.80
6        2425          Quis        20  114.62      2292.40
7        7842      Eligendi         3  602.16      1806.48
8        1767     Occaecati        31   75.55      2342.05
9        4476          Odit        12  309.61      3715.32
```

In [303…

```
# 2. Create a DataFrame with 'Student ID', 'Name', 'Grade', and 'Score'. Add 10
# data. Perform an operation to assign a 'Pass/Fail' status based on whether the
# above 50 (Pass) or below 50 (Fail). Export the modified DataFrame to an Excel
# show the comparison between the initial file and the updated one. What new col
# changes can you see?

student_data = {
    "Student ID": [fake.unique.random_int(min=1000, max=9999) for _ in range(10)
    "Name": [fake.name() for _ in range(10)],
    "Grade": [random.choice(['A', 'B', 'C', 'D', 'E']) for _ in range(10)],
    "Score": [random.randint(30, 100) for _ in range(10)]  # Random scores betwe
}

original_df = pd.DataFrame(student_data)

modified_df = original_df.copy()
modified_df['Pass/Fail'] = modified_df['Score'].apply(lambda x: 'Pass' if x >= 5

with pd.ExcelWriter("students_comparison.xlsx") as writer:
    original_df.to_excel(writer, sheet_name="Original Data", index=False)
    modified_df.to_excel(writer, sheet_name="Modified Data", index=False)

print("Original DataFrame:\n", original_df)
print("\nModified DataFrame with 'Pass/Fail' Status:\n", modified_df)
```

```
Original DataFrame:
    Student ID              Name Grade  Score
0        1741    Zinal Yohannan     C     39
1        2027      Maanas Hegde     D     32
2        6483       Waida Sidhu     C     71
3        6814  Shivansh Khurana     D     51
4        1708     Chaitaly Muni     D     30
5        2692      Arya Thakkar     A     80
6        9832     Anamika Kakar     C     82
7        1783       Amruta Kata     C     87
8        8912          Aadi Bir     D     58
9        5986        Qushi Dora     D     56

Modified DataFrame with 'Pass/Fail' Status:
    Student ID              Name Grade  Score Pass/Fail
0        1741    Zinal Yohannan     C     39      Fail
1        2027      Maanas Hegde     D     32      Fail
2        6483       Waida Sidhu     C     71      Pass
3        6814  Shivansh Khurana     D     51      Pass
4        1708     Chaitaly Muni     D     30      Fail
5        2692      Arya Thakkar     A     80      Pass
6        9832     Anamika Kakar     C     82      Pass
7        1783       Amruta Kata     C     87      Pass
8        8912          Aadi Bir     D     58      Pass
9        5986        Qushi Dora     D     56      Pass
```

In [305…

```
# 1. Create a CSV file and an Excel file with columns 'Student ID', 'Name', 'Gra
# 'Score'. Add 5 rows of student data to each file. Upload the files to Jupyter
# then perform an operation to assign a 'Pass/Fail' status based on whether the
# above 50. Export the modified CSV and Excel files and compare them with the or
# Explain the differences and the new columns added.

student_data = {
    "Student ID": [fake.unique.random_int(min=1000, max=9999) for _ in range(5)]
    "Name": [fake.name() for _ in range(5)],
    "Grade": [random.choice(['A', 'B', 'C', 'D', 'E']) for _ in range(5)],
    "Score": [random.randint(30, 100) for _ in range(5)]  # Random scores betwee
}

df = pd.DataFrame(student_data)

# Save as CSV and Excel
df.to_csv("students.csv", index=False)
df.to_excel("students.xlsx", sheet_name="Original Data", index=False)

print("Original CSV and Excel files created successfully!")
```

```
Original CSV and Excel files created successfully!
```

In [307…

```
df_csv = pd.read_csv("students.csv")

df_excel = pd.read_excel("students.xlsx", sheet_name="Original Data")

print("Files successfully loaded into DataFrames!")
```

```
Files successfully loaded into DataFrames!
```

In [309…

```
df_csv["Pass/Fail"] = df_csv["Score"].apply(lambda x: "Pass" if x >= 50 else "Fa
df_excel["Pass/Fail"] = df_excel["Score"].apply(lambda x: "Pass" if x >= 50 else

print("Pass/Fail column added successfully!")
```

```
Pass/Fail column added successfully!
```

In [311…

```
df_csv.to_csv("students_modified.csv", index=False)
df_excel.to_excel("students_modified.xlsx", sheet_name="Modified Data", index=Fa

print("Modified CSV and Excel files exported successfully!")
```

```
Modified CSV and Excel files exported successfully!
```

In [313…

```
# 2. Create a CSV file and an Excel file with 'City', 'Country', 'Population', a
# rows of city data to both files. Upload the files to Jupyter Notebook, then pe
# operation to calculate the 'Population Density' (Population / Area) and add it
# column. Export both files after modification and compare the original and upda
# What new information can you observe in the 'Population Density' column?

city_data = {
    "City": ["New York", "Los Angeles", "Chicago", "Houston", "Phoenix"],
    "Country": ["USA", "USA", "USA", "USA", "USA"],
    "Population": [8419600, 3980400, 2716000, 2328000, 1690000],  # Sample popul
    "Area": [783.8, 1302, 589, 1625, 1340]  # Sample area in square km
}

df = pd.DataFrame(city_data)

df.to_csv("cities.csv", index=False)
df.to_excel("cities.xlsx", sheet_name="Original Data", index=False)

print("Original CSV and Excel files created successfully!")
```

```
Original CSV and Excel files created successfully!
```

In [315…

```
df_csv = pd.read_csv("cities.csv")

df_excel = pd.read_excel("cities.xlsx", sheet_name="Original Data")

print("Files successfully loaded into DataFrames!")
```

```
Files successfully loaded into DataFrames!
```

In [317…

```
df_csv["Population Density"] = df_csv["Population"] / df_csv["Area"]
df_excel["Population Density"] = df_excel["Population"] / df_excel["Area"]

print("Population Density column added successfully!")
```

```
Population Density column added successfully!
```

In [319…

```
df_csv.to_csv("cities_modified.csv", index=False)
df_excel.to_excel("cities_modified.xlsx", sheet_name="Modified Data", index=Fals

print("Modified CSV and Excel files exported successfully!")
```

```
Modified CSV and Excel files exported successfully!
```

In [ ]: