

IE 5390 – Assignment - 11

Name: Simran Abhay Sinha

Lab 21

1. Use the Automobile_data.csv dataset.

```
import pandas as pd

# Load the dataset
df = pd.read_csv('Automobile_data.csv')
```

- a. Print the first and last five rows.

```
[11]: # Lab 21_1a
# Print the first and last five rows
print("First 5 rows:")
print(df.head())
print("\nLast 5 rows:")
print(df.tail())

First 5 rows:
   index   company body-style wheel-base length engine-type \
0      0  alfa-romero    convertible     88.6   168.8       dohc
1      1  alfa-romero    convertible     88.6   168.8       dohc
2      2  alfa-romero   hatchback      94.5   171.2       ohcv
3      3        audi      sedan       99.8   176.6       ohc
4      4        audi      sedan       99.4   176.6       ohc

   num-of-cylinders horsepower average-mileage   price
0              four          111             21  13495.0
1              four          111             21  16500.0
2              six           154             19  16500.0
3              four          102             24  13950.0
4              five          115             18  17450.0

Last 5 rows:
   index   company body-style wheel-base length engine-type \
56     81  volkswagen      sedan      97.3   171.7       ohc
57     82  volkswagen      sedan      97.3   171.7       ohc
58     86  volkswagen      sedan      97.3   171.7       ohc
59     87        volvo      sedan     104.3   188.8       ohc
60     88        volvo      wagon     104.3   188.8       ohc

   num-of-cylinders horsepower average-mileage   price
56              four          85             27  7975.0
57              four          52             37  7995.0
58              four          100            26  9995.0
59              four          114            23 12940.0
60              four          114            23 13415.0
```

- b. Drop all rows with missing values (NaN) and update the CSV file. Call the file "Automobile_data_cleaned.csv".

```
[12]: # Lab 21_1b
# Drop rows with missing values and save the cleaned file
df_cleaned = df.dropna()
df_cleaned.to_csv('Automobile_data_cleaned.csv', index=False)
print("Cleaned data saved as Automobile_data_cleaned.csv")
```

Cleaned data saved as Automobile_data_cleaned.csv

- c. Find the company name that has the most expensive car.

```
# Lab 21_1c
# Find the company with the most expensive car
most_expensive_car = df.loc[df['price'].idxmax()]
print("Company with the most expensive car:", most_expensive_car['company'])
```

Company with the most expensive car: mercedes-benz

- d. Print all Toyota car details.

```
# Lab 21_1d
# Print all Toyota car details
toyota_cars = df[df['company'].str.lower() == 'toyota']
display(toyota_cars)
```

index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	price
48	66	toyota	hatchback	95.7	158.7	ohc	four	62	35 5348.0
49	67	toyota	hatchback	95.7	158.7	ohc	four	62	31 6338.0
50	68	toyota	hatchback	95.7	158.7	ohc	four	62	31 6488.0
51	69	toyota	wagon	95.7	169.7	ohc	four	62	31 6918.0
52	70	toyota	wagon	95.7	169.7	ohc	four	62	27 7898.0
53	71	toyota	wagon	95.7	169.7	ohc	four	62	27 8778.0
54	79	toyota	wagon	104.5	187.8	dohc	six	156	19 15750.0

- e. Count total cars per company.

```
# Lab 21_1e
# Count total cars per company
car_counts = df['company'].value_counts()
display(car_counts)
```

company	count
toyota	7
bmw	6
mazda	5
nissan	5
audi	4
mercedes-benz	4
mitsubishi	4
volkswagen	4
alfa-romero	3
chevrolet	3
honda	3
isuzu	3
jaguar	3
porsche	3
dodge	2
volvo	2

Name: count, dtype: int64

f. Find each company's highest priced car.

```
# Lab 21_1f
# Find each company's highest priced car
highest_priced_cars = df.loc[df.groupby('company')['price'].idxmax()]
display(highest_priced_cars[['company', 'price']])
```

	company	price
1	alfa-romero	16500.0
6	audi	18920.0
11	bmw	41315.0
15	chevrolet	6575.0
16	dodge	6377.0
19	honda	12945.0
21	isuzu	6785.0
26	jaguar	36000.0
31	mazda	18344.0
35	mercedes-benz	45400.0
39	mitsubishi	8189.0
44	nissan	13499.0
46	porsche	37028.0
54	toyota	15750.0
58	volkswagen	9995.0
60	volvo	13415.0

g. Find the average mileage of each car company.

```
# Lab 21_1g
# Find the average mileage of each company
average_mileage = df.groupby('company')['average-mileage'].mean()
display(average_mileage)
```

company	
alfa-romero	20.333333
audi	20.000000
bmw	19.000000
chevrolet	41.000000
dodge	31.000000
honda	26.333333
isuzu	33.333333
jaguar	14.333333
mazda	28.000000
mercedes-benz	18.000000
mitsubishi	29.500000
nissan	31.400000
porsche	17.000000
toyota	28.714286
volkswagen	31.750000
volvo	23.000000

Name: average-mileage, dtype: float64

h. Sort all cars by Price column

```
# Lab 21_1h
# Sort all cars by price
sorted_cars = df.sort_values(by='price', ascending=True)
display(sorted_cars[['company', 'price']])
```

	company	price
13	chevrolet	5151.0
27	mazda	5195.0
48	toyota	5348.0
36	mitsubishi	5389.0
28	mazda	6095.0
...
11	bmw	41315.0
35	mercedes-benz	45400.0
22	isuzu	NaN
23	isuzu	NaN
47	porsche	NaN

61 rows × 2 columns

i. Create two data frames using the given dictionaries:

```
GermanCars = {'Company': ['Ford', 'Mercedes', 'BMW', 'Audi'], 'Price': [23845, 171995, 135925 , 71400]}
```

```
japaneseCars = {'Company': ['Toyota', 'Honda', 'Nissan', 'Mitsubishi '], 'Price': [29995, 23600, 61500 , 58900]}
```

- Create a key for each datframe.

- Concatenate the two dataframes that you created based on the key.

```
# Lab 21_1i
# Create and concatenate German and Japanese car dataframes
GermanCars = pd.DataFrame({'Company': ['Ford', 'Mercedes', 'BMW', 'Audi'], 'Price': [23845, 171995, 135925 , 71400]})
japaneseCars = pd.DataFrame({'Company': ['Toyota', 'Honda', 'Nissan', 'Mitsubishi'], 'Price': [29995, 23600, 61500 , 58900]})
combined_cars = pd.concat([GermanCars, japaneseCars], keys=['German', 'Japanese'])
display(combined_cars)
```

	Company	Price
German	0 Ford	23845
	1 Mercedes	171995
	2 BMW	135925
	3 Audi	71400
Japanese	0 Toyota	29995
	1 Honda	23600
	2 Nissan	61500
	3 Mitsubishi	58900

j. Create two data frames using the given dictionaries:

```
Car_Price = {'Company': ['Toyota', 'Honda', 'BMV', 'Audi'], 'Price': [23845, 17995, 135925 , 71400]}
```

```
car_Horsepower = {'Company': ['Toyota', 'Honda', 'BMV', 'Audi'], 'horsepower': [141, 80, 182 , 160]}
```

- Merge the two dataframes

```
1]: # Lab 21_1j
# Merge Car_Price and car_Horsepower dataframes
Car_Price = pd.DataFrame({'Company': ['Toyota', 'Honda', 'BMV', 'Audi'], 'Price': [23845, 17995, 135925 , 71400]})
car_Horsepower = pd.DataFrame({'Company': ['Toyota', 'Honda', 'BMV', 'Audi'], 'horsepower': [141, 80, 182 , 160]})
merged_cars = pd.merge(Car_Price, car_Horsepower, on='Company')
display(merged_cars)
```

	Company	Price	horsepower
0	Toyota	23845	141
1	Honda	17995	80
2	BMV	135925	182
3	Audi	71400	160

2. Use the banklist.csv file. The dataset compiled by FDIC contains information about banks that have failed and acquired by other banks.

```
[25]: #Lab21_2
import pandas as pd

# Load the dataset
banklist_df = pd.read_csv('banklist.csv')
```

a. What are the column names?

```
#Lab21_2a
print("Column names:", banklist_df.columns)

Column names: Index(['Bank Name', 'City', 'ST', 'CERT', 'Acquiring Institution',
       'Closing Date', 'Updated Date'],
      dtype='object')
```

b. How many States (ST) are represented in this dataset?

```
#Lab21_2b
# How many States (ST) are represented in this dataset?
num_unique_states = banklist_df['ST'].nunique()
print("Number of unique states:", num_unique_states)
```

Number of unique states: 44

c. Get an array of all the states in the dataset.

```
#Lab21_2c
# Get an array of all the states in the dataset.
unique_states = banklist_df['ST'].unique()
print("Array of all states:", unique_states)

Array of all states: ['AR' 'GA' 'PA' 'TN' 'WI' 'WA' 'CO' 'IL' 'PR' 'FL' 'MN' 'CA' 'MD' 'OK'
 'OH' 'SC' 'VA' 'ID' 'TX' 'CT' 'AZ' 'NV' 'NC' 'KY' 'MO' 'KS' 'AL' 'NJ'
 'MI' 'IN' 'LA' 'IA' 'UT' 'NE' 'MS' 'NM' 'OR' 'NY' 'MA' 'SD' 'WY' 'WV'
 'NH' 'HI']
```

d. What are the top 5 states with the most failed banks?

```
#Lab21_2d
# What are the top 5 states with the most failed banks?
top_5_states_failed_banks = banklist_df['ST'].value_counts().head(5)
print("Top 5 states with most failed banks:\n", top_5_states_failed_banks)
```

ST	count
GA	93
FL	75
IL	66
CA	41
MN	23

Name: count, dtype: int64

e. What are the top 5 acquiring institutions?

```
#Lab21_2e
# What are the top 5 acquiring institutions?
top_5_acquiring_institutions = banklist_df['Acquiring Institution'].value_counts().head(5)
print("Top 5 acquiring institutions:\n", top_5_acquiring_institutions)
```

```
Top 5 acquiring institutions:
Acquiring Institution
No Acquirer           31
State Bank and Trust Company   12
Ameris Bank            10
First-Citizens Bank & Trust Company    9
U.S. Bank N.A.          9
Name: count, dtype: int64
```

f. How many banks has the State Bank of Texas acquired? How many of them were actually in Texas?

```
#Lab21_2f
# How many banks has the State Bank of Texas acquired? How many of them were in Texas?
state_bank_of_texas_acquired = banklist_df[banklist_df['Acquiring Institution'] == 'State Bank of Texas']
num_acquired_by_state_bank_of_texas = state_bank_of_texas_acquired.shape[0]
num_in_texas = state_bank_of_texas_acquired[state_bank_of_texas_acquired['ST'] == 'TX'].shape[0]
print("Number of banks acquired by State Bank of Texas:", num_acquired_by_state_bank_of_texas)
print("Number of these banks in Texas:", num_in_texas)
```

```
Number of banks acquired by State Bank of Texas: 2
Number of these banks in Texas: 1
```

g. What is the most common city in California for a bank to fail in?

```
#Lab21_2g
# What is the most common city in California for a bank to fail in?
california_failed_banks = banklist_df[banklist_df['ST'] == 'CA']
most_common_city_in_ca = california_failed_banks['City'].value_counts().idxmax()
print("Most common city for a bank to fail in California:", most_common_city_in_ca)
```

```
Most common city for a bank to fail in California: Los Angeles
```

h. How many failed banks don't have the word "Bank" in their name?

```
#Lab21_2h
# How many failed banks don't have the word 'Bank' in their name?
banks_without_bank_word = banklist_df[~banklist_df['Bank Name'].str.contains('Bank', case=False, na=False)]
num_banks_without_bank_word = banks_without_bank_word.shape[0]
print("Number of failed banks without 'Bank' in their name:", num_banks_without_bank_word)
```

```
Number of failed banks without 'Bank' in their name: 10
```

i. How many bank names start with the letter 's' ?

```
#Lab21_2i
# How many bank names start with the letter 's'?
banks_starting_with_s = banklist_df[banklist_df['Bank Name'].str.lower().str.startswith('s', na=False)]
num_banks_starting_with_s = banks_starting_with_s.shape[0]
print("Number of bank names starting with 'S':", num_banks_starting_with_s)
```

```
Number of bank names starting with 'S': 53
```

j. How many CERT values are above 20000 ?

```
#Lab21_2j
# How many CERT values are above 20000?
cert_above_20000 = banklist_df[banklist_df['CERT'] > 20000].shape[0]
print("Number of CERT values above 20000:", cert_above_20000)
```

Number of CERT values above 20000: 415

k. How many bank names consist of just two words? (e.g. "First Bank" , "Bank Georgia")

```
#Lab21_2k
# How many bank names consist of just two words?
banks_two_words = banklist_df['Bank Name'].apply(lambda x: len(str(x).split()) == 2)
num_banks_two_words = banks_two_words.sum()
print("Number of bank names with just two words:", num_banks_two_words)
```

Number of bank names with just two words: 113

3. Gen AI Prompt Engineering:

Use the Gen_AI_sales_dataset.csv file.

a. Data Selection and Filtering:

- Filter the DataFrame to include only transactions from the "Electronics" product category.
- Create a DataFrame containing only the rows where total_sales > 1000.

b. Adding New Columns:

- Add a new column called revenue_per_unit that is the ratio of total_sales to units_sold.
- Create a new column month that extracts the month from the date column.

c. Handling Missing Data:

- Identify any missing values in the DataFrame. Replace missing unit_price values with the average unit price across all transactions.
- Drop any rows where units_sold is zero or less.

d. Sorting and Ranking:

- Sort the DataFrame by total_sales in descending order. Select the top 10 rows.
- For each store, rank the products by total_sales in descending order, and add a new column rank_in_store to store this ranking.

e. Aggregation:

- Group the data by store_id and calculate:

i. The total number of units sold per store.

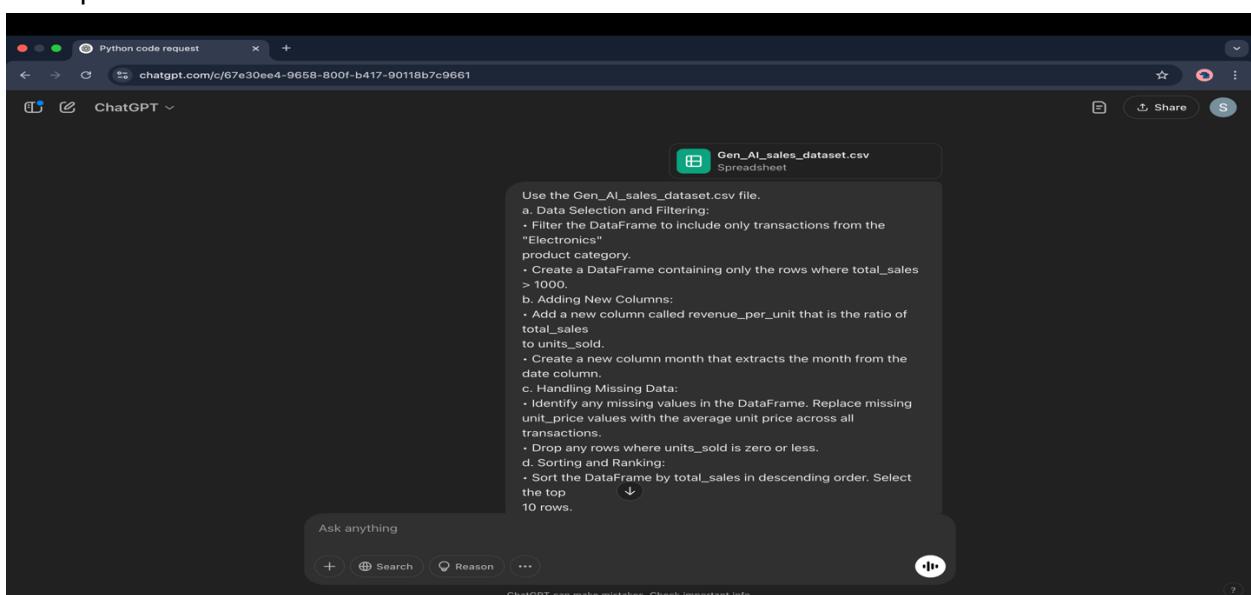
ii. The average unit price per store.

iii. The total sales per store.

- After performing the grouping, reset the index.

Bot Used: ChatGPT

Prompt:



Gen AI:

Python code request chatgpt.com/c/67e30ee4-9658-800f-b417-90118b7c9661

Process Sales Data

```

1 import pandas as pd
2
3 # Load the dataset
4 file_path = "/mnt/data/Gen_AI_sales_dataset.csv"
5 df = pd.read_csv(file_path)
6
7 # a. Data Selection and Filtering
8
9 # Filter for "Electronics" category
10 electronics_df = df[df["product_category"] == "Electronics"].copy()
11
12 # Filter rows where total_sales > 1000
13 high_sales_df = electronics_df[electronics_df["total_sales"] > 1000].copy()
14
15 # b. Adding New Columns
16
17 # Add revenue_per_unit column
18 high_sales_df.loc[:, "revenue_per_unit"] = high_sales_df["total_sales"] / high_sales_df["units_sold"]
19
20 # Ensure date column is properly converted to datetime format
21 high_sales_df.loc[:, "date"] = pd.to_datetime(high_sales_df["date"], errors='coerce')
22
23 # Check if conversion was successful
24 if high_sales_df["date"].isna().sum() > 0:
25     print("Warning: Some date values could not be converted and are NaT")
26
27 # Drop rows where date conversion failed
28 high_sales_df = high_sales_df.dropna(subset=["date"])
29
30 # Extract month from date column
31 high_sales_df.loc[:, "month"] = high_sales_df["date"].dt.month
32
33 # c. Handling Missing Data
34
35 # Replace missing unit_price values with the average unit price
36 avg_unit_price = high_sales_df["unit_price"].mean()
37 high_sales_df.loc[:, "unit_price"] = high_sales_df["unit_price"].fillna(avg_unit_price)
38
39 # Drop rows where units_sold is zero or less
40 high_sales_df = high_sales_df[high_sales_df["units_sold"] > 0].copy()
41
42 # d. Sorting and Ranking
43
44 # Sort by total_sales in descending order and select top 10 rows
45 top_10_sales = high_sales_df.sort_values(by="total_sales", ascending=False).head(10)
46
47 # Rank products by total_sales within each store
48 high_sales_df.loc[:, "rank_in_store"] = high_sales_df.groupby("store_id")["total_sales"].rank()
49
50 # e. Aggregation
51
52 # Group by store_id and calculate required aggregations
53 store_aggregates = high_sales_df.groupby("store_id").agg(
54     total_units_sold=("units_sold", "sum"),
55     avg_unit_price=("unit_price", "mean"),
56     total_sales=("total_sales", "sum")
57 ).reset_index()
58
59 # Display results
60 print(high_sales_df.dtypes) # Debugging step to check column types
61 print(top_10_sales)
62 print(store_aggregates.head())

```

I'll process the dataset according to your requirements. Let's start by loading the data and inspecting its structure.

The dataset contains 100,000 entries with the following columns: `store_id`, `date`, `product_id`, `product_category`, `units_sold`, `unit_price`, and `total_sales`. Now, I'll proceed with the requested data manipulations. [d]

Give me python code

Here's the complete Python code that performs all the required data operations:

Process Sales Data

This code filters, processes, and aggregates the sales dataset as required. Let me know if you need any modifications or explanations! ↴

Ask anything

ChatGPT can make mistakes. Check important info.

I'll process the dataset according to your requirements. Let's start by loading the data and inspecting its structure.

The dataset contains 100,000 entries with the following columns: `store_id`, `date`, `product_id`, `product_category`, `units_sold`, `unit_price`, and `total_sales`. Now, I'll proceed with the requested data manipulations. [d]

Give me python code

Here's the complete Python code that performs all the required data operations:

Process Sales Data

This code filters, processes, and aggregates the sales dataset as required. Let me know if you need any modifications or explanations! ↴

Ask anything

ChatGPT can make mistakes. Check important info.

Jupyter:

```

[50]: #Lab21_3 Genai
import pandas as pd

# Load the dataset
file_path = "Gen_AI_sales_dataset.csv"
df = pd.read_csv(file_path)

# a. Data Selection and Filtering

# Filter for "Electronics" category
electronics_df = df[df["product_category"] == "Electronics"].copy()

# Filter rows where total_sales > 1000
high_sales_df = electronics_df[electronics_df["total_sales"] > 1000].copy()

# b. Adding New Columns

# Add revenue_per_unit column
high_sales_df.loc[:, "revenue_per_unit"] = high_sales_df["total_sales"] / high_sales_df["units_sold"]

# Ensure date column is properly converted to datetime format
high_sales_df["date"] = pd.to_datetime(high_sales_df["date"], errors='coerce')

# Drop rows where date conversion failed
high_sales_df = high_sales_df.dropna(subset=["date"])

# Confirm date is in datetime format before extracting month
if not pd.api.types.is_datetime64_any_dtype(high_sales_df["date"]):
    print("Error: Date column is not in datetime format after conversion")
else:
    high_sales_df["month"] = high_sales_df["date"].dt.month

# c. Handling Missing Data

# Replace missing unit_price values with the average unit price
avg_unit_price = high_sales_df["unit_price"].mean()
high_sales_df["unit_price"] = high_sales_df["unit_price"].fillna(avg_unit_price)

```



```

# c. Handling Missing Data

# Replace missing unit_price values with the average unit price
avg_unit_price = high_sales_df["unit_price"].mean()
high_sales_df.loc[:, "unit_price"] = high_sales_df["unit_price"].fillna(avg_unit_price)

# Drop rows where units_sold is zero or less
high_sales_df = high_sales_df[high_sales_df["units_sold"] > 0].copy()

# d. Sorting and Ranking

# Sort by total_sales in descending order and select top 10 rows
top_10_sales = high_sales_df.sort_values(by="total_sales", ascending=False).head(10)

# Rank products by total_sales within each store
high_sales_df.loc[:, "rank_in_store"] = high_sales_df.groupby("store_id")["total_sales"].rank(method="dense", ascending=False)

# e. Aggregation

# Group by store_id and calculate required aggregations
store_aggregates = high_sales_df.groupby("store_id").agg(
    total_units_sold=("units_sold", "sum"),
    avg_unit_price=("unit_price", "mean"),
    total_sales=("total_sales", "sum")
).reset_index()

# Display results
print(high_sales_df.dtypes) # Debugging step to check column types
print(top_10_sales)
print(store_aggregates.head())

```

store_id	int64
date	datetime64[ns]
product_id	int64
product_category	object
units_sold	int64
unit_price	float64
total_sales	float64

The screenshot shows a Jupyter Notebook interface with multiple tabs at the top: 'Final Project_Due_4-9', 'Week11_Due_4-1', 'Home', 'Week11' (active), and '+'. The main area displays a pandas DataFrame with the following structure:

	store_id	date	product_id	product_category	units_sold	unit_price	total_sales	revenue_per_unit	month
28848	30	2025-12-02	120	Electronics	200				12
63505	1	2025-05-23	104	Electronics	200				
23133	15	2025-06-20	118	Electronics	198				
94231	42	2025-03-08	110	Electronics	199				
15874	21	2025-04-22	112	Electronics	199				
20133	46	2025-02-24	133	Electronics	198				
10795	48	2025-01-04	143	Electronics	198				
75714	44	2025-06-27	137	Electronics	198				
75874	19	2025-03-14	150	Electronics	198				
29070	42	2025-11-17	109	Electronics	199				

Below this, there are two additional sections of data:

	store_id	total_units_sold	avg_unit_price	total_sales
0	1	65835	345.864057	22355094.18
1	2	70451	339.111400	23842039.78
2	3	68064	339.568563	23096447.22
3	4	66811	344.172762	22982318.64
4	5	71152	329.926730	22748954.67

Lab 22

1. Load the company_sales_data.csv file. Explore and understand what the dataset consists of. Do the following:

```
#Lab22_1
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
file_path = 'company_sales_data.csv'
df = pd.read_csv(file_path)
```

- a. Create a line plot that shows total profit per month. Label the axes and provide a title for the plot.

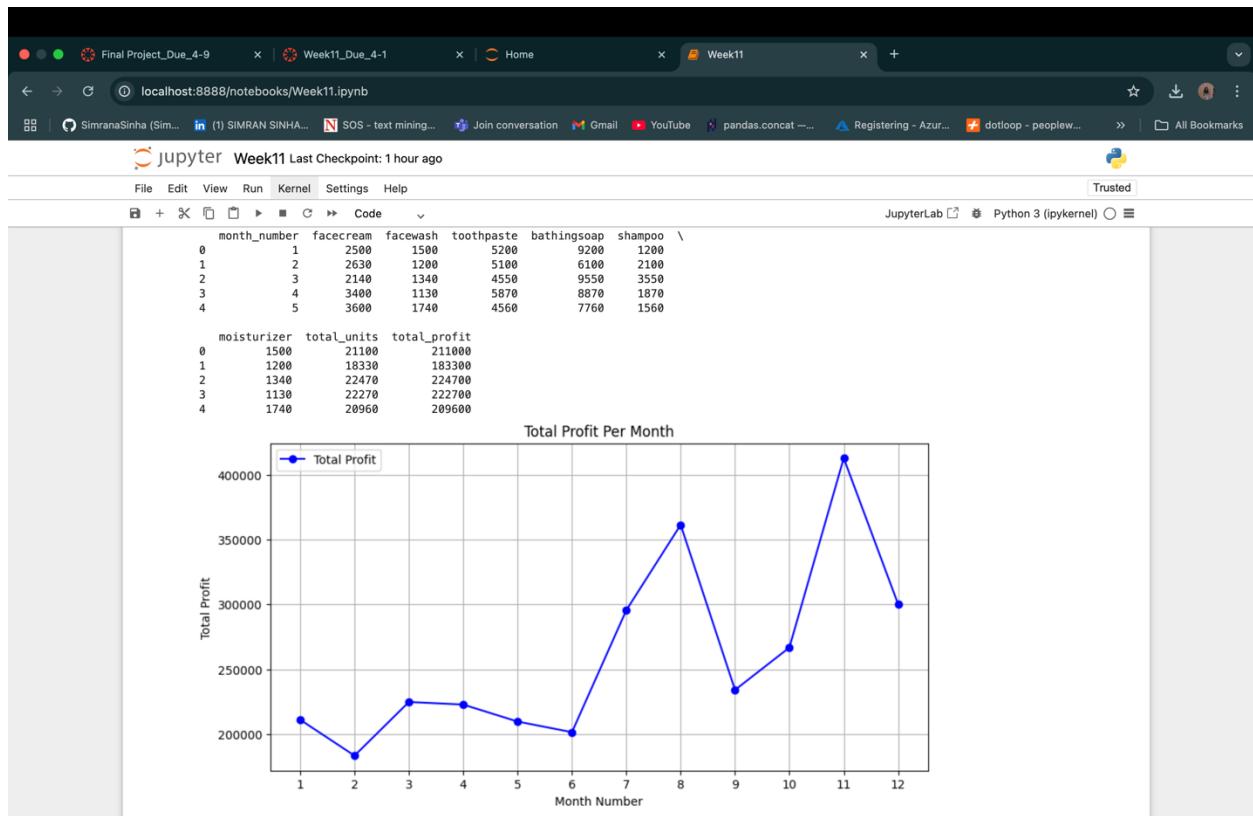
```
#Lab22_1a
# Display basic information about the dataset
print(df.info())
print(df.head()) # Show first few rows to understand structure

# Assuming 'total_profit' is the column representing profit and 'month_number' represents months
plt.figure(figsize=(10,5))
plt.plot(df['month_number'], df['total_profit'], marker='o', linestyle='-', color='b', label='Total Profit')

# Labeling
plt.xlabel('Month Number')
plt.ylabel('Total Profit')
plt.title('Total Profit Per Month')
plt.xticks(df['month_number']) # Ensure month numbers are displayed correctly
plt.legend()
plt.grid(True)

# Show the plot
plt.show()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   month_number  12 non-null    int64  
 1   facecream     12 non-null    int64  
 2   facewash      12 non-null    int64  
 3   toothpaste    12 non-null    int64  
 4   bathingsoap   12 non-null    int64  
 5   shampoo       12 non-null    int64  
 6   moisturizer   12 non-null    int64  
 7   total_units   12 non-null    int64  
 8   total_profit  12 non-null    int64  
dtypes: int64(9)
```



b. Redo part a., but this time include the following style properties:

- Line Style dotted and Line-color should be red
- Show legend at the lower right location.
- X label name = Month Number
- Y label name = Sold units number
- Add a circle marker.
- Line marker color as read
- Line width should be 3

```
#Lab22_1b
# Display basic information about the dataset
print(df.info())
print(df.head()) # Show first few rows to understand structure

# Assuming 'total_profit' is the column representing profit and 'month_number' represents months
plt.figure(figsize=(10,5))
plt.plot(df['month_number'], df['total_profit'], marker='o', linestyle=':', color='r', linewidth=3, label='Profit data of last year')

# Labeling
plt.xlabel('Month Number')
plt.ylabel('Sold units number')
plt.title('Company Sales data of last year')
plt.xticks(df['month_number']) # Ensure month numbers are displayed correctly
plt.legend(loc='lower right')
plt.grid(True)

# Show the plot
plt.show()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   month_number    12 non-null   int64  
 1   facecream       12 non-null   int64  
 2   facewash        12 non-null   int64  
 3   toothpaste      12 non-null   int64  
 4   bathingsoap     12 non-null   int64  
 5   shampoo         12 non-null   int64  
 6   moisturizer     12 non-null   int64  
 7   total_units     12 non-null   int64  
 8   total_profit    12 non-null   int64  
 ..  .....
```

```

month_number facecream facewash toothpaste bathingsoap shampoo \
0 1 2500 1500 5200 9200 1200 \
1 2 2630 1200 5100 6100 2100 \
2 3 2140 1340 4550 9550 3550 \
3 4 3400 1130 5870 8870 1870 \
4 5 3600 1740 4560 7760 1560 \
moisturizer total_units total_profit
0 1500 21100 211000
1 1200 18330 183300
2 1340 22470 224700
3 1130 22270 222700
4 1740 20960 209600

```

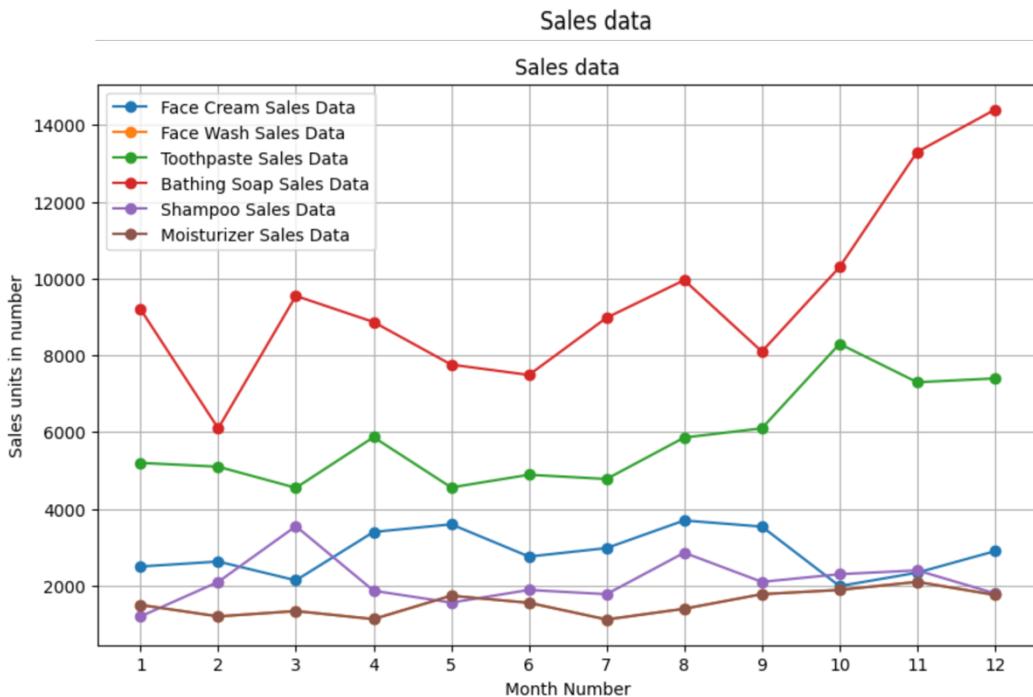


- c. Display the number of units sold per month for each product using multiline plots.
 (i.e., Separate Plotline for each product on a single plot).

```
#Lab22_1c
# Multiline plot for units sold per product
plt.figure(figsize=(10,6))
plt.plot(df['month_number'], df['facecream'], marker='o', linestyle='-', label='Face Cream Sales Data')
plt.plot(df['month_number'], df['facewash'], marker='o', linestyle='-', label='Face Wash Sales Data')
plt.plot(df['month_number'], df['toothpaste'], marker='o', linestyle='-', label='Toothpaste Sales Data')
plt.plot(df['month_number'], df['bathingsoap'], marker='o', linestyle='-', label='Bathing Soap Sales Data')
plt.plot(df['month_number'], df['shampoo'], marker='o', linestyle='-', label='Shampoo Sales Data')
plt.plot(df['month_number'], df['moisturizer'], marker='o', linestyle='-', label='Moisturizer Sales Data')

# Labeling
plt.xlabel('Month Number')
plt.ylabel('Sales units in number')
plt.title('Sales data')
plt.xticks(df['month_number'])
plt.legend()
plt.grid(True)

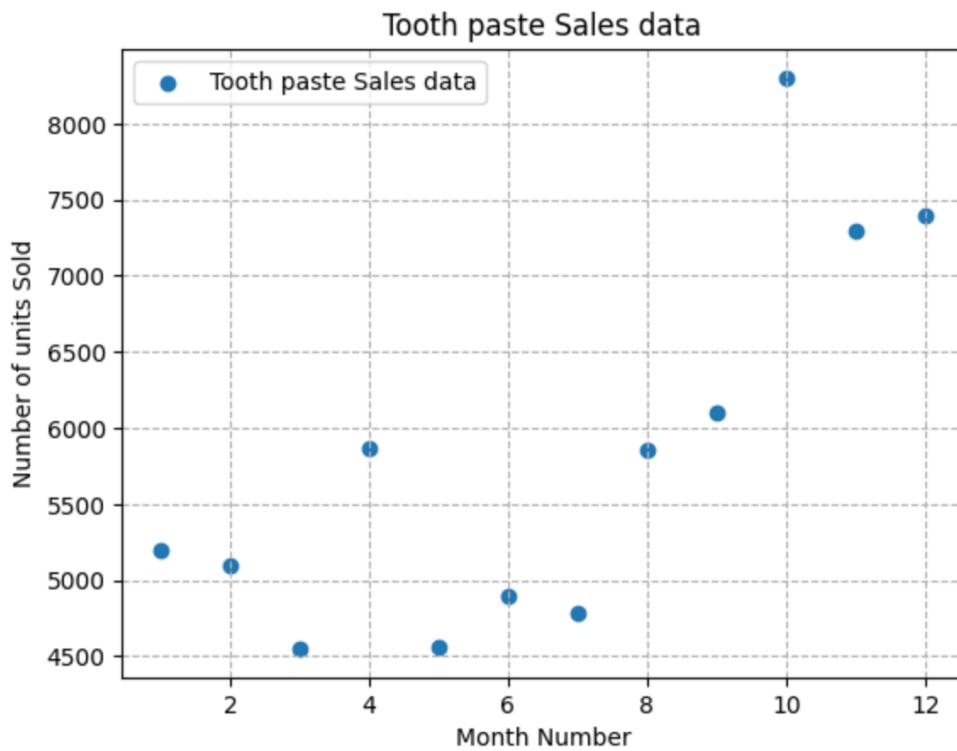
# Show the plot
plt.show()
```



- d. Create a scatter plot of toothpaste sales by each month. Also, add a grid in the plot.
The gridline style should be “-“.

```
[1]: #Lab22_1d
# Load the data
data = {
    'month_number': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
    'toothpaste': [5200, 5100, 4550, 5870, 4560, 4890, 4780, 5860, 6100, 8300, 7300, 7400]
}

# Create a scatter plot
plt.scatter(data['month_number'], data['toothpaste'], label='Tooth paste Sales data')
plt.title('Tooth paste Sales data')
plt.xlabel('Month Number')
plt.ylabel('Number of units Sold')
plt.grid(True, linestyle='--') # Dashed gridlines
plt.legend()
plt.show()
```

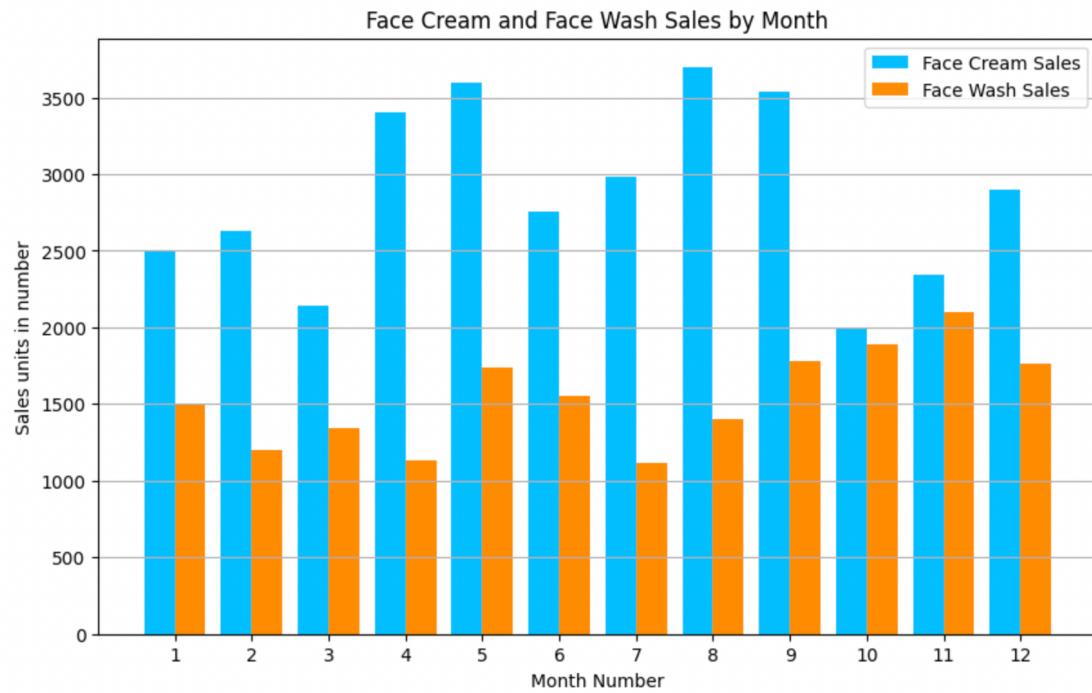


e. Create bar plots for face cream sales and face wash sales by month on a single plot. The bar plot should display the number of units sold per month for each product.

```
#Lab22_1e
# Bar plot for Face Cream and Face Wash Sales
plt.figure(figsize=(10,6))
bar_width = 0.4
plt.bar(df['month_number'] - bar_width/2, df['facecream'], width=bar_width, label='Face Cream Sales', color='deepskyblue')
plt.bar(df['month_number'] + bar_width/2, df['facewash'], width=bar_width, label='Face Wash Sales', color='darkorange')

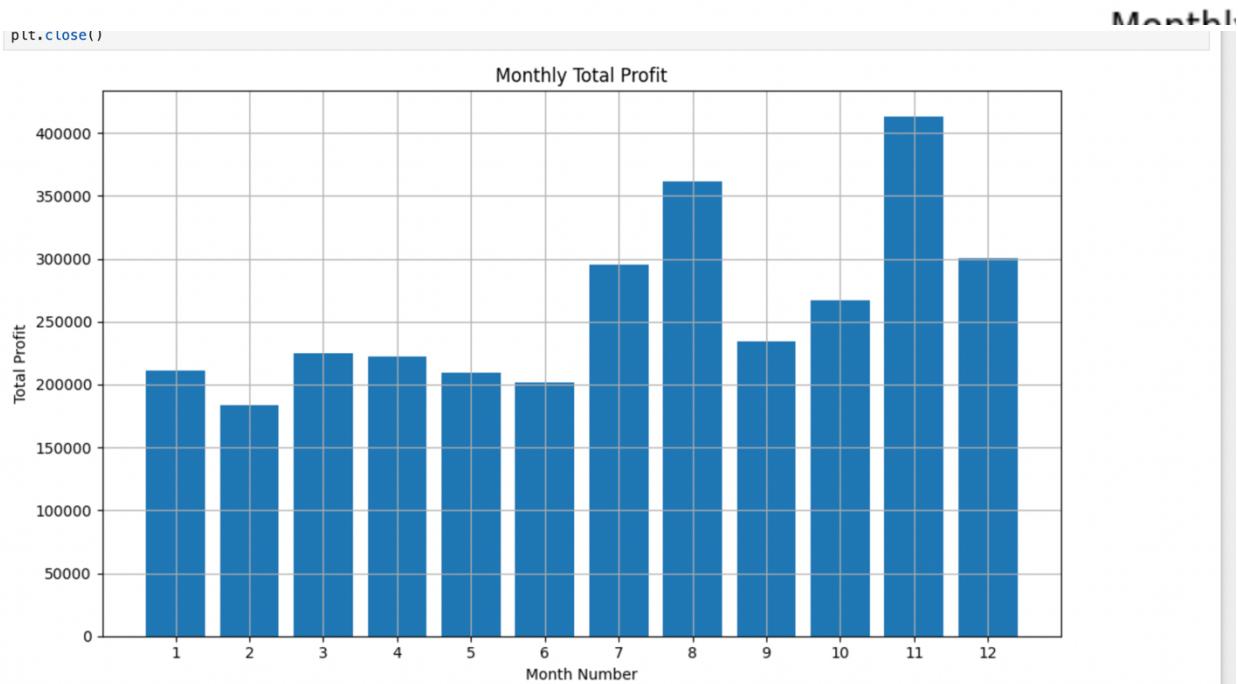
# Labeling
plt.xlabel('Month Number')
plt.ylabel('Sales units in number')
plt.title('Face Cream and Face Wash Sales by Month')
plt.xticks(df['month_number'])
plt.legend()
plt.grid(axis='y')

# Show the plot
plt.show()
```



f. Create a histogram that shows the total profit of each month.

```
[4]: #Lab22_1f
plt.figure(figsize=(10, 6))
plt.bar(df['month_number'], df['total_profit'])
plt.xlabel('Month Number')
plt.ylabel('Total Profit')
plt.title('Monthly Total Profit')
plt.xticks(df['month_number'])
plt.grid(True)
plt.tight_layout()
plt.show()
plt.close()
```

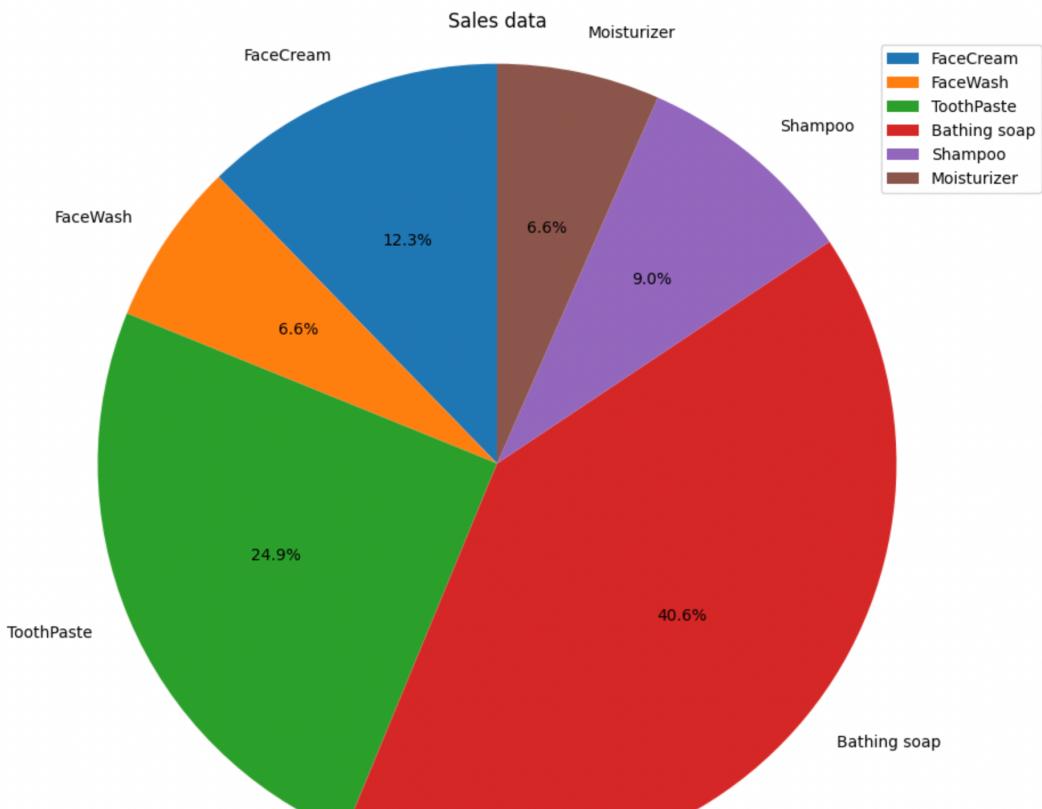


g. Create a pie chart that displays the Number of units sold per year for each product as a percentage.

```

]: #Lab22_1g
# Calculate total annual sales for each product
facecream_sales = df['facecream'].sum()
facewash_sales = df['facewash'].sum()
toothpaste_sales = df['toothpaste'].sum()
bathingsoap_sales = df['bathingsoap'].sum()
shampoo_sales = df['shampoo'].sum()
moisturizer_sales = df['moisturizer'].sum()
# Combine the sales data and labels
sales = [facecream_sales, facewash_sales, toothpaste_sales, bathingsoap_sales, shampoo_sales, moisturizer_sales]
products = ['FaceCream', 'FaceWash', 'ToothPaste', 'Bathing soap', 'Shampoo', 'Moisturizer']
# Calculate percentages
total = sum(sales)
percentages = [round((x/total)*100, 1) for x in sales]
plt.figure(figsize=(10, 8))
plt.pie(sales, labels=products, autopct='%.1f%%', startangle=90)
plt.axis('equal')
plt.title('Sales data')
plt.legend(products)
plt.tight_layout()
plt.show()
plt.close()

```



h. Create line plots of bathing soap sales and of facewash sales by month as subplots

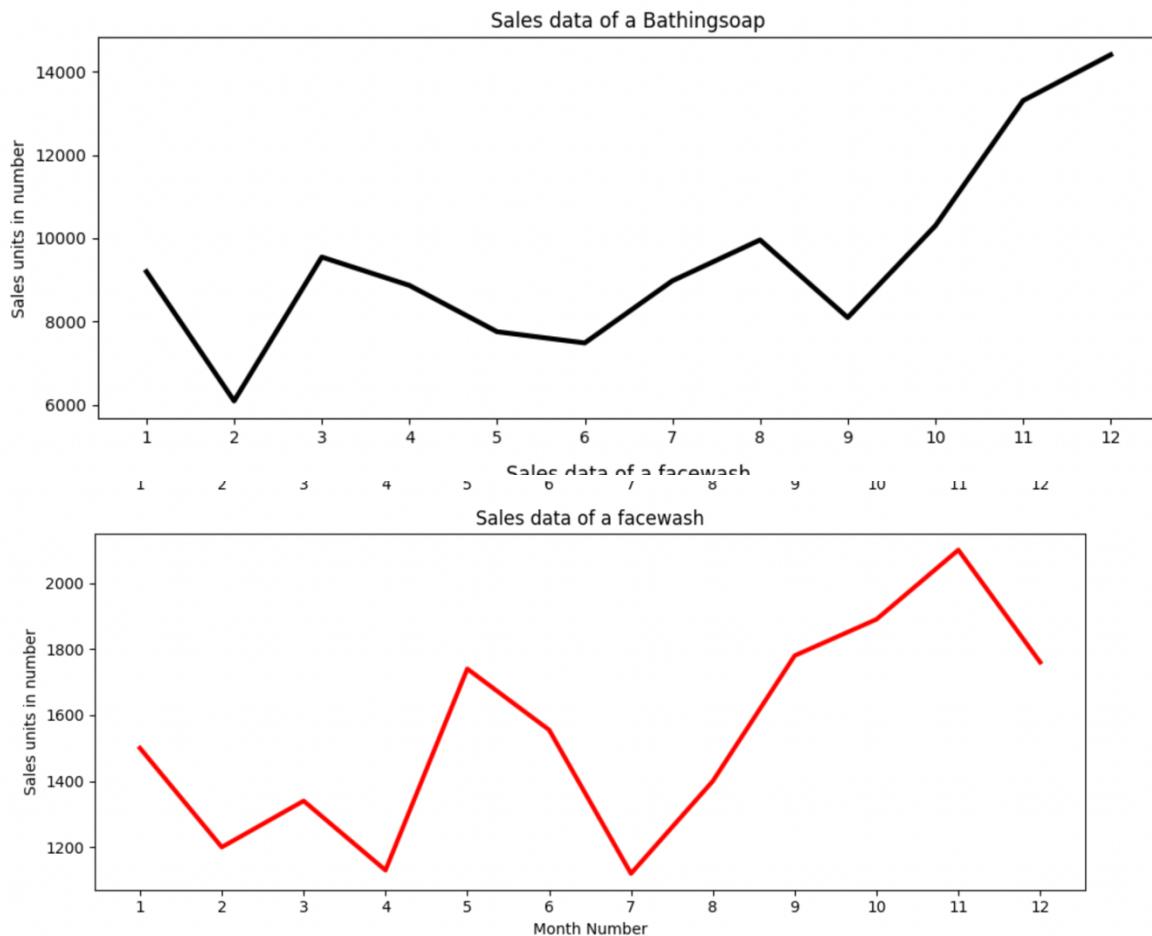
```
#Lab22_1h
plt.figure(figsize=(10, 8))

# First subplot - Bathing soap
plt.subplot(2, 1, 1)
plt.plot(df['month_number'], df['bathingsoap'], color='black', linewidth=3)
plt.title('Sales data of a Bathingsoap')
plt.ylabel('Sales units in number')
plt.xticks(df['month_number'])
plt.grid(False)

# Second subplot - Facewash
plt.subplot(2, 1, 2)
plt.plot(df['month_number'], df['facewash'], color='red', linewidth=3)
plt.title('Sales data of a facewash')
plt.xlabel('Month Number')
plt.ylabel('Sales units in number')
plt.xticks(df['month_number'])
plt.grid(False)

plt.tight_layout()
plt.show()
plt.close()

plt.show()
plt.close()
```



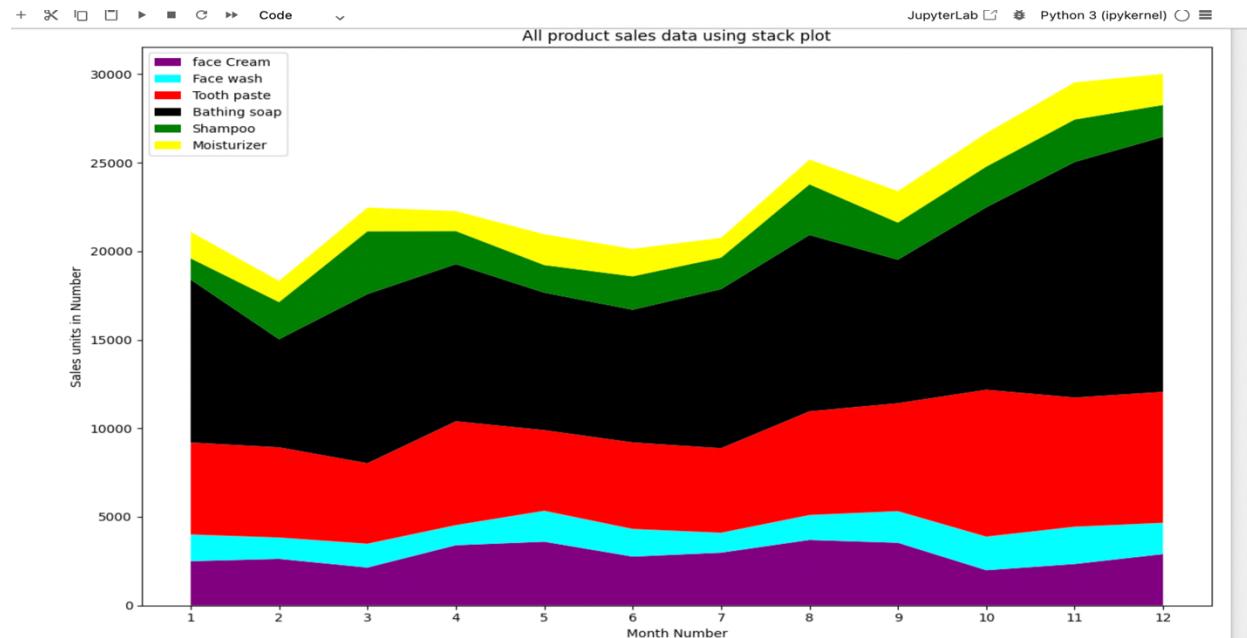
i. Create a stack plot that shows all product sales by month.

```
#Lab22_11
plt.figure(figsize=(12, 8))

months = df['month_number']
facecream = df['facecream']
facewash = df['facewash']
toothpaste = df['toothpaste']
bathingsoap = df['bathingsoap']
shampoo = df['shampoo']
moisturizer = df['moisturizer']

plt.stackplot(months,
              facecream,
              facewash,
              toothpaste,
              bathingsoap,
              shampoo,
              moisturizer,
              colors=['purple', 'cyan', 'red', 'black', 'green', 'yellow'])

plt.xlabel('Month Number')
plt.ylabel('Sales units in Number')
plt.title('All product sales data using stack plot')
plt.legend(['face Cream', 'Face wash', 'Tooth paste', 'Bathing soap', 'Shampoo', 'Moisturizer'], loc='upper left')
plt.xticks(df['month_number'])
plt.tight_layout()
plt.show()
plt.close()
```



2. The Titanic dataset comes inbuilt with the Seaborn package. To load it, import the Seaborn library and type: titanic = sns.load_dataset('titanic')
Once the data is loaded, do the following:

LAB 22 QUESTION 2

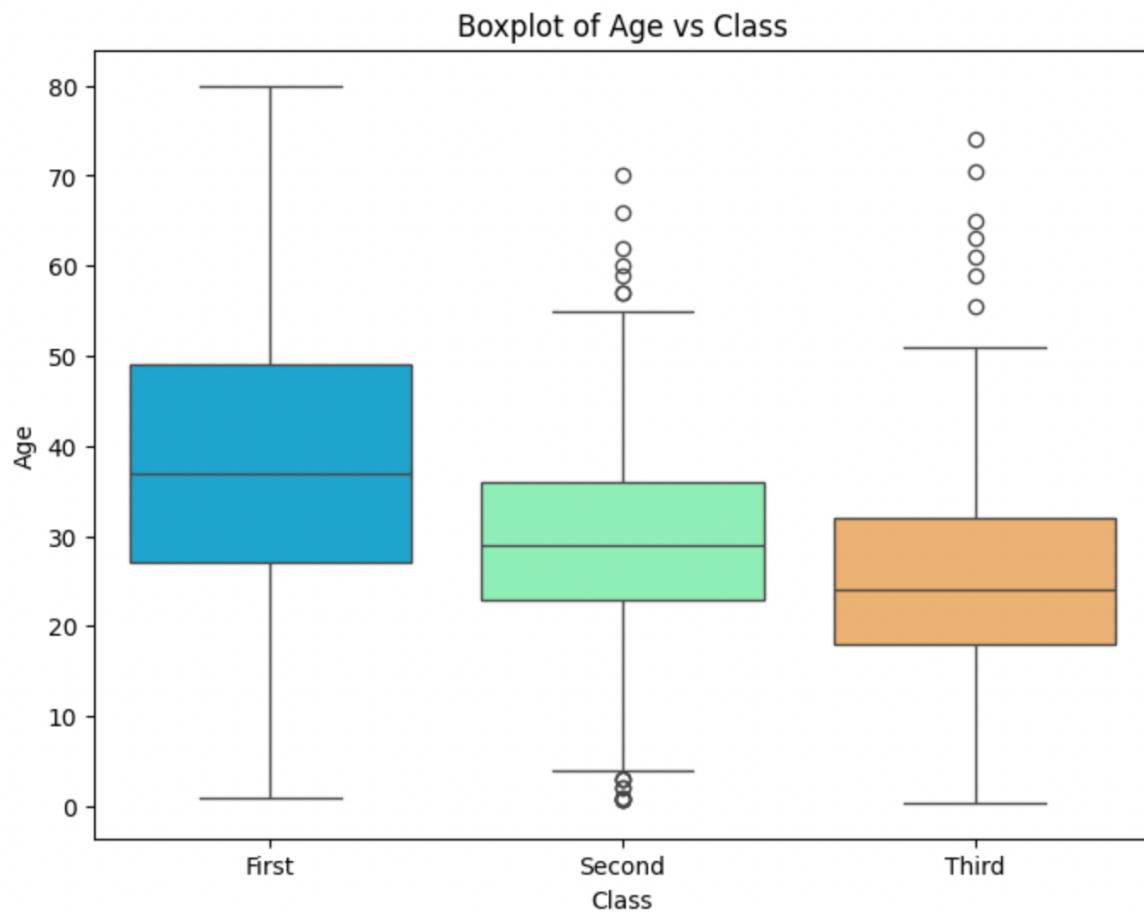
```
#Lab22_2
import ssl
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

ssl._create_default_https_context = ssl._create_unverified_context

titanic = sns.load_dataset('titanic')
```

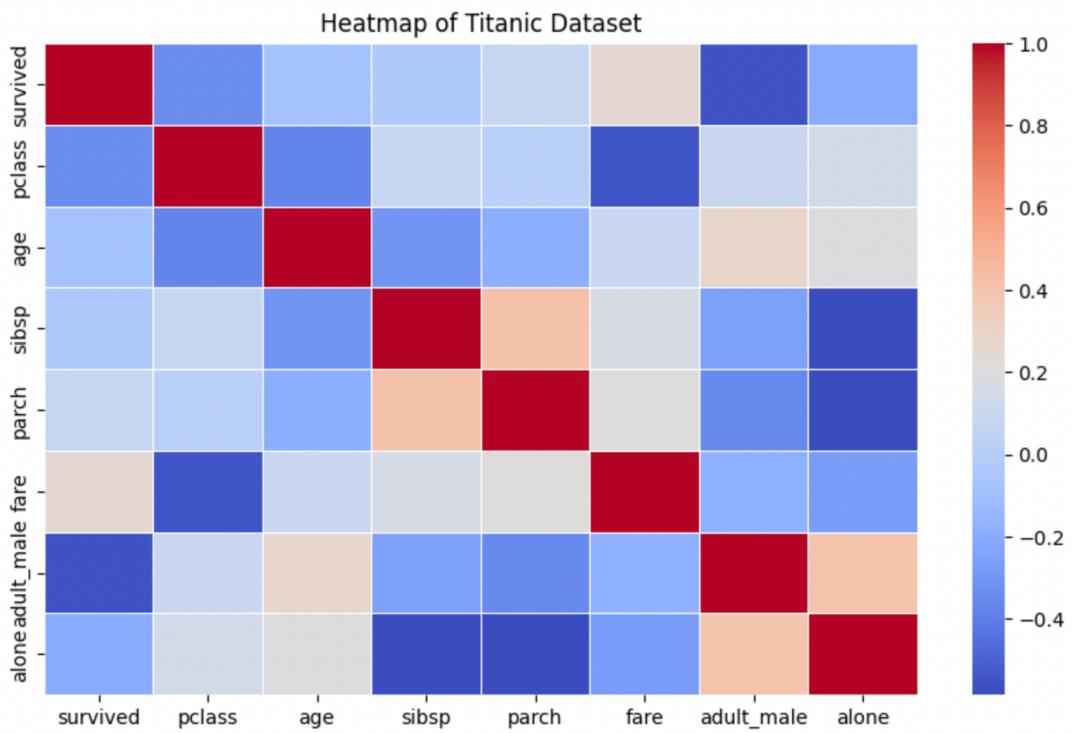
a. Create a boxplot of Age vs Class. Use the rainbow palette.

```
#Lab22_2a
# Create a boxplot of Age vs Class using the rainbow palette
plt.figure(figsize=(8, 6))
sns.boxplot(x='class', y='age', data=titanic, palette='rainbow')
plt.title('Boxplot of Age vs Class')
plt.xlabel('Class')
plt.ylabel('Age')
plt.show()
```



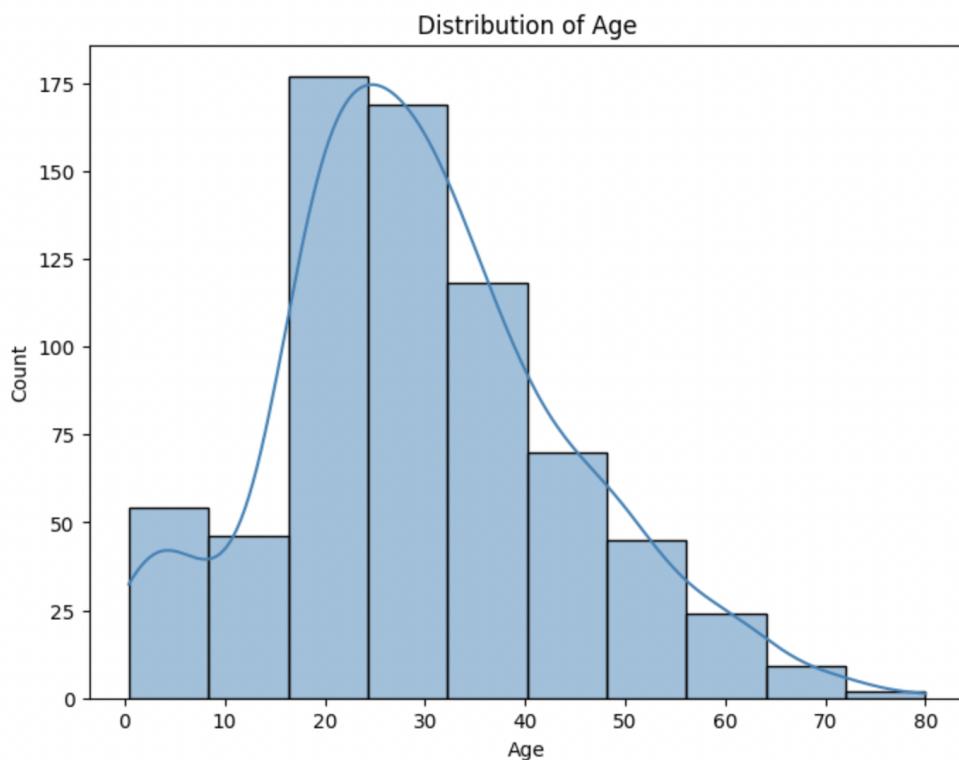
b. Create a heatmap of all the variables.

```
[40]: #Lab22_2b
# Create a heatmap of all the variables
plt.figure(figsize=(10, 6))
sns.heatmap(titanic.corr(numeric_only=True), annot=False, cmap='coolwarm', linewidths=0.5)
plt.title('Heatmap of Titanic Dataset')
plt.show()
```



c. Create a distplot of the count of age.

```
#Lab22_2c
# Create a histogram with KDE for Age distribution
plt.figure(figsize=(8, 6))
sns.histplot(titanic['age'].dropna(), kde=True, bins=10, color='steelblue', edgecolor='black')
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```

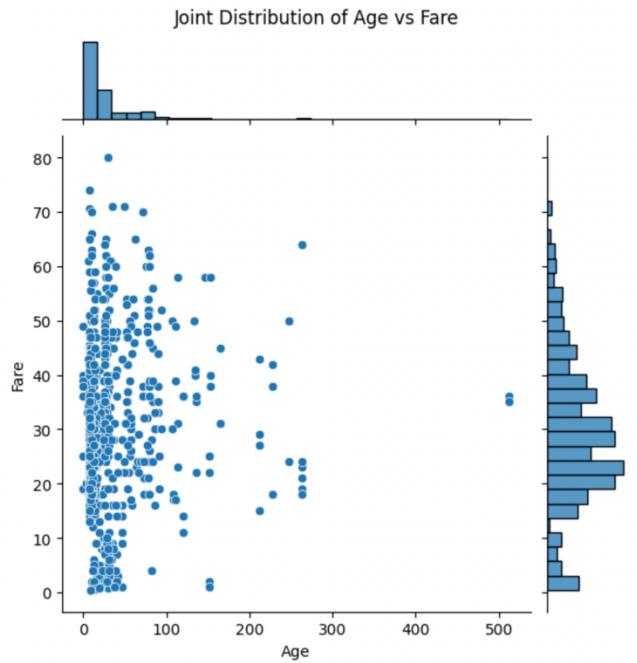


d. Jointplot comprises of three plots.

- One plot displays a bivariate graph which shows how the dependent variable(Y) varies with the independent variable(X).
- Another plot is placed horizontally at the top of the bivariate graph and it shows the distribution of the independent variable(X).
- The third plot is placed on the right margin of the bivariate graph with the orientation set to vertical and it shows the distribution of the dependent variable(Y). See here for more: <https://ajaytech.co/2020/09/28/8780/>

It is very simple to implement using the Seaborn library. Create a jointplot of Age vs Fare variables from the Titanic data.

```
[10]: #Lab22_2d
g = sns.jointplot(data=titanic, x="fare", y="age", kind="scatter", marginal_kws=dict(bins=30, fill=True))
g.fig.suptitle('Joint Distribution of Age vs Fare', y=1.02)
g.set_axis_labels('Age', 'Fare')
plt.show()
```



3. Gen AI Prompt Engineering:

Use Gen_AI_sales_dataset.csv. You will create visualizations that highlight patterns and trends in the dataset, with a focus on product performance, sales trends, and store behavior.

Bot Used: ChatGPT

Prompt:

The screenshot shows a dark-themed ChatGPT interface. At the top, there is a file icon labeled "Gen_AI_sales_dataset.csv" and "Spreadsheet". Below this, a message box contains a task description for "Sales Trends Over Time". The task involves creating a line plot for total sales over a year, identifying peak months, and showing both total sales and units sold on a dual-axis plot. A "Give me python code" button is present at the bottom of this message box. To the right of the message box are two small icons: a square with a checkmark and a circular arrow. Below the first message box, another message box contains a task for "Sales by Store". It asks for a bar chart of total sales per store, analysis of best and worst performing stores, and a stacked bar chart showing monthly sales changes for each store. The entire interface has rounded corners and a clean, modern look.

Sales Trends Over Time:

- Create a line plot showing the total sales over time for the entire year (monthly data).
- Insight: Can you identify peak sales months? What might have caused these peaks (e.g., promotions, holidays)?
- Additional Task: Show both total sales and units sold on the same plot with dual axes to highlight any patterns between the two metrics.

Give me python code

b. Sales by Store:

- Create a bar chart showing the total sales per store for the entire year.
- Insight: Which store performed the best? Which store had the lowest sales? Are there any outliers?
- Additional Task: Break down the sales performance by store over time (monthly). Use a stacked bar chart to show how sales change month-over-month for each store

c. Product Category Performance:

- Create a pie chart or bar plot showing the distribution of sales by product category across the year.
- Insight: Which product category contributed the most to total sales? What percentage of the overall sales does each category represent?
- Additional Task: Create a bar plot showing the average sales per product category and rank the categories based on the average sales.

d. Sales Correlation Analysis:

- Create a scatter plot showing the relationship between units sold and total sales.
- Insight: Is there a correlation between the number of units sold and total sales? Are there any anomalies?
- Additional Task: Add a regression line to the scatter plot to better understand the relationship

e. Store Performance Analysis:

- Create a boxplot for each store showing the distribution of sales over the months.
- Insight: Which stores have the most consistent sales? Which stores have the most variability in sales performance?
- Additional Task: Use a heatmap to show the monthly performance of each store

Gen AI:

I'll first inspect the dataset to understand its structure and determine the appropriate columns for analysis. Then, I'll generate the Python code to create the requested visualization.

The dataset contains sales transactions with the following relevant columns:

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 import numpy as np
5 import scipy.stats as stats
6
7 # Load dataset
8 df = pd.read_csv('/mnt/data/Gen_AI_sales_dataset.csv')
9
10 # Convert date column to datetime format
11 df['date'] = pd.to_datetime(df['date'])
12
13 # Aggregate data by month
14 df['month'] = df['date'].dt.to_period('M')
15 monthly_sales = df.groupby('month').agg({'total_sales': 'sum', 'units_sold': 'sum'}).reset_index()
16 monthly_sales['month'] = monthly_sales['month'].astype(str)
17
18 # Plot dual-axis line chart
19 fig, ax1 = plt.subplots(figsize=(12, 6))
20 ax2 = ax1.twinx()
21
22 sns.lineplot(x='month', y='total_sales', data=monthly_sales, ax=ax1, color='b', marker='o', label='Total Sales')
23 sns.lineplot(x='month', y='units_sold', data=monthly_sales, ax=ax2, color='r', marker='s', label='Units Sold')
24
25 # Labels and title
26 ax1.set_xlabel('Month')
27 ax1.set_ylabel('Total Sales ($)', color='b')
28 ax2.set_ylabel('Units Sold', color='r')
29 plt.title('Monthly Sales Trends')
30 plt.xticks(rotation=45)
31 ax1.legend(loc='upper left')
32 ax2.legend(loc='upper right')
33
34 plt.show()
35
36 # Sales by Store - Bar Chart
37 total_sales_by_store = df.groupby('store_id')['total_sales'].sum().reset_index()

```

You've hit the Free plan limit for GPT-4o. You need GPT-4o to continue this chat because there's an attachment. Your limit resets after 4:36 AM.

Ask anything

+ Search Reason ...

ChatGPT can make mistakes. Check important info.

I'll first inspect the dataset to understand its structure and determine the appropriate columns for analysis. Then, I'll generate the Python code to create the requested visualization.

The dataset contains sales transactions with the following relevant columns:

```

50 # Sales By Store - Bar Chart
51 total_sales_by_store = df.groupby('store_id')['total_sales'].sum().reset_index()
52 plt.figure(figsize=(12, 6))
53 sns.barplot(x='store_id', y='total_sales', data=total_sales_by_store, palette='viridis')
54 plt.xlabel('Store ID')
55 plt.ylabel('Total Sales ($)')
56 plt.title('Total Sales per Store')
57 plt.xticks(rotation=90)
58 plt.show()
59
60 # Sales Performance by Store Over Time (Stacked Bar Chart)
61 monthly_store_sales = df.groupby(['month', 'store_id'])['total_sales'].sum().unstack().fillna(0)
62 monthly_store_sales.plot(kind='bar', stacked=True, figsize=(14, 7), colormap='tab10')
63 plt.xlabel('Month')
64 plt.ylabel('Total Sales ($)')
65 plt.title('Monthly Sales by Store (Stacked)')
66 plt.xticks(rotation=45)
67 plt.legend(title='Store ID', bbox_to_anchor=(1.05, 1), loc='upper left')
68 plt.show()
69
70 # Product Category Performance - Pie Chart
71 category_sales = df.groupby('product_category')['total_sales'].sum().reset_index()
72 plt.figure(figsize=(10, 6))
73 plt.pie(category_sales['total_sales'], labels=category_sales['product_category'], autopct='%1.1f%%', colors=sns.color_palette('coolwarm'))
74 plt.title('Sales Distribution by Product Category')
75 plt.show()
76
77 # Product Category Performance - Bar Plot
78 plt.figure(figsize=(12, 6))
79 sns.barplot(x='product_category', y='total_sales', data=category_sales, palette='coolwarm')
80 plt.xlabel('Product Category')
81 plt.ylabel('Total Sales ($)')
82 plt.title('Total Sales by Product Category')
83 plt.xticks(rotation=45)
84 plt.show()
85
86 # Average Sales per Product Category
87 avg_sales_by_category = df.groupby('product_category')['total_sales'].mean().reset_index().sort_values(by='total_sales'

```

You've hit the Free plan limit for GPT-4o. You need GPT-4o to continue this chat because there's an attachment. Your limit resets after 4:36 AM.

Ask anything

+ Search Reason ...

ChatGPT can make mistakes. Check important info.

I'll first inspect the dataset to understand its structure and determine the appropriate columns for analysis. Then, I'll generate the Python code to create the requested visualization.

The dataset contains sales transactions with the following relevant columns:

```

1 I'll first inspect the dataset to understand its
2 structure and determine the appropriate
3 columns for analysis. Then, I'll generate the
4 Python code to create the requested
5 visualization.
6
7 The dataset contains sales transactions with
8 the following relevant columns:
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111

```

You've hit the free plan limit for GPT-4o. You need GPT-4o to continue this chat because there's an attachment. Your limit resets after 4:36AM.

New chat Get Plus

Ask anything + Search Reason ... ChatGPT can make mistakes. Check important info. ?

Insights:

a. Sales Trends Over Time:

- Peak Sales Months:** From the line plot, we can identify months with the highest total sales. If there are spikes, they may align with seasonal trends, holidays (e.g., December for holiday shopping), or promotional events.
- Possible Causes:** If sales peak during November–December, it could be due to Black Friday, Cyber Monday, or holiday shopping. If mid-year peaks exist, they might correlate with back-to-school sales or summer promotions.

b. Sales by Store:

- Best Performing Store:** The bar chart highlights the store with the highest total sales.
- Lowest Performing Store:** The store with the lowest bar has the least total sales.
- Outliers:** If one or two stores have significantly higher or lower sales than others, they may indicate exceptional performance or underperformance due to location, customer traffic, or inventory issues.

c. Product Category Performance:

- Top-Contributing Category:** The pie chart shows which product category dominates sales. The category with the largest slice contributes the most to revenue.
- Category Percentage:** Each slice's percentage represents its share of total sales, helping identify dependencies on specific products.

d. Sales Correlation Analysis:

Ask anything + Search Reason ... ChatGPT can make mistakes. Check important info. ?

The screenshot shows a web browser window with three tabs open:

- Week11_Due_4-1
- (14) Partha B. | LinkedIn
- Sales Trends Visualization

The main content area displays the following text from ChatGPT:

c. Product Category Performance:

- Top-Contributing Category: The pie chart shows which product category dominates sales. The category with the largest slice contributes the most to revenue.
- Category Percentage: Each slice's percentage represents its share of total sales, helping identify dependencies on specific products.

d. Sales Correlation Analysis:

- Correlation Between Units Sold & Total Sales: The scatter plot and correlation coefficient indicate the relationship. A strong positive correlation suggests that as units sold increase, total sales rise proportionally.
- Anomalies: If there are points far from the trend line, they could indicate high-ticket items with fewer sales or bulk purchases with lower revenue per unit.

e. Store Performance Analysis:

- Consistent Stores: Stores with a small interquartile range (IQR) in the boxplot have stable monthly sales.
- High Variability: Stores with large IQRs or frequent outliers fluctuate in performance, possibly due to seasonality, marketing efforts, or location-based factors.
- Heatmap Observations: If certain months are darker (higher sales) for specific stores, they may benefit from regional events or promotions. ↴

At the bottom of the ChatGPT interface, there is a text input field labeled "Ask anything" and several buttons: "+", "Search", "Reason", and "...". Below the input field, a note says "ChatGPT can make mistakes. Check important info." and a help icon is present.

Solution:

```
# LAB 22 QUESTION 3 GEN AI

#Lab22_3
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df = pd.read_csv("Gen_AI_sales_dataset.csv")
```

a. Sales Trends Over Time:

- Create a line plot showing the total sales over time for the entire year (monthly data).
- Insight: Can you identify peak sales months? What might have caused these peaks (e.g., promotions, holidays)?
- Additional Task: Show both total sales and units sold on the same plot with dual axes to highlight any patterns between the two metrics.

```
|: #Lab22_3a
# Convert date column to datetime format
df['date'] = pd.to_datetime(df['date'])

# Aggregate data by month
df['month'] = df['date'].dt.to_period('M')
monthly_sales = df.groupby('month').agg({'total_sales': 'sum', 'units_sold': 'sum'}).reset_index()
monthly_sales['month'] = monthly_sales['month'].astype(str)

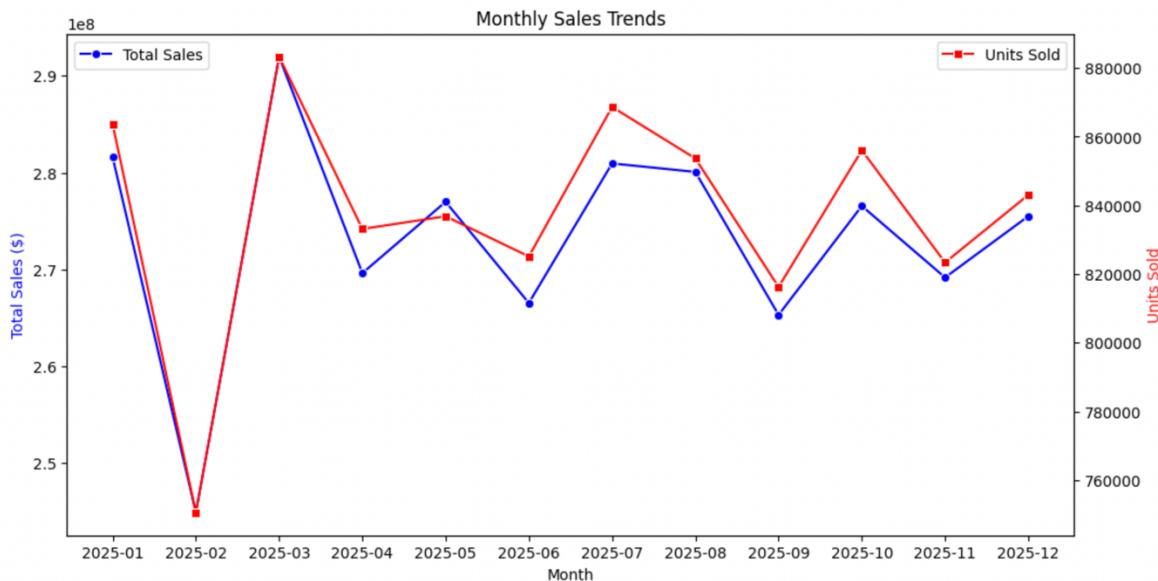
# Plot dual-axis line chart
fig, ax1 = plt.subplots(figsize=(12, 6))
ax2 = ax1.twinx()

sns.lineplot(x='month', y='total_sales', data=monthly_sales, ax=ax1, color='b', marker='o', label='Total Sales')
sns.lineplot(x='month', y='units_sold', data=monthly_sales, ax=ax2, color='r', marker='s', label='Units Sold')

# Labels and title
ax1.set_xlabel('Month')
ax1.set_ylabel('Total Sales ($)')
ax2.set_ylabel('Units Sold')
plt.title('Monthly Sales Trends')
plt.xticks(rotation=45)
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

plt.show()
```

```
plt.show()
```

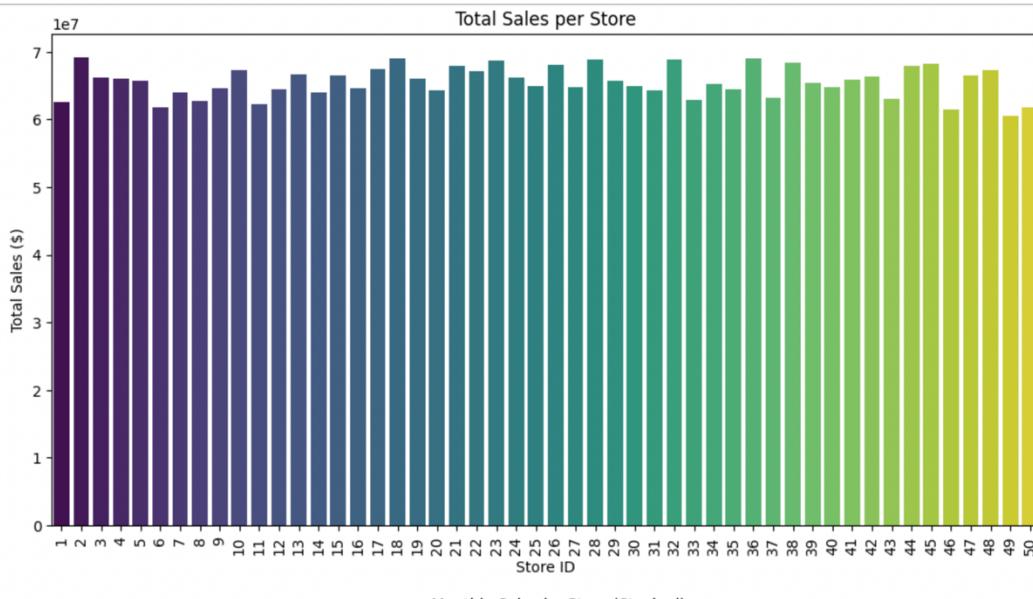


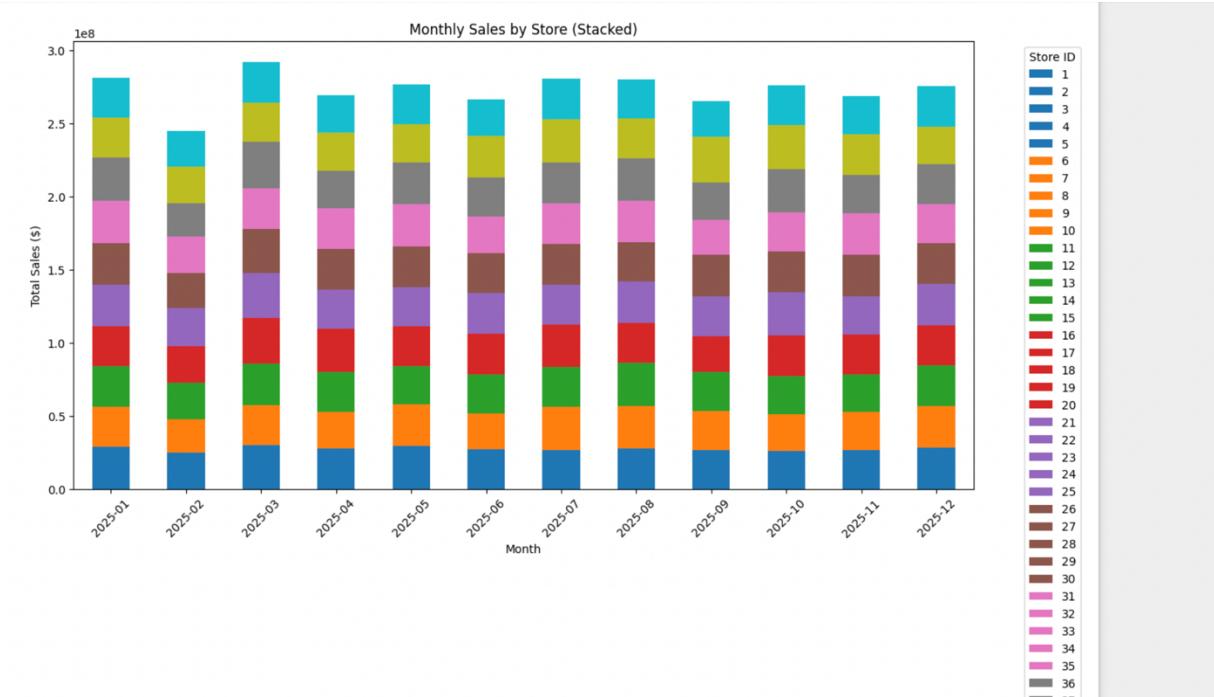
b. Sales by Store:

- Create a bar chart showing the total sales per store for the entire year.
- Insight: Which store performed the best? Which store had the lowest sales? Are there any outliers?
- Additional Task: Break down the sales performance by store over time (monthly). Use a stacked bar chart to show how sales change month-over-month for each store.

```
#Lab22_3b
# Sales by Store - Bar Chart
total_sales_by_store = df.groupby('store_id')['total_sales'].sum().reset_index()
plt.figure(figsize=(12, 6))
sns.barplot(x='store_id', y='total_sales', data=total_sales_by_store, palette='viridis')
plt.xlabel('Store ID')
plt.ylabel('Total Sales ($)')
plt.title('Total Sales per Store')
plt.xticks(rotation=90)
plt.show()

# Sales Performance by Store Over Time (Stacked Bar Chart)
monthly_store_sales = df.groupby(['month', 'store_id'])['total_sales'].sum().unstack().fillna(0)
monthly_store_sales.plot(kind='bar', stacked=True, figsize=(14, 7), colormap='tab10')
plt.xlabel('Month')
plt.ylabel('Total Sales ($)')
plt.title('Monthly Sales by Store (Stacked)')
plt.xticks(rotation=45)
plt.legend(title='Store ID', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```





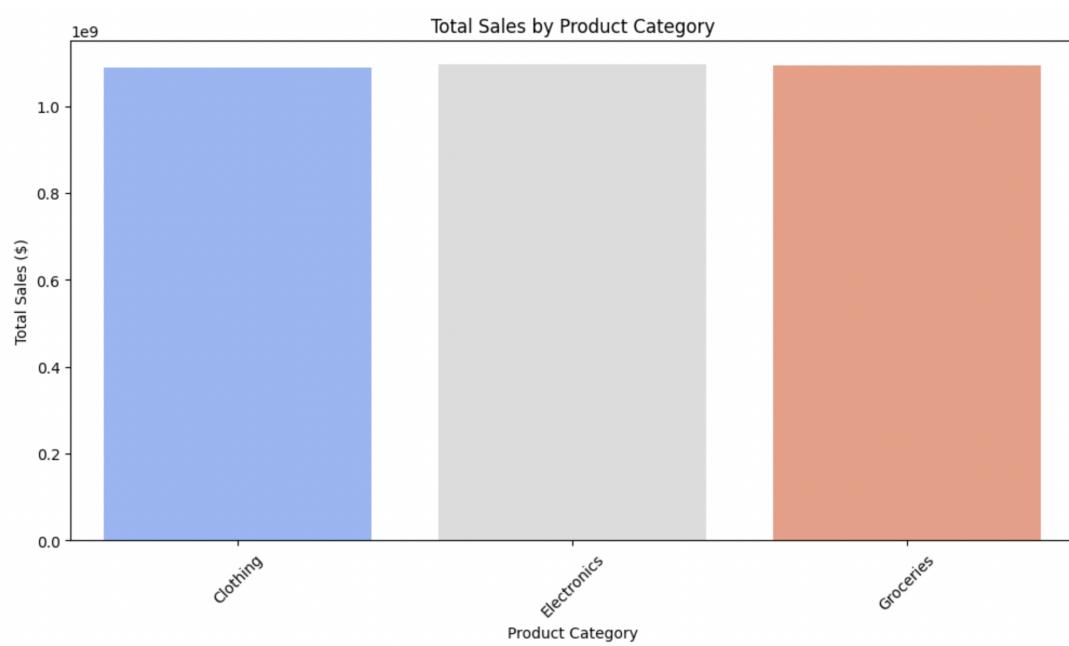
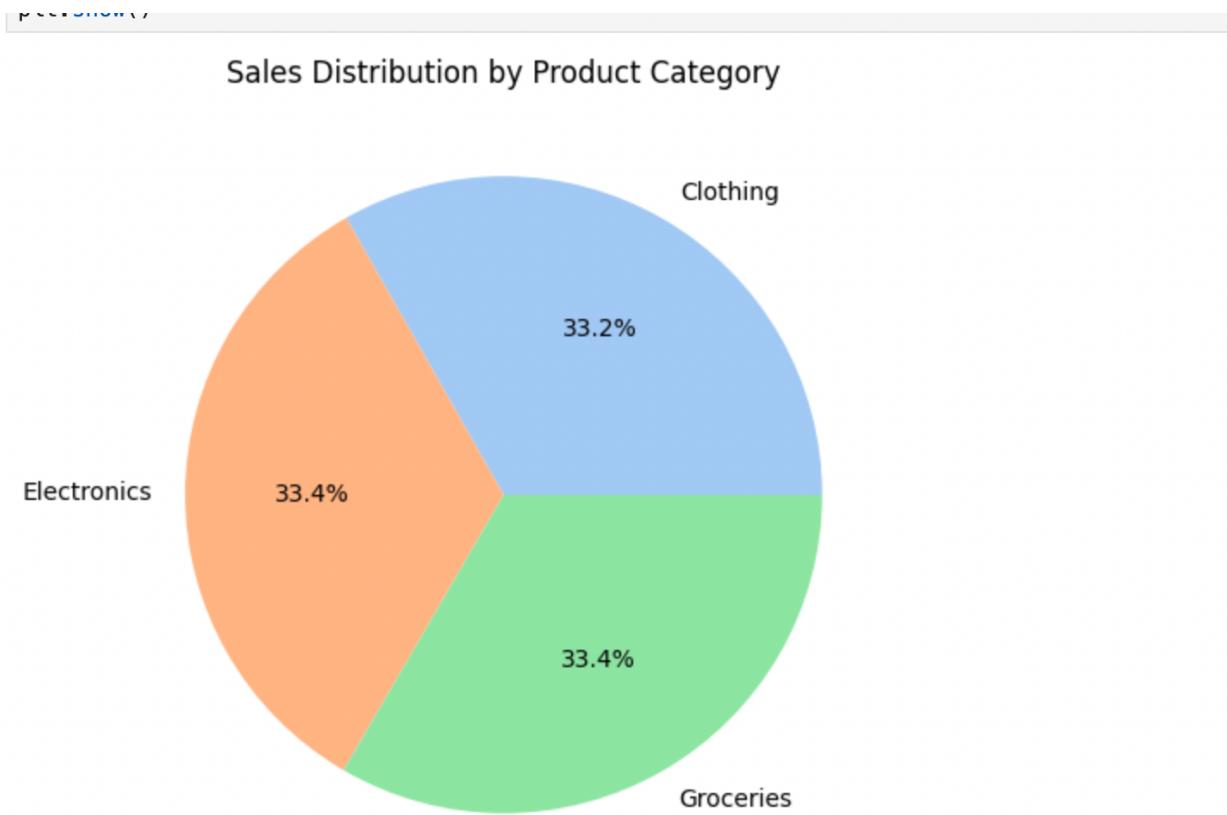
c. Product Category Performance:

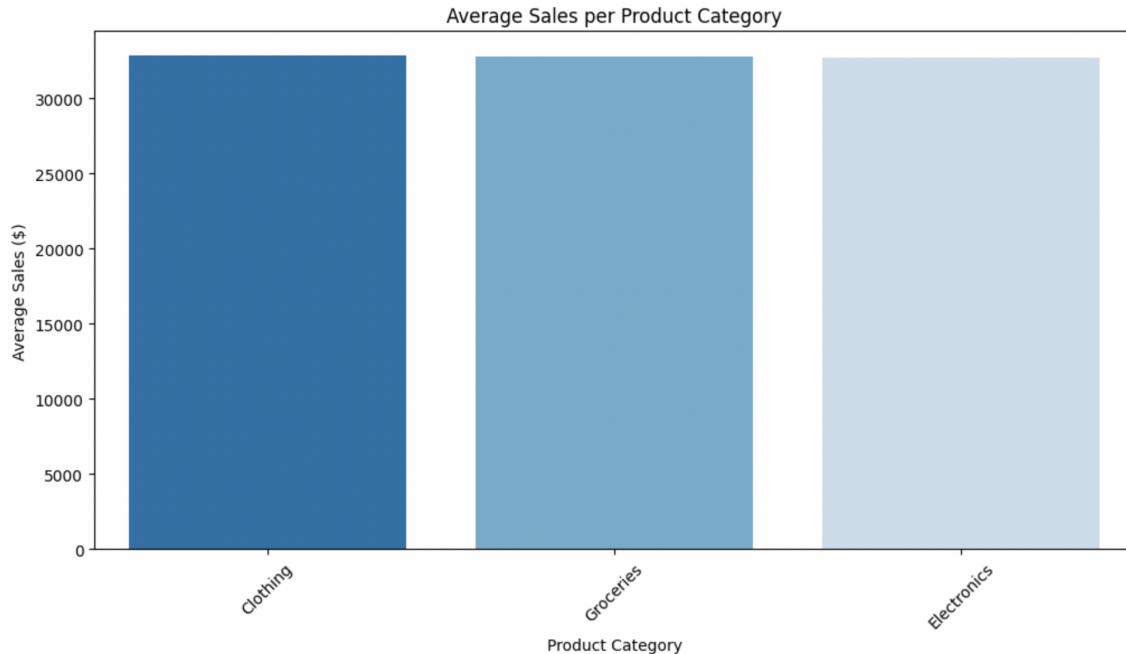
- Create a pie chart or bar plot showing the distribution of sales by product category across the year.
- Insight: Which product category contributed the most to total sales? What percentage of the overall sales does each category represent?
- Additional Task: Create a bar plot showing the average sales per product category and rank the categories based on the average sales.

```
#Lab22_3c
# Product Category Performance - Pie Chart
category_sales = df.groupby('product_category')['total_sales'].sum().reset_index()
plt.figure(figsize=(10, 6))
plt.pie(category_sales['total_sales'], labels=category_sales['product_category'], autopct='%1.1f%%', colors=sns.color_palette('pastel'))
plt.title('Sales Distribution by Product Category')
plt.show()

# Product Category Performance - Bar Plot
plt.figure(figsize=(12, 6))
sns.barplot(x='product_category', y='total_sales', data=category_sales, palette='coolwarm')
plt.xlabel('Product Category')
plt.ylabel('Total Sales ($)')
plt.title('Total Sales by Product Category')
plt.xticks(rotation=45)
plt.show()

# Average Sales per Product Category
avg_sales_by_category = df.groupby('product_category')['total_sales'].mean().reset_index().sort_values(by='total_sales', ascending=False)
plt.figure(figsize=(12, 6))
sns.barplot(x='product_category', y='total_sales', data=avg_sales_by_category, palette='Blues_r')
plt.xlabel('Product Category')
plt.ylabel('Average Sales ($)')
plt.title('Average Sales per Product Category')
plt.xticks(rotation=45)
plt.show()
```



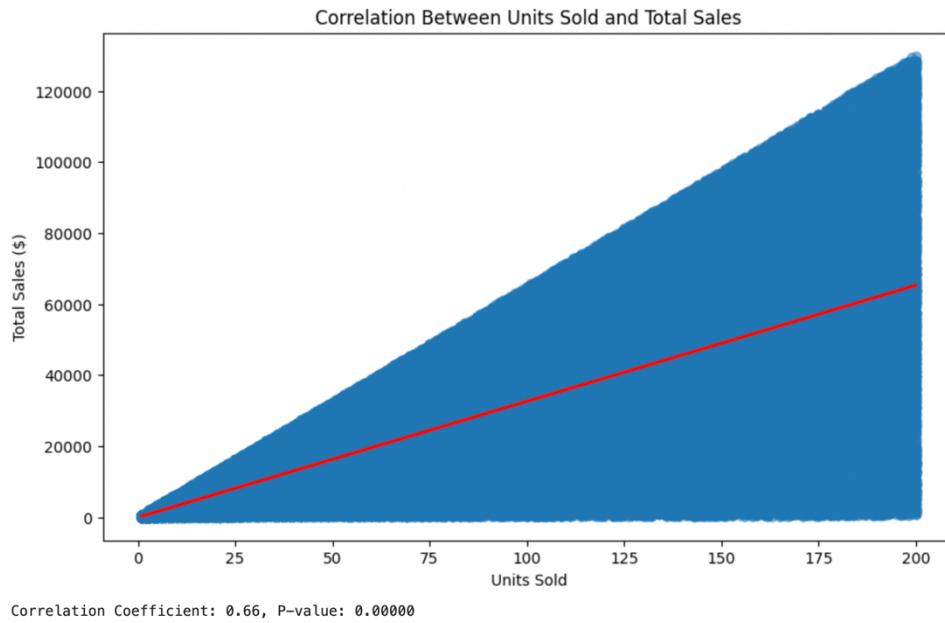


d. Sales Correlation Analysis:

- Create a scatter plot showing the relationship between units sold and total sales.
- Insight: Is there a correlation between the number of units sold and total sales? Are there any anomalies?
- Additional Task: Add a regression line to the scatter plot to better understand the relationship.

```
: import scipy.stats as stats
: #Lab22_3d
# Sales Correlation Analysis - Scatter Plot with Regression Line
plt.figure(figsize=(10, 6))
sns.replot(x='units_sold', y='total_sales', data=df, scatter_kws={'alpha':0.5}, line_kws={'color':'red'})
plt.xlabel('Units Sold')
plt.ylabel('Total Sales ($)')
plt.title('Correlation Between Units Sold and Total Sales')
plt.show()

# Calculate correlation coefficient
correlation, p_value = stats.pearsonr(df['units_sold'], df['total_sales'])
print(f'Correlation Coefficient: {correlation:.2f}, P-value: {p_value:.5f}')
```



e. Store Performance Analysis:

- Create a boxplot for each store showing the distribution of sales over the months.
- Insight: Which stores have the most consistent sales? Which stores have the most variability in sales performance?
- Additional Task: Use a heatmap to show the monthly performance of each store.

```
#Lab22_3e
import matplotlib.pyplot as plt
import seaborn as sns

# Convert date column to datetime format
df['date'] = pd.to_datetime(df['date'])

# Extract year and month for grouping
df['year_month'] = df['date'].dt.to_period('M')

# Aggregate total sales per store per month
store_sales = df.groupby(['store_id', 'year_month'])['total_sales'].sum().reset_index()

# Convert year_month to string format for better plotting
store_sales['year_month'] = store_sales['year_month'].astype(str)

# Plot boxplot for sales distribution by store
plt.figure(figsize=(14, 7))
sns.boxplot(x='store_id', y='total_sales', data=df, palette='husl') # Use a vibrant color palette
plt.xlabel('Store ID', fontsize=14)
plt.ylabel('Total Sales ($)', fontsize=14)
plt.title('Sales Distribution per Store', fontsize=16)
plt.xticks(rotation=45, fontsize=12)
plt.grid(True)
plt.show()

# Pivot the data to create a matrix of sales per store per month
sales_pivot = store_sales.pivot(index='store_id', columns='year_month', values='total_sales')

# Plot heatmap
plt.figure(figsize=(15, 8))
sns.heatmap(sales_pivot, cmap='coolwarm', linewidths=0.5, linecolor='gray', annot=False)
plt.xlabel('Month-Year')
plt.ylabel('Store ID')
plt.title('Monthly Sales Performance by Store')
plt.xticks(rotation=90)
plt.show()
```

