ayush-200pythonchallenges-15-days

July 15, 2024

[486]:

```
# BASIC EXERCISE FOR BEGINNERS
# Given two integer numbers, return their product only if the product is equal␣
↪to or lower than 1000. Otherwise, return their sum
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
product = a * b
sum = a + b

print('First Number: ',a)
print('Second Number: ',b)

if product<1000:
    print ('Product: ',product)
else:
    print ('Sum: ',sum)
```

```
First Number:  45
Second Number:  23
Sum:  68
```

[2]:

```
# Write a program to iterate the first 10 numbers, and in each iteration, print␣
↪the sum of the current and previous number
sum = 0
for i in range(1, 11):
    sum += i
    print("Current Number: ", i, "Previous Number: ",i-1, " Sum: ",sum)
```

```
Current Number:  1 Previous Number:  0  Sum:  1
Current Number:  2 Previous Number:  1  Sum:  3
Current Number:  3 Previous Number:  2  Sum:  6
Current Number:  4 Previous Number:  3  Sum:  10
Current Number:  5 Previous Number:  4  Sum:  15
Current Number:  6 Previous Number:  5  Sum:  21
Current Number:  7 Previous Number:  6  Sum:  28
Current Number:  8 Previous Number:  7  Sum:  36
Current Number:  9 Previous Number:  8  Sum:  45
Current Number:  10 Previous Number:  9  Sum:  55
```

[484]:

```python
# Write a program to accept a string from the user and display characters that are present at an even index number
word = input("Enter any word: ")
size = len(word)
print('Character present at even index number: ')
for i in range(0,size+1,2):
    print(word[i])
```

```
Character present at even index number:
K
i
h
a
```

[482]:

```python
# Write a program to remove characters from a string starting from zero up to n and return a new string
word = input("Enter any word: ")
l = len(word)
r = int(input("Enter no. of characters to be removed: "))
new_word = word[r:l]
print("Original String: ",word)
print('No. of characters removed: ',r)
print("String after removed characters: ",new_word)
```

```
Original String:  Krishna
No. of characters removed:  3
String after removed characters:  shna
```

[6]:

```python
# Check if the first and last number of a list is the same
list = [19, 20, 30, 40, 10]
if list[0] == list[-1]:
    print("SAME")
else:
    print("NOT SAME")
```

```
NOT SAME
```

[7]:

```python
# Display numbers divisible by 5 from a list
list = [10, 20, 75, 46, 55]
print("Given list:",list)
print('Divisible by 5:')
for i in list:
    if i % 5 == 0:
        print(i)
```

```
Given list: [10, 20, 75, 46, 55]
Divisible by 5:
10
```

```
20
75
55
```

[490]:

```python
# Return the count of a given substring from a string
str = "Ayush is a good developer. Ayush is a swimmer. I like book reading.\
Ayush went to gym. He likes travelling. Ayush ate Ice-Cream"
cnt = str.count("Ayush")
print('No. of times word is in string: ',cnt)
```

```
No. of times word is in string:  4
```

[3]:

```python
# Print the following pattern
n = 5
for i in range(1,n+1):
    for j in range(1,i+1):
        print (i,end=" ")
    print(" ")
```

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

[6]:

```python
# Check Palindrome Number. Write a program to check if the given number is a␣
 ↪palindrome number
# A palindrome number is a number that is the same after reverse. For example,␣
 ↪545, is the palindrome numbers
num = input("Enter a number: ")
if num == num[::-1]:
    print(f"{num} is a palindrome.")
else:
    print(f"{num} is not a palindrome.")
```

```
12321 is a palindrome.
```

[7]:

```python
#Create a new list from two list using the following condition
#Given two list of numbers, write a program to create a new list such that the␣
 ↪new list should contain odd numbers from the first list and even numbers␣
 ↪from the second list.
list1 = [1, 2, 3, 4, 5]
list2 = [10, 11, 12, 13, 14]
new_list = []

for num in list1:
    if num % 2 == 1:
        new_list.append(num)
```

```
for num in list2:
    if num % 2 == 0:
        new_list.append(num)

print("New list:", new_list)
```

```
New list: [1, 3, 5, 10, 12, 14]
```

[10]:

```
# Write a Program to extract each digit from an integer in the reverse order ;␣
↪Reverse a given integer number
num = input("Enter a number: ")
print('Original Number: ',num)
new_num = num[::-1]
print('Reversed Number: ',new_num)
```

```
Original Number:  9337753561
Reversed Number:  1653577339
```

[14]:

```
# Calculate income tax for the given income by adhering to the rules below
def calculate_income_tax(income):
    if income <= 10000:
        return 0
    elif income <= 20000:
        return (income - 10000) * 0.10
    else:
        return 10000 * 0.10 + (income - 20000) * 0.20

income = int(input("Enter income: "))
tax_payable = calculate_income_tax(income)
print(f"The income tax payable for an income of ${income} is ${tax_payable:.2f}.
↪")
# .2 in the print statement is used to format the floating-point number␣
↪tax_payable to two decimal places
```

```
The income tax payable for an income of $100000 is $17000.00.
```

[491]:

```
# Print multiplication table from 1 to 10
print('Multiplication Table from 1 to 10')
for i in range(1, 11):
    for j in range(1, 11):
        print(i*j, end=" ")
    print()
```

```
Multiplication Table from 1 to 10
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
```

```
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```

[21]:

```python
# Print a downward Half-Pyramid Pattern of Star (asterisk)
n = 5
for i in range(n):
    print("*"*(n-i))
```

```
*****
****
***
**
*
```

[25]:

```python
#Write a function called exponent(base, exp) that returns an int value of base␣
 ↪raises to the power of exp.
#Note here exp is a integer, and the base is an integer.
def exponent(base, exp):
    return base ** exp
base = int(input("Enter base number: "))
exp = int(input("Enter exp number: "))
print(f"{base} raised to the power of {exp} is {exponent(base, exp)}")
```

```
5 raised to the power of 2 is 25
```

[493]:

```python
# PYTHON INPUT AND OUTPUT EXERCISE
# Write a program to accept two numbers from the user and calculate␣
 ↪multiplication
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))

print('First Number: ',a)
print('Second Number: ',b)
print('Product: ',a*b)
```

```
First Number:  45
Second Number:  53
Product:  2385
```

[27]:

```python
# Display three string "Name", "Is", "James" as "Name**Is**James"
#Use the print() function to format the given words in the mentioned format.␣
 ↪Display the ** separator between each string.
print("My", "Name", "Is", "James", sep='**')
```

```
My**Name**Is**James
```

[31]:

```python
# Convert Decimal number to octal
decimal_number = int(input("Enter decimal number: "))
octal_number = format(decimal_number, 'o')
print(f"The octal representation of {decimal_number} is {octal_number}")

# format(decimal_number, 'o'):
# Converts decimal_number (a decimal integer) to its octal representation using the format specifier 'o'.
# 'o' indicates octal format in Python's format() function.
```

```
The octal representation of 8 is 10
```

[35]:

```python
# Display float number with 2 decimal places
num = float(input("Enter number with more than 2 decimal places: "))
print("Original number: ",num)
print(f"The float number is {num:.2f}")
```

```
Original number:  3.45689
The float number is 3.46
```

[36]:

```python
# Accept a list of 5 float numbers as an input from the user
list = []
print("Enter 5 float numbers:")

for i in range(5):
    num = float(input(f"Enter number {i+1}: "))
    list.append(num)
print("Desired List: ",list)
```

```
Enter 5 float numbers:
Desired List:  [3.23, 7.89, 2.5, 45.6, 98.4561]
```

[55]:

```python
# Write all content of a given file into a new file by skipping line number 5
content = """line1
line2
line3
line4
line5
line6
line7
"""
with open('test.txt', 'w') as file:
    file.write(content)
print("test.txt has been created with the specified content.")

input_file = "test.txt"
```

```
output_file = "output.txt"

with open(input_file, 'r') as f_in:
    lines = f_in.readlines()
    modified_lines = lines[:4] + lines[5:]
with open(output_file, 'w') as f_out:
    f_out.writelines(modified_lines)

print(f"Content of {input_file} excluding line 5 has been written to␣
 ↪{output_file}.")
print("Modified content of output_file:")
print("".join(modified_lines), end="")

# In Python, triple quotes (""" or ''') are used for multi-line strings.
# Triple quotes allow you to create strings that span multiple lines without␣
 ↪using newline characters (\n).
# This is particularly useful for readability and maintaining the formatting of␣
 ↪long text blocks.
```

```
test.txt has been created with the specified content.
Content of test.txt excluding line 5 has been written to output.txt.
Modified content of output_file:
line1
line2
line3
line4
line6
line7
```

[50]:

```
# Accept any three string from one input() call
input_string = input("Enter three strings separated by spaces: ")
strings = input_string.split()

if len(strings) != 3:
    print("Error: Please enter exactly three strings separated by spaces.")
else:
    print("Entered strings:", strings)
```

```
Entered strings: ['Fortis', 'Fortuna', 'Adiuvat']
```

[54]:

```
# Write a program to use string.format() method to format the following three␣
 ↪variables as per the expected output
totalMoney = float(input("Enter totalmoney: "))
quantity = int(input("Enter no. of footballs: "))
price = float(input("Enter price: "))
```

```
print(f"I have {totalMoney:.2f} dollars so I can buy {quantity} football for␣
 ↪{price:.2f} dollars.")
```

```
I have 1000.00 dollars so I can buy 3 football for 300.00 dollars.
```

[56]:

```
# Write a program to check if the given file is empty or not
import os
file_name = 'test.txt'

if os.path.getsize(file_name) == 0:
    print(f"{file_name} is empty.")
else:
    print(f"{file_name} is not empty.")
```

```
test.txt is not empty.
```

[494]:

```
# PYTHON LOOP EXERCISE
# Print First 10 natural numbers using while loop
print('First 10 Natural Numbers')
i = 1
while i <= 10:
    print(i)
    i += 1
```

```
First 10 Natural Numbers
1
2
3
4
5
6
7
8
9
10
```

[61]:

```
# Write a program to print the following number pattern using a loop.
n = 5
for i in range(1,n+1):
    for j in range(1,i+1):
        print(j,end=" ")
    print(" ")
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

[62]:

```
# Write a program to accept a number from a user and calculate the sum of all␣
 ↪numbers from 1 to a given number
n = int(input("Enter number till be added: "))
sum = 0
for i in range(1,n+1):
    sum += i
print(f"Sum of numbers from 1 to {n} is ",sum)
```

```
Sum of numbers from 1 to 15 is  120
```

[64]:

```
# Write a program to print multiplication table of a given number
n = int(input("Enter any number: "))
print(f"Multiplication table of {n}")
for i in range(1,11):
    product = n*i
    print(f"{n} x {i} = ",product)
```

```
Multiplication table of 5
5 x 1 =  5
5 x 2 =  10
5 x 3 =  15
5 x 4 =  20
5 x 5 =  25
5 x 6 =  30
5 x 7 =  35
5 x 8 =  40
5 x 9 =  45
5 x 10 =  50
```

[66]:

```
# Write a program to display only those numbers from a list that satisfy the␣
 ↪following conditions
#   The number must be divisible by five
#   If the number is greater than 150, then skip it and move to the next number
#   If the number is greater than 500, then stop the loop
list = [12, 75, 150, 180, 145, 525, 50]
for i in list:
    if i > 500:
        break
    if i > 150:
        continue
    if i % 5 == 0:
        print(i)
```

```
75
150
145
```

[70]:

```
# Write a program to count the total number of digits in a number using a while␣
  ↪loop.
number = int(input("Enter a number: "))
num = number
digit_count = 0

if number == 0:
    digit_count = 1
else:
    while number != 0:
        number //= 10
        digit_count += 1

print(f"The total number of digits of {num} is {digit_count}")
```

```
The total number of digits of 9337753561 is 10
```

[71]:

```
# Write a program to use for loop to print the following reverse number pattern
n = 5
for i in range(n, 0, -1):
    for j in range(i, 0, -1):
        print(j, end=' ')
    print()
```

```
5 4 3 2 1
4 3 2 1
3 2 1
2 1
1
```

[495]:

```
# Print list in reverse order using a loop
list = [10, 20, 30, 40, 50]
new_list = reversed(list)
# iterate reversed list
print('Reversed list:')
for item in new_list:
    print(item)
```

```
Reversed list:
50
40
30
20
10
```

[73]:

```
# Use else block to display a message "Done" after successful execution of for␣
  ↪loop
```

```
for i in range(1,5):
    print(i)
else:
    print("Done!")
```

```
1
2
3
4
Done!
```

[76]:

```
# Write a program to display all prime numbers within a range
def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True

start = int(input("Enter the start of the range: "))
end = int(input("Enter the end of the range: "))

print(f"Prime numbers between {start} and {end} are:")

for number in range(start, end + 1):
    if is_prime(number):
        print(number, end=' ')
```

```
Prime numbers between 10 and 50 are:
11 13 17 19 23 29 31 37 41 43 47
```

[78]:

```
# Display Fibonacci series up to n terms
n = int(input("Enter any number: "))
fibonacci_series = [0,1]

for i in range(2,n):
    next_term = fibonacci_series[i-1] + fibonacci_series[i-2]
    fibonacci_series.append(next_term)

print(f"Fibonacci series up to {n} terms:")
for term in fibonacci_series:
    print(term, end=' ')
```

```
Fibonacci series up to 15 terms:
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

[79]:

```python
# Write a program to use the loop to find the factorial of a given number
num = int(input("Enter a number: "))
factorial = 1

for i in range(1, num + 1):
    factorial *= i

print(f"The factorial of {num} is {factorial}")
```

```
The factorial of 5 is 120
```

[2]:

```python
# Write a program to rint the cube of all numbers from 1 to a given number
num = int(input("Enter a number: "))
print(f"Cubes of numbers from 1 to {num}:")
for i in range(1, num + 1):
    cube = i ** 3
    print(f"The cube of {i} is {cube}")
```

```
Cubes of numbers from 1 to 5:
The cube of 1 is 1
The cube of 2 is 8
The cube of 3 is 27
The cube of 4 is 64
The cube of 5 is 125
```

[4]:

```python
# Write a program to calculate the sum of series up to n term
n = int(input("Enter the number of terms: "))
series_sum = 0
term = 2
terms = []

for i in range(n):
    series_sum += term
    terms.append(term)
    term = term * 10 + 2

print("Series terms:", ' + '.join(map(str, terms)))
print(f"The sum of the series up to {n} terms is {series_sum}")
```

```
Series terms: 2 + 22 + 222 + 2222 + 22222
The sum of the series up to 5 terms is 24690
```

[5]:

```python
# Write a program to print the following star pattern
rows = 5
for i in range(1, rows + 1):
    print('* ' * i)
for i in range(rows - 1, 0, -1):
    print('* ' * i)
```

```
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

[6]:

```
# PYTHON FUNCTIONS EXERCISE
# Write a program to create a function that takes two arguments, name and age,␣
 ↪and print their value
def func(name, age):
    print("Name:", name)
    print("Age:", age)

name = input("Enter your name: ")
age = input("Enter your age: ")

print("Output")
func(name, age)
```

```
Output
Name: Ayush
Age: 20
```

[7]:

```
# Write a program to create function func1() to accept a variable length of␣
 ↪arguments and print their value
def func1(*args):
    for arg in args:
        print(arg, end=' ')
    print()

func1(1, 2, 3)
func1("apple", "banana", "cherry")
func1(1.5, True, "Hello", 42)
```

```
1 2 3
apple banana cherry
1.5 True Hello 42
```

[34]:

```
# Write a program to create function calculation() such that it can accept two␣
 ↪variables and
# Calculate addition and subtraction. Also, it must return both addition and␣
 ↪subtraction in a single return call
def calculation(a, b):
    addition = a + b
    subtraction = a - b
    return addition, subtraction

res = calculation(40, 10)
print(res)
```

```
(50, 30)
```

[33]:

```
# PYTHON STRING EXERCISE
# Write a program to create a new string made of an input string’s first,␣
 ↪middle, and last character
def string(s):
    first_char = s[0]
    middle_char = s[len(s) // 2]
    last_char = s[-1]

    new_string = first_char + middle_char + last_char
    return new_string

input_string = input("Enter a string: ")

if len(input_string) >= 3:
    result = string(input_string)
    print("Original string: ",input_string)
    print("New string:", result)
else:
    print("The input string is too short to have distinct first, middle, and␣
 ↪last characters.")
```

```
Original string:  ayush
New string: auh
```

[32]:

```
# Write a program to create a new string made of the middle three characters of␣
 ↪an input string
def middle_three_chars(s):
    if len(s) < 3 or len(s) % 2 == 0:
        return "The input string must be at least 3 characters long and have an␣
 ↪odd length."

    mid_index = len(s) // 2
    middle_three = s[mid_index-1:mid_index+2]

    return middle_three

input_string = input("Enter a string: ")
```

```
result = middle_three_chars(input_string)
print("Original string: ",input_string)
print("New string: ",result)
```

```
Original string:  ayush
New string:  yus
```

[28]:

```
# Given two strings, s1 and s2. Write a program to create a new string s3 by␣
 ↪appending s2 in the middle of s1
def append_in_middle(s1, s2):
    mid_index = len(s1) // 2

    s3 = s1[:mid_index] + s2 + s1[mid_index:]
    return s3

s1 = input("Enter the first string (s1): ")
s2 = input("Enter the second string (s2): ")

s3 = append_in_middle(s1, s2)
print("First string: ",s1)
print("Second string: ",s2)
print("The new string (s3) is:", s3)
```

```
First string:  ayush
Second string:  mayurakshi
The new string (s3) is: aymayurakshiush
```

[27]:

```
# Given two strings, s1 and s2, write a program to return a new string made of␣
 ↪s1 and s2's first, middle, and last characters
def get_first_middle_last(s):
    first_char = s[0]
    middle_char = s[len(s) // 2]
    last_char = s[-1]
    return first_char, middle_char, last_char

def create_new_string(s1, s2):
    s1_first_char, s1_middle_char, s1_last_char = get_first_middle_last(s1)
    s2_first_char, s2_middle_char, s2_last_char = get_first_middle_last(s2)
    new_string = s1_first_char + s2_first_char + s1_middle_char +␣
 ↪s2_middle_char + s1_last_char + s2_last_char
    return new_string

s1 = input("Enter the first string (s1): ")
s2 = input("Enter the second string (s2): ")

if len(s1) >= 3 and len(s2) >= 3:
    result = create_new_string(s1, s2)
    print("First string: ",s1)
    print("Second string: ",s2)
    print("The new string is:", result)
else:
    print("Both input strings must have at least 3 characters.")
```

```
First string:  qwert
Second string:  asdfg
The new string is: qaedtg
```

[26]:

```
# Given string contains a combination of the lower and upper case letters
# Write a program to arrange the characters of a string so that all lowercase␣
 ↪letters should come first
def arrange_string(s):
    lower_case = [char for char in s if char.islower()]
    upper_case = [char for char in s if char.isupper()]
    arranged_string = ''.join(lower_case + upper_case)
    return arranged_string

input_string = input("Enter a string: ")

result = arrange_string(input_string)
print("Original string: ",input_string)
print("The rearranged string is:", result)

# The first char before for specifies that each element in the resulting list␣
 ↪should be the character char that satisfies the condition if char.islower()
```

```
Original string:  AaYUsghk
The rearranged string is: asghkAYU
```

[36]:

```
# Count all letters, digits, and special symbols from a given string
def count_characters(s):
    letters = digits = special_symbols = 0

    for char in s:
        if char.isalpha():
            letters += 1
        elif char.isdigit():
            digits += 1
        elif not char.isspace():
            special_symbols += 1

    return letters, digits, special_symbols

input_string = input("Enter a string: ")
letters, digits, special_symbols = count_characters(input_string)
```

```
print(f"Original String: {input_string}")
print(f"Letters: {letters}")
print(f"Digits: {digits}")
print(f"Special symbols: {special_symbols}")

# char.isalpha() checks for alphabetic characters
# char.isdigit() checks for digits characters
# char.isalnum() checks for alphanumeric characters
# char.isspace() checks for whitespace characters
```

```
Original String:  Hello123!@#
Letters: 5
Digits: 3
Special symbols: 3
```

[4]:

```
# Write a program to count occurrences of all characters within a string
def count_characters(s):
    counts = {}
    for char in s:
        counts[char] = counts.get(char, 0) + 1
    return counts

input_string = input("Enter a string: ")
result = count_characters(input_string)
print(f"Original String: {input_string}")

for char, count in result.items():
    print(f"{char}: {count}")
```

```
Original String: australia
a: 3
u: 1
s: 1
t: 1
r: 1
l: 1
i: 1
```

[1]:

```
# Remove empty strings from a list of strings
def remove_empty_strings(string_list):
    return [string for string in string_list if string]

string_list = ["hello", "", "world", "", "python", "", ""]
filtered_list = remove_empty_strings(string_list)

print("Original list:", string_list)
print("Filtered list:", filtered_list)
```

```
Original list: ['hello', '', 'world', '', 'python', '', '']
Filtered list: ['hello', 'world', 'python']
```

[5]:

```
# Remove special symbols or punctuation from a string
def remove_punctuation(string):
    cleaned_list = []
    for char in string:
        if char.isalnum() or char.isspace():
            cleaned_list.append(char)

    cleaned_string = ''.join(cleaned_list)
    return cleaned_string

input_string = input("Enter a string: ")
cleaned_string = remove_punctuation(input_string)

print("String with punctuation: ", input_string)
print("String without punctuation: ", cleaned_string)
```

```
String with punctuation:  /*Jon is @developer & musician
String without punctuation:  Jon is developer  musician
```

[7]:

```
# Removal all characters from a string except integers
def remove_non_integers(string):
    cleaned_list = [char for char in string if char.isdigit()]
    cleaned_string = ''.join(cleaned_list)
    return cleaned_string

input_string = input("Enter a string: ")
cleaned_string = remove_non_integers(input_string)

print("Original string: ", input_string)
print("String with only integers:", cleaned_string)
```

```
Original string:  Hello123, world! 456
String with only integers: 123456
```

[11]:

```
# Write a program to find words with both alphabets and numbers from an input␣
 ↪string
def func(input_string):
    words = input_string.split()

    result = []
    for word in words:
        has_alpha = any(char.isalpha() for char in word)
        has_digit = any(char.isdigit() for char in word)
        if has_alpha and has_digit:
            result.append(word)
    return result

input_string = input("Enter a string: ")
words = func(input_string)

print("Original string: ", input_string)
print("Words with both alphabets and numbers:", " ".join(words))
```

```
Original string:  abc123 def 456ghi 789
Words with both alphabets and numbers: abc123 456ghi
```

[12]:

```
#  Replace each special symbol with # in the following string
def replace_special_symbols(string):
    result = ""
    for char in string:
        if char.isalnum() or char.isspace():
            result += char
        else:
            result += "#"
    return result

input_string = input("Enter a string: ")
modified_string = replace_special_symbols(input_string)

print("Original string: ", input_string)
print("Modified string:", modified_string)
```

```
Original string:  /*Jon is @developer & musician!!
Modified string: ##Jon is #developer # musician##
```

[2]:

```
# PYTHON DATA STRUCTURE EXERCISE (List, Set, Dictionary, and Tuple Operations)
# Create a list by picking an odd-index items from the first list and even␣
 ↪index items from the second
def create_list(list1, list2):
    odd_index_items = []
    even_index_items = []

    for i in range(1, len(list1), 2):
        odd_index_items.append(list1[i])
    for i in range(0, len(list2), 2):
        even_index_items.append(list2[i])

    return odd_index_items + even_index_items
```

```
list1 = input("Enter elements of first list: ").split()
list2 = input("Enter elements of second list: ").split()
new_list = create_list(list1, list2)

print("First list: ",list1)
print("Second list: ",list2)
print("New list: ",new_list)
```

```
First list:  ['2', '4', '6', '8', '10', '12', '1', '4']
Second list:  ['3', '6', '9', '12', '15', '18', '21']
New list:  ['4', '8', '12', '4', '3', '9', '15', '21']
```

[6]:

```
# Slice list into 3 equal chunks and reverse each chunk
def slice_and_reverse(list):
    chunk_size = len(list) // 3
    chunk1 = list[:chunk_size]
    chunk2 = list[chunk_size:chunk_size*2]
    chunk3 = list[chunk_size*2:]

    reversed_chunk1 = chunk1[::-1]
    reversed_chunk2 = chunk2[::-1]
    reversed_chunk3 = chunk3[::-1]

    print("Chunk 1:", chunk1)
    print("After reversing it:", reversed_chunk1)
    print("Chunk 2:", chunk2)
    print("After reversing it:", reversed_chunk2)
    print("Chunk 3:", chunk3)
    print("After reversing it:", reversed_chunk3)
    return ()

input_list = input("Enter the elements of the list, separated by spaces: ").
 ↪split()

if len(input_list) >= 3 and len(input_list) % 3 == 0:
    result_list = slice_and_reverse(input_list)
else:
    print("The list must have at least 3 elements and its length should be␣
 ↪divisible by 3.")
```

```
Chunk 1: ['1', '2', '3']
After reversing it: ['3', '2', '1']
Chunk 2: ['4', '5', '6']
After reversing it: ['6', '5', '4']
Chunk 3: ['7', '8', '9']
After reversing it: ['9', '8', '7']
```

[7]:

```
# Write a program to iterate a given list and count the occurrence of each␣
 ↪element and create a dictionary to show the count of each element.
def count_occurrences(list):
    occurrence_dict = {}
    for element in list:
        if element in occurrence_dict:
            occurrence_dict[element] += 1
        else:
            occurrence_dict[element] = 1
    return occurrence_dict

input_list = input("Enter the elements of the list separated by spaces: ").
 ↪split()
occurrences = count_occurrences(input_list)

print("Input List: ",input_list)
print("Element occurrence count: ",occurrences)
```

```
Input List:  ['1', '2', '2', '3', '3', '3', '4', '4', '4', '4', '5', '5', '5',
'5', '5']
Element occurrence count:  {'1': 1, '2': 2, '3': 3, '4': 4, '5': 5}
```

[9]:

```
# Create a Python set such that it shows the element from both lists in a pair
def create_pairs_set(list1, list2):
    pairs = set(zip(list1, list2))
    return pairs

list1_input = input("Enter elements of first list separated by spaces: ").
 ↪split()
list2_input = input("Enter elements of second list separated by spaces: ").
 ↪split()
pairs_set = create_pairs_set(list1_input, list2_input)

print("List 1: ",list1_input)
print("List 2: ",list2_input)
print("Output list with pairs:", pairs_set)

# 'zip' function to pair elements from two lists provided by the user
# zip(list1, list2) would produce an iterator of tuples like [(1, 'a'), (2,␣
 ↪'b'), (3, 'c')]
# Using set(zip(list1, list2)) ensures that the resulting collection of pairs␣
 ↪is unique and unordered, which can be useful in scenarios where duplicates␣
 ↪are not desired
```

```
List 1:  ['1', '2', '3', '4', '5']
List 2:  ['a', 'b', 'c', 'd', 'e']
Output list with pairs: {('1', 'a'), ('2', 'b'), ('5', 'e'), ('3', 'c'), ('4',
'd')}
```

[11]:

```
# Checks if one set is a subset or superset of another set. If found, delete␣
 ↪all elements from that set
def check_and_clear_sets(set1, set2):
    if set1.issubset(set2):
        print(f"{set1} is a subset of {set2}")
        set1.clear()
    elif set1.issuperset(set2):
        print(f"{set1} is a superset of {set2}")
        set2.clear()
    else:
        print("No subset or superset relationship found.")

set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7, 8, 9}
check_and_clear_sets(set1, set2)

print("Set1: ",set1)
print("Set2: ",set2)
```

```
No subset or superset relationship found.
Set1:  {1, 2, 3, 4, 5}
Set2:  {3, 4, 5, 6, 7, 8, 9}
```

[16]:

```
# Get all values from the dictionary and add them to a list but don't add␣
 ↪duplicates
user_input = input("Enter dictionary items (key:value pairs separated by␣
 ↪commas): ")
items = [item.strip() for item in user_input.split(",")]

user_dict = {}
for item in items:
    parts = item.split(":")
    if len(parts) == 2:
        key, value = parts
        user_dict[key] = value
unique_values = list(set(user_dict.values()))

print("Input_Dict= ",user_input)
print("List of unique values:", unique_values)
```

```
Input_Dict=  a:1,b:2,c:1,d:3
List of unique values: ['1', '2', '3']
```

[20]:

```
# Remove duplicates from a list and create a tuple and find the minimum and␣
 ↪maximum number
```

```
user_input = input("Enter numbers separated by commas: ")
numbers_list = [int(num.strip()) for num in user_input.split(",")]

unique_numbers = list(set(numbers_list))
numbers_tuple = tuple(unique_numbers)

min_number = min(numbers_tuple)
max_number = max(numbers_tuple)

print("Input number:", user_input)
print("Tuple of unique numbers:", numbers_tuple)
print("Minimum number:", min_number)
print("Maximum number:", max_number)
```

```
Input number: 12,23,45,56,45,23,12,89,76,45
Tuple of unique numbers: (12, 45, 76, 23, 56, 89)
Minimum number: 12
Maximum number: 89
```

[21]:

```
# Concatenate two lists in the following order
# Define the lists
list1 = ["Hello ", "take "]
list2 = ["Dear", "Sir"]

concatenated_list = [i + j for i in list1 for j in list2]
print(concatenated_list)
```

```
['Hello Dear', 'Hello Sir', 'take Dear', 'take Sir']
```

[22]:

```
# Given a two Python list. Write a program to iterate both lists simultaneously␣
  ↪and
# Display items from list1 in original order and items from list2 in reverse␣
  ↪order
list1 = [10, 20, 30, 40]
list2 = [100, 200, 300, 400]

for x, y in zip(list1, list2[::-1]):
    print(x, y)
```

```
10 400
20 300
30 200
40 100
```

[23]:

```
# Write a program to add item 7000 after 6000 in the following Python List
list1 = [10, 20, [300, 400, [5000, 6000], 500], 30, 40]
```

```
list1[2][2].append(7000)
print(list1)

# understand indexing
# list1[0] = 10
# list1[1] = 20
# list1[2] = [300, 400, [5000, 6000], 500]
# list1[2][2] = [5000, 6000]
```

```
[10, 20, [300, 400, [5000, 6000, 7000], 500], 30, 40]
```

[26]:
```
#  You have given a nested list. Write a program to extend it by adding the␣
 ↪sublist ["h", "i", "j"] like as of alphabetical order
list1 = ["a", "b", ["c", ["d", "e", ["f", "g"], "k"], "l"], "m", "n"]
sub_list = ["h", "i", "j"]

list1[2][1][2].extend(sub_list)
print(list1)

# .append(element): Adds element as a single item to the end of the list. If␣
 ↪element is a list, it will be added as a nested list.
# .extend(iterable): Adds each element of iterable to the end of the list. If␣
 ↪iterable is a list, its elements are added individually, not as a nested list
```

```
['a', 'b', ['c', ['d', 'e', ['f', 'g', 'h', 'i', 'j'], 'k'], 'l'], 'm', 'n']
```

[29]:
```
# Write a program to find value 20 in the list, and if it is present, replace␣
 ↪it with 200
list1 = [5, 10, 20, 30, 20, 40]
for i in range(len(list1)):
    if list1[i] == 20:
        list1[i] = 200
print(list1)

# .replace() method is available for strings but not for lists
# text = "I have 20 apples and 20 oranges."
# new_text = text.replace("20", "200")
# print(new_text)
# Output: I have 200 apples and 200 oranges.
```

```
[5, 10, 200, 30, 200, 40]
```

[30]:
```
# Given a Python list, write a program to remove all occurrences of item 20.
list1 = [5, 20, 15, 20, 25, 50, 20]
list1 = [item for item in list1 if item != 20]
print(list1)
```

```
[5, 15, 25, 50]
```

[496]:

```
# Write a Python program to convert them into a dictionary in a way that item␣
 ↪from list1 is the key and item from list2 is the value
list1 = ['a', 'b', 'c', 'd']
list2 = [1, 2, 3, 4]

dictionary = dict(zip(list1, list2))
print(dictionary)

# 'dict' function can take an iterable of key-value pairs (like the tuples␣
 ↪produced by zip) and convert it into a dictionary
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

[31]:

```
# Merge two Python dictionaries into one
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'd': 4}

dict1.update(dict2)
print(dict1)

# .update() function updates the dictionary with elements from another␣
 ↪dictionary object or from an iterable of key-value pairs
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

[32]:

```
# Print the value of key ‘history’ from the below dict
# Define the dictionary
sampleDict = {
    "class": {
        "student": {
            "name": "Mike",
            "marks": {
                "physics": 70,
                "history": 80
            }
        }
    }
}

history_mark = sampleDict["class"]["student"]["marks"]["history"]
print(history_mark)

# understand how to locate NESTED KEY
# sampleDict['class'] = {'student': {'name': 'Mike', 'marks': {'physics': 70,␣
 ↪'history': 80}}}
# sampleDict['class']['student'] = {'name': 'Mike', 'marks': {'physics': 70,␣
 ↪'history': 80}}
```

```
# sampleDict['class']['student']['marks'] = {'physics': 70, 'history': 80}
```

```
80
```

[34]:

```
# Initialize dictionary with default values
employees = ['Kelly', 'Emma']
defaults = {"designation": 'Developer', "salary": 8000}

employee_dict = dict.fromkeys(employees, defaults)
print(employee_dict)

# .fromkeys() method is used to create a new dictionary from a given sequence␣
  ↪of keys, with all values set to a specified value
# Here .fromkeys() sets the same value for all keys and we need each employee␣
  ↪to have their own dictionary of default values
```

```
{'Kelly': {'designation': 'Developer', 'salary': 8000}, 'Emma': {'designation':
'Developer', 'salary': 8000}}
```

[33]:

```
# Write a Python program to create a new dictionary by extracting the mentioned␣
  ↪keys from the below dictionary
sample_dict = {
    "name": "Kelly",
    "age": 25,
    "salary": 8000,
    "city": "New York"
}

keys = ["name", "salary"]
new_dict = {key: sample_dict[key] for key in keys if key in sample_dict}

print(new_dict)
```

```
{'name': 'Kelly', 'salary': 8000}
```

[36]:

```
# Delete a list of keys from a dictionary
my_dict = {"name": "Kelly","age": 25,"salary": 8000,"city": "New york"}
keys_to_remove = ["name", "salary"]

for key in keys_to_remove:
    my_dict.pop(key, None) # None is provided to avoid KeyError if the key is␣
  ↪not found
print(my_dict)
```

```
{'age': 25, 'city': 'New york'}
```

[37]:

```
# Check if a value exists in a dictionary
def value_exists_in_dict(dict, value):
    return value in dict.values()

my_dict = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
value_to_check = 3

if value_exists_in_dict(my_dict, value_to_check):
    print(f"The value {value_to_check} exists in the dictionary.")
else:
    print(f"The value {value_to_check} does not exist in the dictionary.")

# To check if a value exists in a dictionary in Python, you can use 'in'␣
  ↪operator along with the .values() method of the dictionary
```

```
The value 3 exists in the dictionary.
```

[39]:

```
# Rename key of a dictionary
sample_dict = {
    "name": "Kelly",
    "age": 25,
    "salary": 8000,
    "city": "New york"
}

sample_dict['location'] = sample_dict.pop('city')
print(sample_dict)
```

```
{'name': 'Kelly', 'age': 25, 'salary': 8000, 'location': 'New york'}
```

[38]:

```
# Get the key of a minimum value from the following dictionary
sample_dict = {
    'a': 10,
    'b': 2,
    'c': 5,
    'd': 8
}
min_key = min(sample_dict, key=sample_dict.get)
print(f"The key with the minimum value is: {min_key}")
```

```
The key with the minimum value is: b
```

[40]:

```
# Write a Python program to change Brad's salary to 8500 in the following␣
  ↪dictionary
sample_dict = {
    'emp1': {'name': 'Jhon', 'salary': 7500},
    'emp2': {'name': 'Emma', 'salary': 8000},
    'emp3': {'name': 'Brad', 'salary': 6500}
}

sample_dict['emp3']['salary'] = 8500
print(sample_dict)
```

```
{'emp1': {'name': 'Jhon', 'salary': 7500}, 'emp2': {'name': 'Emma', 'salary':
8000}, 'emp3': {'name': 'Brad', 'salary': 8500}}
```

[41]:

```
# Swap two tuples in Python
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)

temp = tuple1
tuple1 = tuple2
tuple2 = temp

print("After swapping:")
print("tuple1:", tuple1)
print("tuple2:", tuple2)
```

```
After swapping:
tuple1: (4, 5, 6)
tuple2: (1, 2, 3)
```

[45]:

```
# Check if all items in the tuple are the same
def are_all_items_same(sample_tuple):
    if not sample_tuple:
        print("The tuple is empty.")
        return False
    first_item = sample_tuple[0]

    for item in sample_tuple:
        if item != first_item:
            print(f"Item {item} is not equal to the first item {first_item}.")
            return False

    print("All items in the tuple are the same.")
    return True

sample_tuple_1 = (1, 1, 1, 1)
print("Sample Tuple 1:", sample_tuple_1)
print("Result:", are_all_items_same(sample_tuple_1))
print()

sample_tuple_2 = (1, 2, 1, 1)
print("Sample Tuple 2:", sample_tuple_2)
print("Result:", are_all_items_same(sample_tuple_2))
print()

sample_tuple_3 = ()
print("Sample Tuple 3:", sample_tuple_3)
print("Result:", are_all_items_same(sample_tuple_3))
print()
```

```
Sample Tuple 1: (1, 1, 1, 1)
All items in the tuple are the same.
Result: True

Sample Tuple 2: (1, 2, 1, 1)
Item 2 is not equal to the first item 1.
Result: False

Sample Tuple 3: ()
The tuple is empty.
Result: False
```

[4]:

```
# PYTHON DATE AND TIME EXERCISE
# Print current date and time
import datetime
now = datetime.datetime.now()
print(now)

# Get Current Date
current_date = datetime.date.today()
print(current_date)

# Among all the attributes of datetime module, the most commonly used classes␣
 ↪in the datetime module are:
# datetime.datetime - represents a single point in time, including a date and a␣
 ↪time
# datetime.date - represents a date (year, month, and day) without a time
# datetime.time - represents a time (hour, minute, second, and microsecond)␣
 ↪without a date
# datetime.timedelta - represents a duration, which can be used to perform␣
 ↪arithmetic with datetime objects
```

```
2024-07-09 13:56:53.937557
2024-07-09
```

[5]:

```
# Print today's year, month and day
from datetime import date
today = date.today()
```

```
print("Current year:", today.year)
print("Current month:", today.month)
print("Current day:", today.day)
```

```
Current year: 2024
Current month: 7
Current day: 9
```

[6]:

```
# Convert string into a datetime object
import datetime
date_string = "Feb 25 2020 4:20PM"
format_string = "%b %d %Y %I:%M%p"

datetime_obj = datetime.datetime.strptime(date_string, format_string)
print(datetime_obj.strftime("%Y-%m-%d %H:%M:%S"))



# .strptime(string,format code) class method takes two arguments:
# string (that be converted to datetime)
# format code

# .strftime() method takes one or more format codes as an argument and returns␣
 ↪a formatted string based on it
# The format string should match the date-time string exactly :
# %a         Abbreviated weekday name.         Sun, Mon, ...
# %A         Full weekday name.         Sunday, Monday, ...
# %w         Weekday as a decimal number.         0, 1, ..., 6
# %d         Day of the month as a zero-padded decimal.         01, 02, ..., 31
# %-d         Day of the month as a decimal number.         1, 2, ..., 30
# %b         Abbreviated month name.         Jan, Feb, ..., Dec
# %B         Full month name.         January, February, ...
# %m         Month as a zero-padded decimal number.         01, 02, ..., 12
# %-m         Month as a decimal number.         1, 2, ..., 12
# %y         Year without century as a zero-padded decimal number.         00,␣
 ↪01, ..., 99
# %-y         Year without century as a decimal number.         0, 1, ..., 99
# %Y         Year with century as a decimal number.         2013, 2019 etc.
# %H         Hour (24-hour clock) as a zero-padded decimal number.         00,␣
 ↪01, ..., 23
# %-H         Hour (24-hour clock) as a decimal number.         0, 1, ..., 23
# %I         Hour (12-hour clock) as a zero-padded decimal number.         01,␣
 ↪02, ..., 12
# %-I         Hour (12-hour clock) as a decimal number.         1, 2, ... 12
# %p         Locale's AM or PM.         AM, PM
# %M         Minute as a zero-padded decimal number.         00, 01, ..., 59
# %-M         Minute as a decimal number.         0, 1, ..., 59
# %S        Second as a zero-padded decimal number.        00, 01, ..., 59
# %-S        Second as a decimal number.        0, 1, ..., 59
# %f        Microsecond as a decimal number, zero-padded on the left.
 ↪        000000 - 999999
# %z        UTC offset in the form +HHMM or -HHMM.
# %Z        Time zone name.
# %j        Day of the year as a zero-padded decimal number.        001, 002, ..
 ↪., 366
# %-j        Day of the year as a decimal number.        1, 2, ..., 366
# %U        Week number of the year (Sunday as the first day of the week). All␣
 ↪days in a new year preceding the first Sunday are considered to be in week 0.
 ↪        00, 01, ..., 53
# %W        Week number of the year (Monday as the first day of the week). All␣
 ↪days in a new year preceding the first Monday are considered to be in week 0.
 ↪        00, 01, ..., 53
# %c        Locale’s appropriate date and time representation.        Mon Sep␣
 ↪30 07:06:05 2013
# %x        Locale’s appropriate date representation.        09/30/13
# %X        Locale’s appropriate time representation.        07:06:05
# %%        A literal '%' character.        %
```

```
2020-02-25 16:20:00
```

[497]:

```
# Subtract a week (7 days)  from a given date in Python
import datetime
given_date = datetime.datetime(2020, 2, 25, 16, 20)
one_week = datetime.timedelta(days=7)
new_date = given_date - one_week
print(new_date)

# .timedelta() allows you to easily add or subtract specific durations to/from␣
 ↪datetime objects,
# Calculate the difference between dates, and create custom time durations
```

```
2020-02-18 16:20:00
```

[498]:

```
# Print a date in a the following format
# Day_name  Day_number  Month_name  Year
import datetime
given_date = datetime.datetime(2020, 2, 25)
formatted_date = given_date.strftime("%A %d %B %Y")
print(formatted_date)
```

```
Tuesday 25 February 2020
```

[499]:

```
# Find the day of the week of a given date
import datetime
```

```python
given_date = datetime.datetime(2020, 2, 25)
day_name = given_date.strftime("%A")
print(day_name)

# 'strftime' method can also be used to directly get the name of the day of the week
```

```
Tuesday
```

[7]:

```python
# Add a week (7 days) and 12 hours to a given date
import datetime
given_date = datetime.datetime(2020, 2, 25, 16, 20)
time_delta = datetime.timedelta(days=7, hours=12)
new_date = given_date + time_delta
print(new_date)
```

```
2020-03-04 04:20:00
```

[13]:

```python
# Calculate number of days between two given dates
from datetime import datetime

date1_str = input("Enter the first date (YYYY-MM-DD): ")
date2_str = input("Enter the second date (YYYY-MM-DD): ")

date1 = datetime.strptime(date1_str, "%Y-%m-%d")
date2 = datetime.strptime(date2_str, "%Y-%m-%d")
print("Date 1: ",date1)
print("Date 2: ",date2)

if date1 < date2:
    difference = date2 - date1
    days_difference = difference.days
    print(f"The number of days between {date1_str} and {date2_str} is {days_difference} days.")
elif date1 > date2:
    difference = date1 - date2
    days_difference = difference.days
    print(f"The number of days between {date2_str} and {date1_str} is {days_difference} days.")
else:
    print("Both dates are the same.")


# Using 'import datetime' gives you access to other classes and functions in the datetime module (like date, time, timedelta, timezone) without additional imports
# Using 'from datetime import datetime', you would need separate import␣
 ↪statements for each additional class or function you want to use like (from␣
 ↪datetime import datetime, timedelta)
```

```
Date 1:  2020-06-14 00:00:00
Date 2:  2019-06-28 00:00:00
The number of days between 2019-6-28 and 2020-6-14 is 352 days.
```

[18]:

```
# Write a program that asks the user for the current hour and for how many␣
 ↪hours in the future they want to go.
import datetime

current_hour = int(input("Enter the current hour (0-23): "))
future_hours = int(input("Enter the number of hours into the future: "))
now = datetime.datetime.now()

current_time = now.replace(hour=current_hour, minute=0, second=0, microsecond=0)
future_time = current_time + datetime.timedelta(hours=future_hours)

print(f"Time after {future_hours} hours from now will be {future_time.
 ↪strftime('%Y-%m-%d %H:%M:%S')}")
```

```
Time after 7 hours from now will be 2024-07-09 22:00:00
```

[19]:

```
# Write a Python program to determine whether a given year is a leap year
def leap_year(y):
    if y % 400 == 0:
        return True
    if y % 100 == 0:
        return False
    if y % 4 == 0:
        return True
    else:
        return False

print(leap_year(1900))
print(leap_year(2004))
```

```
False
True
```

[20]:

```
#  Write a Python program to print a string five times, with a delay of three␣
 ↪seconds
import time
def print_with_delay(string, repeat, delay):
    count = 0
    while count < repeat:
        print(string)
        time.sleep(delay)
        count += 1
print_with_delay("Hello, world!", 5, 3)

# .sleep() function in Python is used to pause the execution of the current␣
 ↪thread for a specified number of seconds. This function is part of the␣
 ↪'time' module
```

```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

[24]:

```
# Write a Python program to get the week number
import datetime
def get_week_number(year, month, day):
    date = datetime.date(year, month, day)
    return date.isocalendar()[1]

year = int(input("Enter the year: "))
month = int(input("Enter the month: "))
day = int(input("Enter the day: "))

week_number = get_week_number(year, month, day)
print(f"The week number for {year}-{month:02d}-{day:02d} is: {week_number}")

# 'month:02d' and 'day:02d' ensure that the month and day are displayed as two␣
 ↪digits (e.g., "06" for June)
# datetime.date: This is a class from the datetime module in Python, which␣
 ↪represents a date (year, month, day) in the Gregorian calendar
# date.isocalendar(): This method returns a tuple representing the ISO calendar␣
 ↪date for the date object
# Returned tuple has three components: (ISO year, ISO week number, ISO weekday).
# ISO year: The year according to the ISO calendar (often the same as the␣
 ↪Gregorian year, but can differ at the start and end of the year).
# ISO week number: The week number of the year, ranging from 1 to 53.
# ISO weekday: The day of the week, where Monday is 1 and Sunday is 7
# date.isocalendar()[1]: This accesses the second element of the tuple (index␣
 ↪1), which is the ISO week number
```

```
The week number for 2024-07-10 is: 28
```

[26]:

```
# PYTHON OOP EXERCISE
# Write a Python program to create a Vehicle class with max_speed and mileage␣
 ↪instance attributes
```

```
class Vehicle:
    def __init__(self, max_speed, mileage):
        self.max_speed = max_speed
        self.mileage = mileage

model1 = Vehicle(240, 18)
print(f"Your vehicle max. speed is {model1.max_speed} and mileage is {model1.
 ↪mileage}")
```

```
Your vehicle max. speed is 240 and mileage is 18
```

[500]:

```
# Create a Vehicle class without any variables and methods
class Vehicle:
    pass
```

[28]:

```
# Create a child class Bus that will inherit all of the variables and methods␣
 ↪of the Vehicle class
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def display_info(self):
        print(f"Vehicle Info: {self.year} {self.make} {self.model}")

class Bus(Vehicle):
    def __init__(self, make, model, year, capacity):
        super().__init__(make, model, year)
        self.capacity = capacity

    def display_info(self):
        super().display_info()
        print(f"Bus Capacity: {self.capacity} passengers")

bus = Bus("Mercedes", "Sprinter", 2020, 20)
bus.display_info()
```

```
Vehicle Info: 2020 Mercedes Sprinter
Bus Capacity: 20 passengers
```

[32]:

```
# Create a Bus class that inherits from the Vehicle class. Give the capacity␣
 ↪argument of Bus.seating_capacity() a default value of 50
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
```

```python
        self.year = year

    def display_info(self):
        print(f"Vehicle Info: {self.year} {self.make} {self.model}")

class Bus(Vehicle):
    def __init__(self, make, model, year, capacity=50):
        super().__init__(make, model, year)
        self.capacity = capacity

    def seating_capacity(self):
        return f"The seating capacity of the bus is {self.capacity} passengers"

    def display_info(self):
        super().display_info()
        print(f"Bus Capacity: {self.capacity} passengers")

bus = Bus("Mercedes", "Sprinter", 2020)
bus.display_info()
print(bus.seating_capacity())
```

```
Vehicle Info: 2020 Mercedes Sprinter
Bus Capacity: 50 passengers
The seating capacity of the bus is 50 passengers
```

[37]:

```python
# Create a Bus class that inherits from the Vehicle class. Give the capacity␣
 ↪argument of Bus.seating_capacity() a default value of 50
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def display_info(self):
        print(f"Vehicle Info: {self.year} {self.make} {self.model}")

class Bus(Vehicle):
    def __init__(self, make, model, year, capacity=50):
        super().__init__(make, model, year)
        self.capacity = capacity

    def seating_capacity(self):
        return f"The seating capacity of the bus is {self.capacity} passengers"

    def display_info(self):
        super().display_info()
        print(f"Bus Capacity: {self.capacity} passengers")
```

```
bus = Bus("Mercedes", "Sprinter", 2020)
bus.display_info()
print(bus.seating_capacity())
```

```
Vehicle Info: 2020 Mercedes Sprinter
Bus Capacity: 50 passengers
The seating capacity of the bus is 50 passengers
```

[38]:

```
# Define a property that must have the same value for every class instance␣
 ↪(object)
class Car:
    wheels = 4

    def __init__(self, color, brand):
        self.color = color
        self.brand = brand

car1 = Car("Red", "Toyota")
car2 = Car("Blue", "Honda")

print(car1.wheels)
print(car2.wheels)

print(car1.color)
print(car1.brand)
print(car2.color)
print(car2.brand)

Car.wheels = 6

print(car1.wheels)
print(car2.wheels)
```

```
4
4
Red
Toyota
Blue
Honda
6
6
```

[40]:

```
# Create a Bus child class that inherits from the Vehicle class. The default␣
 ↪fare charge of any vehicle is seating capacity * 100.
# If Vehicle is Bus instance, we need to add an extra 10% on full fare as a␣
 ↪maintenance charge.
```

```
# So total fare for bus instance will become the final amount = total fare +␣
 ↪10% of the total fare.
# Note: The bus seating capacity is 50. so the final fare amount should be 5500.
 ↪ You need to override the fare() method of a Vehicle class in Bus class
class Vehicle:
    def __init__(self, seating_capacity):
        self.seating_capacity = seating_capacity

    def fare(self):
        return self.seating_capacity * 100

class Bus(Vehicle):
    def __init__(self, seating_capacity=50):
        super().__init__(seating_capacity)

    def fare(self):
        total_fare = super().fare()
        maintenance_charge = total_fare * 0.10
        final_amount = total_fare + maintenance_charge
        return final_amount

bus = Bus()
print("Total Bus fare is:", bus.fare())
```

```
Total Bus fare is: 5500.0
```

[501]:

```
# Write a program to determine which class a given Bus object belongs to.
class Vehicle:
    def __init__(self, name, mileage, capacity):
        self.name = name
        self.mileage = mileage
        self.capacity = capacity

class Bus(Vehicle):
    pass

School_bus = Bus("School Volvo", 12, 50)
print(f"The class of the object School_bus is: {type(School_bus).__name__}")

if isinstance(School_bus, Bus): # checks if School_bus is an instance of the␣
 ↪Bus class.
    print("School_bus is an instance of the Bus class")
if isinstance(School_bus, Vehicle): # checks if School_bus is an instance of␣
 ↪the Vehicle class. Since Bus is a subclass of Vehicle, this will also return␣
 ↪'True'
    print("School_bus is also an instance of the Vehicle class")
```

```
# Using type():
# type(School_bus).__name__ returns the name of the class to which the object School_bus belongs. In this case, it will return "Bus".
# __name__ attribute is used to get the name of the class as a string
# type(School_bus) gives us the class type <class '__main__.Bus'>.
# type(School_bus).__name__ extracts the name of the class, which is "Bus"
```

```
The class of the object School_bus is: Bus
School_bus is an instance of the Bus class
School_bus is also an instance of the Vehicle class
```

[14]:

```
# Modeling a Bank Account

# First, identify the attributes and behaviors of a bank account:
# Attributes: account number, account holder name, balance
# Behaviors: depositing, withdrawing, checking balance
class BankAccount:
    def __init__(self, account_number, name, balance):
        self.account_number = account_number
        self.name = name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
        else:
            print("Insufficient funds")

    def check_balance(self):
        print(f"Balance: {self.balance}")

acct1 = BankAccount("1234", "John Doe", 500)
acct2 = BankAccount("2345", "Jane Doe", 100)

acct1.deposit(100)
acct1.check_balance()
acct2.withdraw(200)
```

```
Balance: 600
Insufficient funds
```

[15]:

```
# Modeling Students and Courses
```

```python
# Used in academic systems to model students, courses, grades, etc. This exercise builds classes for students and courses.
# First, identify attributes and behaviors:
# Student:
# Attributes: name, id, list of courses taken
# Behaviors: enrolling in a course, dropping a course, viewing courses taken

# Course:
# Attributes: course title, instructor, max students
# Behaviors: adding students, dropping students, viewing enrolled students
class Course:
    def __init__(self, title, instructor, max_students):
        self.title = title
        self.instructor = instructor
        self.max_students = max_students
        self.students = []

    def add_student(self, student):
        if len(self.students) < self.max_students:
            self.students.append(student)
            student.courses.append(self)
        else:
            print(f"Course {self.title} batch is full.")

    def drop_student(self, student):
        if student in self.students:
            self.students.remove(student)
            student.courses.remove(self)

    def __str__(self):
        return f"Course: {self.title}, Instructor: {self.instructor}, Enrolled: {[s.name for s in self.students]}"

class Student:
    def __init__(self, name, student_id):
        self.name = name
        self.student_id = student_id
        self.courses = []

    def __str__(self):
        return f"Student: {self.name}, ID: {self.student_id}, Courses: {[c.title for c in self.courses]}"

course1 = Course("Math 101", "Dr. Smith", 2)
course2 = Course("Physics 101", "Dr. Johnson", 2)

student1 = Student("Alice", 1)
```

```
student2 = Student("Bob", 2)
student3 = Student("Charlie", 3)

course1.add_student(student1)
course1.add_student(student2)
course1.add_student(student3)

print(student1)
print(course1)

course1.drop_student(student1)
print(student1)
print(course1)
```

```
Course Math 101 batch is full.
Student: Alice, ID: 1, Courses: ['Math 101']
Course: Math 101, Instructor: Dr. Smith, Enrolled: ['Alice', 'Bob']
Student: Alice, ID: 1, Courses: []
Course: Math 101, Instructor: Dr. Smith, Enrolled: ['Bob']
```

[7]:

```
# OOP is very useful for developing graphical applications. This exercise␣
 ↪builds a multiple choice quiz app with a GUI using Python’s tkinter module␣
 ↪and OOP principles
import tkinter as tk

class Quiz:
    def __init__(self, window):
        self.window = window
        self.window.title("Math Quiz")
        self.window.geometry("600x400")

        # Quiz data
        self.questions = [
            {"question": "What is 2 + 4?", "options": ["5", "7", "6"], "answer":
 ↪ "6"},
            {"question": "What is 10 - 9?", "options": ["1", "3", "2"],␣
 ↪"answer": "1"}
        ]
        self.current_question = 0
        self.score = 0

        # GUI components
        self.question_label = tk.Label(window, text="", font=('Arial', 14))
        self.question_label.pack(pady=20)

        self.option_buttons = []
        for i in range(3):
            btn = tk.Button(window, text="", font=('Arial', 12), command=lambda␣
 ↪i=i: self.check_answer(i))
            btn.pack(pady=10)
            self.option_buttons.append(btn)

        self.display_question()

    def display_question(self):
        # Get current question data
        question_data = self.questions[self.current_question]
        question_text = question_data["question"]
        options = question_data["options"]

        # Update GUI with current question and options
        self.question_label.config(text=question_text)
        for i, option in enumerate(options):
            self.option_buttons[i].config(text=option)

    def check_answer(self, index):
        # Check if the selected option is correct
        selected_option = self.option_buttons[index].cget("text")
        correct_answer = self.questions[self.current_question]["answer"]
        if selected_option == correct_answer:
            self.score += 1

        # Move to the next question or end the quiz
        self.current_question += 1
        if self.current_question < len(self.questions):
            self.display_question()
        else:
            self.end_quiz()

    def end_quiz(self):
        # Show the final score and hide the buttons
        self.question_label.config(text=f"Quiz over! Final score: {self.score}")
        for btn in self.option_buttons:
            btn.pack_forget()

# Main program
if __name__ == "__main__":
    window = tk.Tk()
    quiz = Quiz(window)
    window.mainloop()
```

[6]:

```
# OOP is commonly used in game development. This exercise builds a basic␣
 ↪turn-based strategy game in Python using OOP principles.
```

```python
# The game has a map with different kinds of tiles. Each player has a team of characters that can move and perform actions on the map.
class MapTile:
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Character:
    def __init__(self, name, health, attack, defense, x, y):
        self.name = name
        self.health = health
        self.attack = attack
        self.defense = defense
        self.x = x
        self.y = y

    def move(self, dx, dy):
        self.x += dx
        self.y += dy

    def is_alive(self):
        return self.health > 0

    def __str__(self):
        return f"{self.name}: (Health: {self.health}, Position: ({self.x}, {self.y}))"

class Plains(MapTile):
    pass

class Forest(MapTile):
    def __init__(self, x, y):
        super().__init__(x, y)

class Warrior(Character):
    def attack_enemy(self, enemy):
        damage = max(self.attack - enemy.defense, 0)  # Ensure damage is not negative
        enemy.health -= damage
        print(f"{self.name} attacks {enemy.name} for {damage} damage!")

class Archer(Character):
    def ranged_attack(self, enemy):
        damage = max(self.attack - enemy.defense, 0)  # Ensure damage is not negative
        enemy.health -= damage
        print(f"{self.name} performs a ranged attack on {enemy.name} for␣
 ↪{damage} damage!")

# Initialize map
game_map = [
    [Plains(0,0), Forest(1,0), Plains(2,0)],
    [Forest(0,1), Plains(1,1), Plains(2,1)],
    [Plains(0,2), Plains(1,2), Plains(2,2)]
]

# Initialize characters
warrior = Warrior("Jon", 100, 20, 10, 0, 0)
archer = Archer("Arya", 80, 15, 5, 2, 0)

# Game loop
while True:
    # Display character states
    print(warrior)
    print(archer)

    # Handle player actions
    warrior.move(1, 0)
    archer.ranged_attack(warrior)

    # Enemy actions
    if not warrior.is_alive():
        print("Game over! Warrior is dead.")
        break
    if not archer.is_alive():
        print("Game over! Archer is dead.")
        break
```

```
Jon: (Health: 100, Position: (0, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 95, Position: (1, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 90, Position: (2, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 85, Position: (3, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 80, Position: (4, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
```

```
Jon: (Health: 75, Position: (5, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 70, Position: (6, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 65, Position: (7, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 60, Position: (8, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 55, Position: (9, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 50, Position: (10, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 45, Position: (11, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 40, Position: (12, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 35, Position: (13, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 30, Position: (14, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 25, Position: (15, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 20, Position: (16, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 15, Position: (17, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 10, Position: (18, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Jon: (Health: 5, Position: (19, 0))
Arya: (Health: 80, Position: (2, 0))
Arya performs a ranged attack on Jon for 5 damage!
Game over! Warrior is dead.
```

[503]:

```
# Using Polymorphism in Python
# calculate_area() instance method created in both Circle and Rectangle class
class Circle:
    pi = 3.14

    def __init__(self, redius):
        self.radius = redius

    def calculate_area(self):
        print("Area of circle :", self.pi * self.radius * self.radius)

class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def calculate_area(self):
        print("Area of Rectangle :", self.length * self.width)

def area(shape):
    # call action
    shape.calculate_area()

cir = Circle(5)
rect = Rectangle(10, 5)

area(cir)
area(rect)
```

```
Area of circle : 78.5
Area of Rectangle : 50
```

[1]:

```
# Bank Account Management using @classmethod , @staticmethod
# Create a BankAccount class to manage customers' bank accounts. Each account␣
 ↪should have an account number, account holder's name, and balance
class BankAccount:
    bank_name = "ABC Bank"

    def __init__(self, account_number, holder_name, balance=0):
        self.account_number = account_number
        self.holder_name = holder_name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited {amount}. New balance: {self.balance}")
    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
            print(f"Withdrew {amount}. New balance: {self.balance}")
        else:
            print("Insufficient funds")

    @classmethod
    def get_bank_name(cls):
        return cls.bank_name

    @staticmethod
    def calculate_interest(amount, rate, years):
        return amount * (1 + rate/100) ** years

# Example usage
account1 = BankAccount("1234567890", "John Doe", 500)
account1.deposit(200)
account1.withdraw(100)
print(f"Name of BANK: {BankAccount.get_bank_name()}")
print(f"Calculated Interest: {BankAccount.calculate_interest(1000, 5, 3)}")
```

```
Deposited 200. New balance: 700
Withdrew 100. New balance: 600
Name of BANK: ABC Bank
Calculated Interest: 1157.6250000000002
```

[10]:

```
# Online Shopping System using @classmethod , @staticmethod
# Create a Product class to manage an online store's products. Each product␣
 ↪should have a name, price, and stock quantity
class Product:
    discount_rate = 0

    def __init__(self, name, price, stock):
        self.name = name
        self.price = price
        self.stock = stock

    def purchase(self, quantity):
        if self.stock >= quantity:
            self.stock -= quantity
            print(f"Purchased {quantity} of {self.name}. Remaining stock: {self.
 ↪stock}")
        else:
            print("Insufficient stock")

    @classmethod
    def set_discount_rate(cls, rate):
        cls.discount_rate = rate

    @staticmethod
    def apply_discount(price, discount_rate):
        return price * (1 - discount_rate / 100)

Product.set_discount_rate(10)
product1 = Product("Laptop", 1000, 50)
product1.purchase(5)
print(Product.apply_discount(product1.price, Product.discount_rate))
```

```
Purchased 5 of Laptop. Remaining stock: 45
900.0
```

[13]:

```
# Library Management System using @classmethod , @staticmethod
# Create a Book class to manage a library's books. Each book should have a␣
 ↪title, author, and the number of copies available
class Book:
    library_name = "Library"

    def __init__(self, title, author, copies):
        self.title = title
        self.author = author
        self.copies = copies

    def borrow(self):
        if self.copies > 0:
            self.copies -= 1
            print(f"Borrowed {self.title}. Remaining copies: {self.copies}")
        else:
            print("No copies available")

    def return_book(self):
        self.copies += 1
        print(f"Returned {self.title}. Available copies: {self.copies}")

    @classmethod
    def set_library_name(cls, name):
        cls.library_name = name

    @staticmethod
    def is_valid_isbn(isbn):
        return len(str(isbn)) == 13

Book.set_library_name("City Library")
book1 = Book("The Great Gatsby", "F. Scott Fitzgerald", 3)
```

```
book1.borrow()
book1.return_book()
print(Book.is_valid_isbn(1234567890123))
```

```
Borrowed The Great Gatsby. Remaining copies: 2
Returned The Great Gatsby. Available copies: 3
True
```

[5]:

```
# Write a Python program to create a class representing a shopping cart.␣
 ↪Include methods for adding and removing items, and calculating the total␣
 ↪price
class ShoppingCart:
    def __init__(self):
        self.items = {}  # Using a dictionary to store items

    def add_item(self, name, price, quantity):
        if name in self.items:
            self.items[name]['quantity'] += quantity  # Update quantity if item␣
 ↪already exists
        else:
            self.items[name] = {'price': price, 'quantity': quantity}  # Add␣
 ↪new item
        print(f"Added {quantity} x {name} at ${price} each to the cart.")

    def remove_item(self, name):
        if name in self.items:
            del self.items[name]  # Remove item from the dictionary
            print(f"Removed {name} from the cart.")
        else:
            print(f"Item {name} not found in the cart.")

    def calculate_total(self):
        total = 0
        for item in self.items.values():
            total += item['price'] * item['quantity']
        return total

    def __str__(self):
        cart_contents = "Shopping Cart:\n"
        if not self.items:
            cart_contents += "The cart is empty."
        else:
            for name, details in self.items.items():
                cart_contents += f" {details['quantity']} x {name} at␣
 ↪${details['price']} each\n"
        return cart_contents
```

```python
# Example usage:
cart = ShoppingCart()
cart.add_item('Apple', 0.5, 4)
cart.add_item('Banana', 0.3, 6)
print(cart)
print(f"Total: ${cart.calculate_total():.2f}")
cart.remove_item('Apple')
print(cart)
print(f"Total: ${cart.calculate_total():.2f}")

# __str__ method returns a string representation of the shopping cart's contents, making it easy to print the cart
```

```
Added 4 x Apple at $0.5 each to the cart.
Added 6 x Banana at $0.3 each to the cart.
Shopping Cart:
 4 x Apple at $0.5 each
 6 x Banana at $0.3 each

Total: $3.80
Removed Apple from the cart.
Shopping Cart:
 6 x Banana at $0.3 each

Total: $1.80
```

[6]:

```python
# Write a Python program to create a calculator class. Include methods for basic arithmetic operations
class Calculator:
    def add(self, a, b):
        """Returns the sum of a and b."""
        return a + b

    def subtract(self, a, b):
        """Returns the difference of a and b."""
        return a - b

    def multiply(self, a, b):
        """Returns the product of a and b."""
        return a * b

    def divide(self, a, b):
        """Returns the quotient of a and b. Raises an error if b is zero."""
        if b == 0:
            raise ValueError("Cannot divide by zero.")
        return a / b
```

```
calculator = Calculator()

result_add = calculator.add(10, 5)
print(f"10 + 5 = {result_add}")

result_subtract = calculator.subtract(10, 5)
print(f"10 - 5 = {result_subtract}")

result_multiply = calculator.multiply(10, 5)
print(f"10 * 5 = {result_multiply}")

result_divide = calculator.divide(10, 5)
print(f"10 / 5 = {result_divide}")

try:
    calculator.divide(10, 0)
except ValueError as e:
    print(e)

# 'try-except' block in Python is used for handling exceptions, which are␣
 ↪errors that occur during the execution of a program
# The main purpose of using a try-except block is to prevent the program from␣
 ↪crashing and to provide a way to handle errors gracefully
# lines such as """Returns the quotient of a and b. Raises an error if b is␣
 ↪zero.""" in the code are known as docstrings.
# Important for Clarity, Usage Guidance, Automatic Documentation, Code␣
 ↪Maintenance
# If you erase the docstrings, the function will still work the same way␣
 ↪because docstrings do not affect the execution of the code
```

```
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2.0
Cannot divide by zero.
```

[2]:

```
# PYTHON NUMPY EXERCISE
# Create a 4X2 integer array and Prints its attributes
import numpy as np

# Create a 4x2 integer array
array = np.array([[1, 2], [3, 4], [5, 6], [7, 8]], dtype=int)

# Print the attributes of the array
print("Array:\n", array)
print("Shape:", array.shape)
print("Dimensions:", array.ndim)
print("Size:", array.size)
print("Data type:", array.dtype)
print("Item size (in bytes):", array.itemsize)
print("Total size (in bytes):", array.nbytes)
```

```
Array:
 [[1 2]
 [3 4]
 [5 6]
 [7 8]]
Shape: (4, 2)
Dimensions: 2
Size: 8
Data type: int64
Item size (in bytes): 8
Total size (in bytes): 64
```

[27]:

```
# Create a 5X2 integer array from a range between 100 to 200 such that the␣
 ↪difference between each element is 10
import numpy as np
array = np.arange(100, 200, 10).reshape(5, 2)

print("5x2 integer array:")
print(array)
```

```
5x2 integer array:
[[100 110]
 [120 130]
 [140 150]
 [160 170]
 [180 190]]
```

[28]:

```
# Following is the provided numPy array. Return array of items by taking the␣
 ↪third column from all rows
import numpy as np
sampleArray = np.array([[11, 22, 33], [44, 55, 66], [77, 88, 99]])

third_column = sampleArray[:, 2]
print("Third column from all rows:")
print(third_column)

# sampleArray[:, 2] uses slicing to select all rows (:) and the third column (2)
```

```
Third column from all rows:
[33 66 99]
```

[1]:

```python
# Return array of odd rows and even columns from numpy array
import numpy as np
sampleArray = np.array([[3, 6, 9, 12],
                        [15, 18, 21, 24],
                        [27, 30, 33, 36],
                        [39, 42, 45, 48],
                        [51, 54, 57, 60]])

result_array = sampleArray[::2, 1::2]

print("Array of odd rows and even columns:")
print(result_array)

# `sampleArray[::2, 1::2]`in NumPy means: "Select every second row starting␣
 ↪from the beginning, and from those rows, select every second column starting␣
 ↪from the second column
```

```
Array of odd rows and even columns:
[[ 6 12]
 [30 36]
 [54 60]]
```

[1]:

```python
# Create a result array by adding the following two NumPy arrays. Next, modify␣
 ↪the result array by calculating the square of each element
import numpy as np

array1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
array2 = np.array([[9, 8, 7], [6, 5, 4], [3, 2, 1]])

result_array = array1 + array2
print("Result of addition:")
print(result_array)

squared_array = result_array ** 2
print("Squared result array:")
print(squared_array)
```

```
Result of addition:
[[10 10 10]
 [10 10 10]
 [10 10 10]]
Squared result array:
[[100 100 100]
 [100 100 100]
 [100 100 100]]
```

[2]:

```
# Split the array into four equal-sized sub-arrays
# Note: Create an 8X3 integer array from a range between 10 to 34 such that the␣
 ↪difference between each element is 1 and then Split the array into four␣
 ↪equal-sized sub-arrays
import numpy as np
array = np.arange(10, 34).reshape(8, 3)
sub_arrays = np.array_split(array, 4)

print("Original 8x3 array:")
print(array)

print("\nFour equal-sized sub-arrays:")
for i, sub_array in enumerate(sub_arrays):
    print(f"Sub-array {i+1}:\n{sub_array}\n")

# np.array_split function in NumPy is used to split an array into multiple␣
 ↪sub-arrays.
# The syntax is: numpy.array_split(ary, indices_or_sections, axis=0) axis = 0␣
 ↪for ROWS
# indices_or_sections: If an integer, N, the array will be divided into N equal␣
 ↪arrays along the specified axis
# enumerate function in Python adds a counter to an iterable and returns it as␣
 ↪an enumerate object. This can be useful for getting the index of elements in␣
 ↪a loop.
# The syntax is: enumerate(iterable, start=0)
# iterable: An object that supports iteration.
# start: The starting index of the counter. Default is 0.
```

```
Original 8x3 array:
[[10 11 12]
 [13 14 15]
 [16 17 18]
 [19 20 21]
 [22 23 24]
 [25 26 27]
 [28 29 30]
 [31 32 33]]

Four equal-sized sub-arrays:
Sub-array 1:
[[10 11 12]
 [13 14 15]]

Sub-array 2:
[[16 17 18]
 [19 20 21]]
```

```
Sub-array 3:
[[22 23 24]
 [25 26 27]]

Sub-array 4:
[[28 29 30]
 [31 32 33]]
```

[4]:

```python
# Print max from axis 0 and min from axis 1 from the following 2-D array
import numpy as np
sample_array = np.array([[34,43,73],[82,22,12],[53,94,66]])

print("Original array:")
print(sample_array)

max_in_columns = np.max(sample_array, axis=0)
min_in_rows = np.min(sample_array, axis=1)

print("\nMaximum values from each column (axis 0):")
print(max_in_columns)

print("\nMinimum values from each row (axis 1):")
print(min_in_rows)
```

```
Original array:
[[34 43 73]
 [82 22 12]
 [53 94 66]]

Maximum values from each column (axis 0):
[82 94 73]

Minimum values from each row (axis 1):
[34 12 53]
```

[10]:

```python
# Delete the second column from a given array and insert the following new␣
  ↪column in its place
import numpy as np
sampleArray = np.array([[34, 43, 73],
                        [82, 22, 12],
                        [53, 94, 66]])

newColumn = np.array([[10, 10, 10]])

print("Original array:")
print(sampleArray)
```

```
sampleArray = np.delete(sampleArray, 1, axis=1)
print("\nArray after deleting the second column:")
print(sampleArray)

resultArray = np.insert(sampleArray, 1, newColumn, axis=1)

print("\nArray after inserting the new column in place of the deleted column:")
print(resultArray)

# np.delete function removes elements from an array along a specified axis
# The syntax is: numpy.delete(arr, obj, axis=None)
# obj: Indices of elements to remove.  Can be a single integer, a list of␣
 ↪integers, or a slice object
# np.insert function inserts values into an array at specified indices along a␣
 ↪specified axis.
# The syntax is: numpy.insert(arr, obj, values, axis=None)
# obj: Index or indices before which values are inserted. Can be a single␣
 ↪integer, a list of integers, or a slice object
```

```
Original array:
[[34 43 73]
 [82 22 12]
 [53 94 66]]

Array after deleting the second column:
[[34 73]
 [82 12]
 [53 66]]

Array after inserting the new column in place of the deleted column:
[[34 10 73]
 [82 10 12]
 [53 10 66]]
```

[11]:

```
# Write a NumPy program to find the dot product of two arrays of different␣
 ↪dimensions
import numpy as np

array1 = np.array([[1, 2, 3],
                   [4, 5, 6]])
array2 = np.array([[7, 8],
                   [9, 10],
                   [11, 12]])

result = np.dot(array1, array2)
print("Dot product of array1 and array2:")
print(result)
```

```
Dot product of array1 and array2:
[[ 58  64]
 [139 154]]
```

[12]:

```
# Write a NumPy program to create a 3x3 array with random values and subtract␣
 ↪the mean of each row from each element
import numpy as np
array = np.random.rand(3, 3)
row_means = np.mean(array, axis=1).reshape(-1, 1)

result = array - row_means
print("Original array:")
print(array)
print("\nMean of each row:")
print(row_means)
print("\nArray after subtracting the mean of each row from each element:")
print(result)

# In .reshape(-1,1):
# -1 is a placeholder that tells NumPy to automatically calculate the size of␣
 ↪that dimension based on the other dimensions and the total size of the␣
 ↪original array
# 1 specifies that there should be one column
# result of np.mean(array, axis=1) is: [[0.44272273],[0.61672385],[0.6149264]]
# Reshaping it with .reshape(-1, 1) gives:
# [[0.44272273],
# [0.61672385],
# [0.6149264]]
```

```
Original array:
[[0.95325444 0.18752611 0.18738764]
 [0.87681475 0.8069058  0.16645099]
 [0.98874369 0.32635792 0.5296776 ]]

Mean of each row:
[[0.44272273]
 [0.61672385]
 [0.6149264 ]]

Array after subtracting the mean of each row from each element:
[[ 0.51053171 -0.25519662 -0.25533509]
 [ 0.26009091  0.19018195 -0.45027285]
 [ 0.37381728 -0.28856848 -0.0852488 ]]
```

[15]:

```python
# Write a NumPy program to create a 5x5 array with random values and normalize it row-wise and column-wise
import numpy as np
array = np.random.rand(5, 5)

# Normalize row-wise
# Calculating the mean and standard deviation for each row
row_means = array.mean(axis=1).reshape(-1, 1)
row_stds = array.std(axis=1).reshape(-1, 1)
row_normalized = (array - row_means) / row_stds

# Normalize column-wise
# Calculating the mean and standard deviation for each column
col_means = array.mean(axis=0)
col_stds = array.std(axis=0)
col_normalized = (array - col_means) / col_stds

print("Original array:")
print(array)
print("\nRow-wise normalized array:")
print(row_normalized)
print("\nColumn-wise normalized array:")
print(col_normalized)
```

```
Original array:
[[0.98919152 0.39096092 0.94276794 0.62214898 0.02838466]
 [0.676815   0.35427651 0.33779027 0.75037403 0.26643583]
 [0.33129145 0.3506952  0.93058791 0.43435207 0.16996051]
 [0.18892908 0.362895   0.21217769 0.39426089 0.61914297]
 [0.13220931 0.3496321  0.16723058 0.86709959 0.3405181 ]]

Row-wise normalized array:
[[ 1.10287761 -0.56955316  0.9730945   0.07676289 -1.58318184]
 [ 1.0151789  -0.62464347 -0.70846136  1.3891614  -1.07123547]
 [-0.43408613 -0.35893936  1.8868668  -0.03495336 -1.05888795]
 [-1.078004    0.0479861  -0.92752795  0.25100107  1.70654478]
 [-0.90897426 -0.08250808 -0.77585161  1.8844861  -0.11715216]]

Column-wise normalized array:
[[ 1.62255048  1.9056702   1.22618742  0.04704686 -1.30165091]
 [ 0.65805462 -0.48281095 -0.52067157  0.75660682 -0.09363926]
 [-0.40878624 -0.71598584  1.19101787 -0.99216636 -0.58321177]
 [-0.84834524  0.07832941 -0.88337498 -1.21401926  1.69620417]
 [-1.02347361 -0.78520282 -1.01315873  1.40253194  0.28229777]]
```

[16]:

```python
# Write a NumPy program to create a 5x5 array with random values and sort each row and column
```

```python
import numpy as np
array = np.random.rand(5, 5)
sorted_rows = np.sort(array, axis=1)
sorted_columns = np.sort(array, axis=0)

print("Original array:")
print(array)
print("\nRow-sorted array:")
print(sorted_rows)
print("\nColumn-sorted array:")
print(sorted_columns)

# axis=0 (Columns): The operation is performed down each column (the first axis)
# axis=1 (Rows): The operation is performed across each row (the second axis)
```

```
Original array:
[[0.20650013 0.52288908 0.7130276  0.49577991 0.96028396]
 [0.54489744 0.55526681 0.9002697  0.23159231 0.50226098]
 [0.49844121 0.04053189 0.06648593 0.5891797  0.66712936]
 [0.32685685 0.15803469 0.00722939 0.83737271 0.68344151]
 [0.76536385 0.5076951  0.00868161 0.96461711 0.00446798]]

Row-sorted array:
[[0.20650013 0.49577991 0.52288908 0.7130276  0.96028396]
 [0.23159231 0.50226098 0.54489744 0.55526681 0.9002697 ]
 [0.04053189 0.06648593 0.49844121 0.5891797  0.66712936]
 [0.00722939 0.15803469 0.32685685 0.68344151 0.83737271]
 [0.00446798 0.00868161 0.5076951  0.76536385 0.96461711]]

Column-sorted array:
[[0.20650013 0.04053189 0.00722939 0.23159231 0.00446798]
 [0.32685685 0.15803469 0.00868161 0.49577991 0.50226098]
 [0.49844121 0.5076951  0.06648593 0.5891797  0.66712936]
 [0.54489744 0.52288908 0.7130276  0.83737271 0.68344151]
 [0.76536385 0.55526681 0.9002697  0.96461711 0.96028396]]
```

[29]:

```python
# Write a NumPy program to create a 6x6 array with random values and compute␣
 ↪the inverse of the matrix
import numpy as np
matrix = np.random.rand(6, 6)
inverse_matrix = np.linalg.inv(matrix)

print("Original matrix:")
print(matrix)

print("\nInverse matrix:")
print(inverse_matrix)
```

```
# np.linalg module in NumPy contains a variety of linear algebra functions and␣
 ↪utilities
# These functions allow you to perform complex matrix operations, solve linear␣
 ↪equations, and work with various mathematical properties of matrices
# np.linalg.inv(matrix) computes the inverse of the matrix
```

```
Original matrix:
[[0.56862572 0.85418327 0.77985299 0.47369591 0.02322836 0.04931759]
 [0.96160756 0.01604805 0.56520005 0.83916669 0.67619626 0.07158025]
 [0.05903489 0.33150876 0.61256416 0.09893189 0.39201706 0.1131309 ]
 [0.33679365 0.11730827 0.41548131 0.58673108 0.93818264 0.08180591]
 [0.86177438 0.67666756 0.6524815  0.4456739  0.75758906 0.54158888]
 [0.76988529 0.46116614 0.63288422 0.82528372 0.6672375  0.63992516]]

Inverse matrix:
[[-0.19505126  1.14364557 -0.52509334 -0.95000407  1.35147187 -1.04241077]
 [ 1.27039941 -1.42758646 -1.5377394   1.37444834  0.74542221 -0.47294683]
 [-0.39775266  1.04826855  2.68753149 -1.43839284 -0.78578665  0.28719042]
 [ 0.8300662  -0.46475357 -1.08246428  1.08926117 -1.70763252  1.48535688]
 [-0.3447266  -0.37095428 -0.19154512  1.29362974  0.84303692 -0.77693808]
 [-0.99854226 -0.39768062  0.67768073 -1.17861677 -0.06274851  1.76809249]]
```

[68]:

```
# Write a NumPy program to create a 4x4 array with random values and calculate␣
 ↪the determinant
import numpy as np
matrix = np.random.rand(4, 4)
determinant = np.linalg.det(matrix)

print("Original matrix:")
print(matrix)
print("\nDeterminant of the matrix:")
print(determinant)

# np.linalg.det(matrix) computes the determinant of the matrix
```

```
Original matrix:
[[0.22726647 0.42354339 0.55965198 0.61343504]
 [0.58428172 0.95086509 0.54928872 0.34293651]
 [0.25809976 0.03232737 0.02061806 0.2828395 ]
 [0.30642553 0.66719034 0.96337302 0.5906614 ]]

Determinant of the matrix:
0.03191156426218017
```

[61]:

```
# Write a NumPy program to create a 3x3x3 array with random values and flatten␣
 ↪it to a 1D array
```

```python
import numpy as np
array_3d = np.random.random((3, 3, 3))
array_1d = array_3d.flatten()

print("3x3x3 Array with random values:\n", array_3d)
print("\nFlattened 1D Array:\n", array_1d)
```

```
3x3x3 Array with random values:
 [[[0.55307645 0.27843723 0.38743749]
  [0.2018715  0.82504408 0.12893466]
  [0.01698397 0.50704501 0.08930451]]

 [[0.56392978 0.10555409 0.29482896]
  [0.50246788 0.89785826 0.32407555]
  [0.56915236 0.20056267 0.51846979]]

 [[0.09198698 0.25800206 0.01211686]
  [0.61943943 0.83673633 0.89815397]
  [0.85084609 0.63667454 0.16560678]]]

Flattened 1D Array:
 [0.55307645 0.27843723 0.38743749 0.2018715  0.82504408 0.12893466
 0.01698397 0.50704501 0.08930451 0.56392978 0.10555409 0.29482896
 0.50246788 0.89785826 0.32407555 0.56915236 0.20056267 0.51846979
 0.09198698 0.25800206 0.01211686 0.61943943 0.83673633 0.89815397
 0.85084609 0.63667454 0.16560678]
```

[62]:

```python
# Write a NumPy program to create a 4x4 array with random values and extract␣
 ↪the upper triangular part of the matrix and the lower triangular part of the␣
 ↪matrix
import numpy as np

array = np.random.random((4, 4))
upper_triangular = np.triu(array)
lower_triangular = np.tril(array)

print("4x4 Array with random values:\n", array)
print("\nUpper Triangular Part of the Matrix:\n", upper_triangular)
print("\nLower Triangular Part of the Matrix:\n", lower_triangular)

# np.triu(array_4x4) returns the upper triangular part of the 4x4 array, which␣
 ↪includes the diagonal and all elements above it, setting other elements to␣
 ↪zero
# np.tril(array_4x4) returns the lower triangular part of the 4x4 array, which␣
 ↪includes the diagonal and all elements below it, setting other elements to␣
 ↪zero
```

```
4x4 Array with random values:
 [[0.45172837 0.97828052 0.80704306 0.93336332]
 [0.79815621 0.84842863 0.43826179 0.22303767]
 [0.5151425  0.50012653 0.33531199 0.10622906]
 [0.16717417 0.23255741 0.01123529 0.42697622]]

Upper Triangular Part of the Matrix:
 [[0.45172837 0.97828052 0.80704306 0.93336332]
 [0.         0.84842863 0.43826179 0.22303767]
 [0.         0.         0.33531199 0.10622906]
 [0.         0.         0.         0.42697622]]

Lower Triangular Part of the Matrix:
 [[0.45172837 0.         0.         0.        ]
 [0.79815621 0.84842863 0.         0.        ]
 [0.5151425  0.50012653 0.33531199 0.        ]
 [0.16717417 0.23255741 0.01123529 0.42697622]]
```

[67]:

```
# Write a NumPy program to create a 4x4 array with random values and rotate the␣
 ↪array 90 degrees counterclockwise and clockwise , 180 degrees
import numpy as np
array_4x4 = np.random.random((4, 4))

rotated_array_counterclockwise = np.rot90(array_4x4, k=1)
rotated_array_clockwise = np.rot90(array_4x4, k=3)
rotated_array_180_degrees = np.rot90(array_4x4, k=2)

print("Original 4x4 Array with random values:\n", array_4x4)
print("\nArray rotated 90 degrees counterclockwise:\n",␣
 ↪rotated_array_counterclockwise)
print("\nArray rotated 90 degrees clockwise:\n", rotated_array_clockwise)
print("\nArray rotated 180 degrees clockwise:\n", rotated_array_180_degrees)

# np.rot90(array_4x4, k=1) rotates the 4x4 array 270 degrees counterclockwise,␣
 ↪which is equivalent to 90 degrees counter_clockwise
# np.rot90(array_4x4, k=3) rotates the 4x4 array 270 degrees counterclockwise,␣
 ↪which is equivalent to 90 degrees clockwise
```

```
Original 4x4 Array with random values:
 [[0.89183795 0.04817756 0.08371152 0.21619819]
 [0.50370213 0.01917049 0.95901086 0.07367123]
 [0.36120382 0.09593932 0.91570512 0.80996383]
 [0.1367884  0.7192231  0.52919388 0.65296882]]

Array rotated 90 degrees counterclockwise:
 [[0.21619819 0.07367123 0.80996383 0.65296882]
 [0.08371152 0.95901086 0.91570512 0.52919388]
 [0.04817756 0.01917049 0.09593932 0.7192231 ]
 [0.89183795 0.50370213 0.36120382 0.1367884 ]]

Array rotated 90 degrees clockwise:
 [[0.1367884  0.36120382 0.50370213 0.89183795]
 [0.7192231  0.09593932 0.01917049 0.04817756]
 [0.52919388 0.91570512 0.95901086 0.08371152]
 [0.65296882 0.80996383 0.07367123 0.21619819]]

Array rotated 180 degrees clockwise:
 [[0.65296882 0.52919388 0.7192231  0.1367884 ]
 [0.80996383 0.91570512 0.09593932 0.36120382]
 [0.07367123 0.95901086 0.01917049 0.50370213]
 [0.21619819 0.08371152 0.04817756 0.89183795]]
```

[71]:

```python
# Write a NumPy program to create a 4x4 array with random values and shift all␣
  ↪elements one position to the right and left
import numpy as np
array_4x4 = np.random.random((4, 4))

shifted_right = np.roll(array_4x4, shift=1, axis=1)
shifted_left = np.roll(array_4x4, shift=-1, axis=1)

print("Original 4x4 Array with random values:\n", array_4x4)
print("\nArray with elements shifted one position to the right:\n",␣
  ↪shifted_right)
print("\nArray with elements shifted one position to the left:\n", shifted_left)
```

```
Original 4x4 Array with random values:
 [[0.0932131  0.75461176 0.20470384 0.31774873]
 [0.88100928 0.63900003 0.3965915  0.27429217]
 [0.92721093 0.1738047  0.1290139  0.4353815 ]
 [0.07246394 0.07838688 0.20680017 0.61407863]]

Array with elements shifted one position to the right:
 [[0.31774873 0.0932131  0.75461176 0.20470384]
 [0.27429217 0.88100928 0.63900003 0.3965915 ]
 [0.4353815  0.92721093 0.1738047  0.1290139 ]
 [0.61407863 0.07246394 0.07838688 0.20680017]]

Array with elements shifted one position to the left:
 [[0.75461176 0.20470384 0.31774873 0.0932131 ]
 [0.63900003 0.3965915  0.27429217 0.88100928]
 [0.1738047  0.1290139  0.4353815  0.92721093]
 [0.07838688 0.20680017 0.61407863 0.07246394]]
```

[74]:

```
# Write a NumPy program to create a 4x4 array with random values and shift all␣
 ↪elements one position downwards and upwards
import numpy as np
array_4x4 = np.random.random((4, 4))

shifted_down = np.roll(array_4x4, shift=1, axis=0)
shifted_up = np.roll(array_4x4, shift=-1, axis=0)

print("Original 4x4 Array with random values:\n", array_4x4)
print("\nArray with elements shifted one position downwards:\n", shifted_down)
print("\nArray with elements shifted one position upwards:\n", shifted_up)
```

```
Original 4x4 Array with random values:
 [[0.25704513 0.89001867 0.92328007 0.55997578]
 [0.29243044 0.86777834 0.24783799 0.41560666]
 [0.43290738 0.92281618 0.31106083 0.57851774]
 [0.03765825 0.8322112  0.02507477 0.22984515]]

Array with elements shifted one position downwards:
 [[0.03765825 0.8322112  0.02507477 0.22984515]
 [0.25704513 0.89001867 0.92328007 0.55997578]
 [0.29243044 0.86777834 0.24783799 0.41560666]
 [0.43290738 0.92281618 0.31106083 0.57851774]]

Array with elements shifted one position upwards:
 [[0.29243044 0.86777834 0.24783799 0.41560666]
 [0.43290738 0.92281618 0.31106083 0.57851774]
 [0.03765825 0.8322112  0.02507477 0.22984515]
 [0.25704513 0.89001867 0.92328007 0.55997578]]
```

[79]:

```
# Write a NumPy program to count the number of days of specific month taken by␣
 ↪user
import numpy as np
def is_leap_year(year):
    return (year % 4 == 0) and (year % 100 != 0 or year % 400 == 0)

def days_in_month(year, month):
    days_in_month_nonleap = np.array([31, 28, 31, 30, 31, 30,
                                      31, 31, 30, 31, 30, 31])

    if is_leap_year(year):
        days_in_month_nonleap[1] = 29

    return days_in_month_nonleap[month - 1]

year = int(input("Enter the year: "))
month = int(input("Enter the month (1-12): "))
```

```
days = days_in_month(year, month)
print(f"Number of days in {month}/{year}: {days}")
```

```
Number of days in 7/2024: 31
```

[257]:

```
# Write a NumPy program to display all the dates for the month of July, 2024
import numpy as np

start_date = np.datetime64('2024-07-01')
end_date = np.datetime64('2024-08-01')

dates = np.arange(start_date, end_date, dtype='datetime64[D]')
print(dates)

# dtype='datetime64[D]' specifies that we want the dates with day precision,␣
 ↪meaning each element in the array will be a specific day
```

```
['2024-07-01' '2024-07-02' '2024-07-03' '2024-07-04' '2024-07-05'
 '2024-07-06' '2024-07-07' '2024-07-08' '2024-07-09' '2024-07-10'
 '2024-07-11' '2024-07-12' '2024-07-13' '2024-07-14' '2024-07-15'
 '2024-07-16' '2024-07-17' '2024-07-18' '2024-07-19' '2024-07-20'
 '2024-07-21' '2024-07-22' '2024-07-23' '2024-07-24' '2024-07-25'
 '2024-07-26' '2024-07-27' '2024-07-28' '2024-07-29' '2024-07-30'
 '2024-07-31']
```

[100]:

```
# PYTHON PANDAS EXERCISES
# DATASET 01
# Display all the data from the uploaded CSV file as a table
import pandas as pd

file_path = r'c:\Users\ashwi\Downloads\Automobile_data.csv'
df = pd.read_csv(file_path)
df.set_index('index',inplace = True)
df
```

[100]:

```
           company   body-style  wheel-base  length engine-type  \
index
0      alfa-romero  convertible        88.6   168.8        dohc
1      alfa-romero  convertible        88.6   168.8        dohc
2      alfa-romero    hatchback        94.5   171.2        ohcv
3             audi        sedan        99.8   176.6         ohc
4             audi        sedan        99.4   176.6         ohc
…            …          …         …     …         …
81      volkswagen        sedan        97.3   171.7         ohc
82      volkswagen        sedan        97.3   171.7         ohc
86      volkswagen        sedan        97.3   171.7         ohc
```

```
87           volvo        sedan       104.3   188.8         ohc
88           volvo        wagon       104.3   188.8         ohc

      num-of-cylinders  horsepower  average-mileage    price
index
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
…                 …         …              …       …
81                four          85               27   7975.0
82                four          52               37   7995.0
86                four         100               26   9995.0
87                four         114               23  12940.0
88                four         114               23  13415.0

[61 rows x 9 columns]
```

[114]:

```python
# From the given dataset print the first 10 rows
import pandas as pd

file_path = r'c:\Users\ashwi\Downloads\Automobile_data.csv'
df = pd.read_csv(file_path)
df.set_index('index',inplace = True)
df.head(10)
```

[114]:

```
           company   body-style  wheel-base  length engine-type  \
index
0      alfa-romero  convertible        88.6   168.8        dohc
1      alfa-romero  convertible        88.6   168.8        dohc
2      alfa-romero    hatchback        94.5   171.2        ohcv
3             audi        sedan        99.8   176.6         ohc
4             audi        sedan        99.4   176.6         ohc
5             audi        sedan        99.8   177.3         ohc
6             audi        wagon       105.8   192.7         ohc
9              bmw        sedan       101.2   176.8         ohc
10             bmw        sedan       101.2   176.8         ohc
11             bmw        sedan       101.2   176.8         ohc

      num-of-cylinders  horsepower  average-mileage    price
index
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
```

```
5                 five          110               19  15250.0
6                 five          110               19  18920.0
9                 four          101               23  16430.0
10                four          101               23  16925.0
11                 six          121               21  20970.0
```

[115]:
```python
# From the given dataset print the last 10 rows
import pandas as pd

file_path = r'c:\Users\ashwi\Downloads\Automobile_data.csv'
df = pd.read_csv(file_path)
df.set_index('index',inplace = True)
df.tail(10)
```

[115]:
```
          company body-style  wheel-base  length engine-type num-of-cylinders  \
index
69         toyota      wagon        95.7   169.7         ohc             four
70         toyota      wagon        95.7   169.7         ohc             four
71         toyota      wagon        95.7   169.7         ohc             four
79         toyota      wagon       104.5   187.8        dohc              six
80     volkswagen      sedan        97.3   171.7         ohc             four
81     volkswagen      sedan        97.3   171.7         ohc             four
82     volkswagen      sedan        97.3   171.7         ohc             four
86     volkswagen      sedan        97.3   171.7         ohc             four
87          volvo      sedan       104.3   188.8         ohc             four
88          volvo      wagon       104.3   188.8         ohc             four

       horsepower  average-mileage    price
index
69             62               31   6918.0
70             62               27   7898.0
71             62               27   8778.0
79            156               19  15750.0
80             52               37   7775.0
81             85               27   7975.0
82             52               37   7995.0
86            100               26   9995.0
87            114               23  12940.0
88            114               23  13415.0
```

[255]:
```python
# Print most expensive car's company name and price from this datasheet
import pandas as pd

file_path = r'c:\Users\ashwi\Downloads\Automobile_data.csv'
df = pd.read_csv(file_path)
df.set_index('index',inplace = True)
```

```
most_expensive_car = df.loc[df['price'].idxmax()]

most_expensive_car_company = most_expensive_car['company']
most_expensive_car_price = most_expensive_car['price']

print("Most expensive Car's company:", most_expensive_car_company)
print("Price:", most_expensive_car_price)

# df['price']: This selects the 'price' column from the DataFrame df
# .idxmax(): This function is called on the Series obtained from df['price'].␣
 ↪It returns the index of the first occurrence of the maximum value in the␣
 ↪Series
# df.loc[...]: The .loc() method is used to access a group of rows and columns␣
 ↪by labels or a boolean array
```

```
Most expensive Car's company: mercedes-benz
Price: 45400.0
```

[123]:

```
# Print All Toyota Cars details from thia
import pandas as pd

file_path = r'c:\Users\ashwi\Downloads\Automobile_data.csv'
df = pd.read_csv(file_path)
df.set_index('index',inplace = True)

toyota_cars = df.loc[df['company'] == 'toyota']
toyota_cars
```

[123]:

```
      company body-style  wheel-base  length engine-type num-of-cylinders  \
index
66     toyota  hatchback        95.7   158.7         ohc             four
67     toyota  hatchback        95.7   158.7         ohc             four
68     toyota  hatchback        95.7   158.7         ohc             four
69     toyota      wagon        95.7   169.7         ohc             four
70     toyota      wagon        95.7   169.7         ohc             four
71     toyota      wagon        95.7   169.7         ohc             four
79     toyota      wagon       104.5   187.8        dohc              six

       horsepower  average-mileage    price
index
66             62               35   5348.0
67             62               31   6338.0
68             62               31   6488.0
69             62               31   6918.0
70             62               27   7898.0
71             62               27   8778.0
79            156               19  15750.0
```

[141]:

```
# Count total cars per company from this datasheet
import pandas as pd

file_path = r'c:\Users\ashwi\Downloads\Automobile_data.csv'
df = pd.read_csv(file_path)
df.set_index('index',inplace = True)

car_counts = df.groupby('company').size()
car_counts

# df.groupby('company') splits the DataFrame df into groups where each group␣
 ↪contains rows that have the same value in the 'company' column
# Applying .size() function returns a Series where the index is the unique␣
 ↪values of 'Make', and the values are the counts of occurrences of each␣
 ↪unique 'company' value in the original DataFrame.
```

[141]:

```
company
alfa-romero      3
audi             4
bmw              6
chevrolet        3
dodge            2
honda            3
isuzu            3
jaguar           3
mazda            5
mercedes-benz    4
mitsubishi       4
nissan           5
porsche          3
toyota           7
volkswagen       4
volvo            2
dtype: int64
```

[137]:

```
# Find each company's Higesht price car
import pandas as pd

file_path = r'c:\Users\ashwi\Downloads\Automobile_data.csv'
df = pd.read_csv(file_path)
df.set_index('index',inplace = True)

idx = df.groupby('company')['price'].idxmax()
highest_price_cars = df.loc[idx]
highest_price_cars
```

[137]:

```
             company   body-style  wheel-base  length engine-type  \
index
1        alfa-romero  convertible        88.6   168.8        dohc
6               audi        wagon       105.8   192.7         ohc
14               bmw        sedan       103.5   193.8         ohc
18         chevrolet        sedan        94.5   158.8         ohc
19             dodge    hatchback        93.7   157.3         ohc
28             honda        sedan        96.5   175.4         ohc
30             isuzu        sedan        94.3   170.7         ohc
35            jaguar        sedan       102.0   191.7        ohcv
43             mazda        sedan       104.9   175.0         ohc
47     mercedes-benz      hardtop       112.0   199.2        ohcv
52        mitsubishi        sedan        96.3   172.4         ohc
57            nissan        sedan       100.4   184.6        ohcv
62           porsche  convertible        89.5   168.9        ohcf
79            toyota        wagon       104.5   187.8        dohc
86        volkswagen        sedan        97.3   171.7         ohc
88             volvo        wagon       104.3   188.8         ohc

      num-of-cylinders  horsepower  average-mileage    price
index
1                 four         111               21  16500.0
6                 five         110               19  18920.0
14                 six         182               16  41315.0
18                four          70               38   6575.0
19                four          68               31   6377.0
28                four         101               24  12945.0
30                four          78               24   6785.0
35              twelve         262               13  36000.0
43                four          72               31  18344.0
47               eight         184               14  45400.0
52                four          88               25   8189.0
57                 six         152               19  13499.0
62                 six         207               17  37028.0
79                 six         156               19  15750.0
86                four         100               26   9995.0
88                four         114               23  13415.0
```

[140]:

```python
# Find the average mileage of each car making company
import pandas as pd

file_path = r'c:\Users\ashwi\Downloads\Automobile_data.csv'
df = pd.read_csv(file_path)
df.set_index('index',inplace = True)

average_mileage = df.groupby('company')['average-mileage'].mean()
average_mileage
```

[140]:
```
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
```

[148]:
```python
# Sort all Cars by Price column
import pandas as pd

file_path = r'c:\Users\ashwi\Downloads\Automobile_data.csv'
df = pd.read_csv(file_path)
df.set_index('index',inplace = True)

sorted_cars = df.sort_values(by='price')
sorted_cars
```

[148]:
```
             company body-style  wheel-base  length engine-type  \
index
16         chevrolet  hatchback        88.4   141.1           l
36             mazda  hatchback        93.1   159.1         ohc
66            toyota  hatchback        95.7   158.7         ohc
49        mitsubishi  hatchback        93.7   157.3         ohc
37             mazda  hatchback        93.1   159.1         ohc
…              …        …         …     …         …
14               bmw      sedan       103.5   193.8         ohc
47     mercedes-benz    hardtop       112.0   199.2        ohcv
31             isuzu      sedan        94.5   155.9         ohc
32             isuzu      sedan        94.5   155.9         ohc
63           porsche  hatchback        98.4   175.7       dohcv

      num-of-cylinders  horsepower  average-mileage    price
index
16               three          48               47   5151.0
36                four          68               30   5195.0
```

```
66                 four         62              35   5348.0
49                 four         68              37   5389.0
37                 four         68              31   6095.0
…                 …         …              …       …
14                  six        182              16  41315.0
47                eight        184              14  45400.0
31                 four         70              38      NaN
32                 four         70              38      NaN
63                eight        288              17      NaN

[61 rows x 9 columns]
```

[259]:

```
# Concatenate two data frames using the following conditions
# Create two data frames using the following two dictionaries.
import pandas as pd

GermanCars = {'Company': ['Ford', 'Mercedes', 'BMV', 'Audi'], 'Price': [23845,␣
  ↪171995, 135925 , 71400]}
JapaneseCars = {'Company': ['Toyota', 'Honda', 'Nissan', 'Mitsubishi'], 'Price':
  ↪ [29995, 23600, 61500, 58900]}

df_german = pd.DataFrame(GermanCars, index=[0, 1, 2, 3])
df_japanese = pd.DataFrame(JapaneseCars, index=[0, 1, 2, 3])

# Add a new level to the index for each DataFrame
df_german.index = pd.MultiIndex.from_product([['Germany'], df_german.index])
df_japanese.index = pd.MultiIndex.from_product([['Japan'], df_japanese.index])

df_cars = pd.concat([df_german, df_japanese])
df_cars

# df_cars.index += 1 can't be written cause it  cannot perform __iadd__ with␣
  ↪this index type: MultiIndex
# pd.MultiIndex.from_product([['Germany'], df_german.index]) creates a␣
  ↪multi-level index for df_german with 'Germany' as the first level and the␣
  ↪original index as the second level.
# pd.MultiIndex.from_product([['Japan'], df_japanese.index]) creates a␣
  ↪multi-level index for df_japanese with 'Japan' as the first level and the␣
  ↪original index as the second level.
```

[259]:

```
              Company   Price
Germany 0        Ford   23845
        1    Mercedes  171995
        2         BMV  135925
        3        Audi   71400
Japan   0      Toyota   29995
        1       Honda   23600
```

```
2      Nissan   61500
3  Mitsubishi   58900
```

[252]:

```
# Create two data frames using the following two Dicts, Merge two data frames,␣
↪and append the second data frame as a new column to the first data frame
import pandas as pd

Car_Price = {'Company': ['Toyota', 'Honda', 'BMV', 'Audi'], 'Price': [23845,␣
↪17995, 135925 , 71400]}
car_Horsepower = {'Company': ['Toyota', 'Honda', 'BMV', 'Audi'], 'horsepower':␣
↪[141, 80, 182 , 160]}

df_price = pd.DataFrame(Car_Price)
df_horsepower = pd.DataFrame(car_Horsepower)

df_merged = pd.merge(df_price, df_horsepower, on='Company')
df_merged.index += 1
df_merged

# pd.merge(df_price, df_horsepower, on='Company') merges the two DataFrames on␣
↪the 'Company' column
```

[252]:

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

[245]:

```
# DATASET 02
# Load the Excel file and examine its contents
import pandas as pd

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
↪Worksheets\Netflix-Movies-Sample-Data2.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)
df.index += 1
df
```

[245]:

```
                                      Name  Year Age_Rating  \
1                               Casablanca  1942         PG
2                                   Psycho  1960          R
3                            The Godfather  1972          R
4             Star Wars: Episode IV - A New Hope  1977         PG
5                  E.T. the Extra-Terrestrial  1982         PG
6                  Terminator 2: Judgment Day  1991          R
7                             Forrest Gump  1994      PG-13
```

```
8                                            Titanic   1997     PG-13
9                                         The Matrix   1999         R
10                                         Gladiator   2000         R
11  The Lord of the Rings: The Fellowship of the R…  2001     PG-13
12                                     Spirited Away   2001        PG
13                                   The Dark Knight   2008     PG-13
14                                         Inception   2010     PG-13
15                                      The Avengers   2012     PG-13
16                                  Django Unchained   2012         R
17                                            Frozen   2013        PG
18                                      Interstellar   2014     PG-13
19                                       The Martian   2015     PG-13
20                                        La La Land   2016     PG-13
21                                           Get Out   2017         R
22                                The Shape of Water   2017         R
23                                     Black Panther   2018     PG-13
24                                             Joker   2019         R
25                                          Parasite   2019         R
26                          The Shawshank Redemption   1994         R
27                                      Pulp Fiction   1994         R
28                                  Schindler's List   1993         R
29                          The Silence of the Lambs   1991         R
30                                    The Green Mile   1999         R
31                                        Fight Club   1999         R
32                                      The Prestige   2006     PG-13
33                                      The Departed   2006         R
34                            No Country for Old Men   2007         R
35                               Slumdog Millionaire   2008         R
36                                The Social Network   2010     PG-13
37                          The Grand Budapest Hotel   2014         R
38                                          Whiplash   2014         R
39   Birdman or (The Unexpected Virtue of Ignorance)   2014         R
40                                         Spotlight   2015         R
41                                         Moonlight   2016         R
42         Three Billboards Outside Ebbing, Missouri   2017         R
43                                              Roma   2018         R
44                     Once Upon a Time in Hollywood   2019         R
45                                              1917   2019         R
46                                           Dunkirk   2017     PG-13
47                                      The Revenant   2015         R
48                                           Arrival   2016     PG-13
49                                               Her   2013         R
50                                         Gone Girl   2014         R

      Duration               Category  IMDb_Rating
1    102 mins           Drama/Romance          8.5
2    109 mins         Horror/Thriller          8.5
```

```
3    175 mins          Crime/Drama             9.2
4    121 mins      Action/Adventure            8.6
5    115 mins         Family/Sci-Fi            7.8
6    137 mins         Action/Sci-Fi            8.5
7    142 mins         Drama/Romance            8.8
8    195 mins         Drama/Romance            7.8
9    136 mins         Action/Sci-Fi            8.7
10   155 mins          Action/Drama            8.5
11   178 mins       Adventure/Drama            8.8
12   125 mins   Animation/Adventure            8.6
13   152 mins          Action/Crime            9.0
14   148 mins      Action/Adventure            8.8
15   143 mins      Action/Adventure            8.0
16   165 mins         Drama/Western            8.4
17   102 mins   Animation/Adventure            7.4
18   169 mins       Adventure/Drama            8.6
19   144 mins      Adventure/Sci-Fi            8.0
20   128 mins          Comedy/Drama            8.0
21   104 mins        Horror/Mystery            7.7
22   123 mins       Adventure/Drama            7.3
23   134 mins      Action/Adventure            7.3
24   122 mins           Crime/Drama            8.4
25   132 mins          Comedy/Drama            8.6
26   142 mins                 Drama            9.3
27   154 mins           Crime/Drama            8.9
28   195 mins       Biography/Drama            8.9
29   118 mins           Crime/Drama            8.6
30   189 mins           Crime/Drama            8.6
31   139 mins                 Drama            8.8
32   130 mins         Drama/Mystery            8.5
33   151 mins           Crime/Drama            8.5
34   122 mins           Crime/Drama            8.1
35   120 mins         Drama/Romance            8.0
36   120 mins       Biography/Drama            7.7
37    99 mins      Adventure/Comedy            8.1
38   106 mins           Drama/Music            8.5
39   119 mins          Comedy/Drama            7.7
40   129 mins           Crime/Drama            8.1
41   111 mins                 Drama            7.4
42   115 mins           Crime/Drama            8.2
43   135 mins                 Drama            7.7
44   161 mins          Comedy/Drama            7.6
45   119 mins             Drama/War            8.3
46   106 mins          Action/Drama            7.8
47   156 mins      Action/Adventure            8.0
48   116 mins          Drama/Sci-Fi            7.9
49   126 mins         Drama/Romance            8.0
```

```
50   149 mins          Drama/Mystery          8.1
```

[250]:

```
# Write a pandas code to get all names of column headings in this dataset
import pandas as pd

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Netflix-Movies-Sample-Data2.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

column_headings = df.columns.tolist()
f'Column Headings: {column_headings}'
```

[250]:

```
"Column Headings: ['Name', 'Year', 'Age_Rating', 'Duration', 'Category',
'IMDb_Rating']"
```

[247]:

```
# Write a pandas code for counting no. of movies in the dataset
import pandas as pd

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Netflix-Movies-Sample-Data2.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

num_movies = df.shape[0]
f"Total No. of films are {num_movies}"
```

[247]:

```
'Total No. of films are 50'
```

[207]:

```
# Write a Pandas program to count no. of Movies released before 2000 in this␣
 ↪Datasheet
import pandas as pd

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Netflix-Movies-Sample-Data2.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

movies_before_2000 = df[df['Year'] < 2000].shape[0]
print("Number of movies released before the year 2000:", movies_before_2000)
```

```
Number of movies released before the year 2000: 15
```

[263]:

```
# Write a Pandas program to get unique movie categories / genres in the dataset
import pandas as pd
```

```
file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Netflix-Movies-Sample-Data2.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

unique_genres = df['Category'].unique()
print(unique_genres)
```

```
[' Drama/Romance ' ' Horror/Thriller ' ' Crime/Drama '
 ' Action/Adventure ' ' Family/Sci-Fi ' ' Action/Sci-Fi ' ' Action/Drama '
 ' Adventure/Drama ' ' Animation/Adventure ' ' Action/Crime '
 ' Drama/Western ' ' Adventure/Sci-Fi ' ' Comedy/Drama '
 ' Horror/Mystery ' ' Drama ' ' Biography/Drama ' ' Drama/Mystery '
 ' Adventure/Comedy ' ' Drama/Music ' ' Drama/War ' ' Drama/Sci-Fi ']
```

[212]:

```
# List no. of movies released as per Category
import pandas as pd

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Netflix-Movies-Sample-Data2.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

movie_counts = df.groupby('Category').size()
movie_counts
```

[212]:

```
Category
Action/Adventure        5
Action/Crime            1
Action/Drama            2
Action/Sci-Fi           2
Adventure/Comedy        1
Adventure/Drama         3
Adventure/Sci-Fi        1
Animation/Adventure     2
Biography/Drama         2
Comedy/Drama            4
Crime/Drama             9
Drama                   4
Drama/Music             1
Drama/Mystery           2
Drama/Romance           5
Drama/Sci-Fi            1
Drama/War               1
Drama/Western           1
Family/Sci-Fi           1
Horror/Mystery          1
```

```
Horror/Thriller         1
dtype: int64
```

[218]:

```
# List Movie Name for each category having maximum IMDB rating from datasheet
import pandas as pd

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Netflix-Movies-Sample-Data2.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

df['IMDb_Rating'] = pd.to_numeric(df['IMDb_Rating'], errors='coerce') # Ensure␣
 ↪'IMDB Rating' is numeric
df = df.dropna(subset=['IMDb_Rating']) # Drop rows with NaN values in the 'IMDB␣
 ↪Rating' column

idx = df.groupby('Category')['IMDb_Rating'].idxmax() # movie with the maximum␣
 ↪IMDB rating for each category
top_movies_per_genre = df.loc[idx, ['Name','Category', 'IMDb_Rating']]
top_movies_per_genre

# errors='coerce': If there are any non-numeric values in the 'IMDB Rating'␣
 ↪column, they will be converted to NaN (Not a Number)
# subset=['IMDB Rating']: This parameter specifies that the operation should␣
 ↪only consider the 'IMDB Rating' column when deciding which rows to drop
```

[218]:

```
                                                  Name            Category  \
13                                           Inception    Action/Adventure
12                                     The Dark Knight        Action/Crime
9                                            Gladiator        Action/Drama
8                                           The Matrix       Action/Sci-Fi
36                              The Grand Budapest Hotel    Adventure/Comedy
10  The Lord of the Rings: The Fellowship of the R…      Adventure/Drama
18                                          The Martian    Adventure/Sci-Fi
11                                        Spirited Away  Animation/Adventure
27                                     Schindler's List     Biography/Drama
24                                             Parasite        Comedy/Drama
2                                         The Godfather         Crime/Drama
25                             The Shawshank Redemption               Drama
37                                             Whiplash         Drama/Music
31                                         The Prestige       Drama/Mystery
6                                          Forrest Gump       Drama/Romance
47                                              Arrival        Drama/Sci-Fi
44                                                 1917           Drama/War
15                                     Django Unchained       Drama/Western
4                            E.T. the Extra-Terrestrial       Family/Sci-Fi
20                                              Get Out      Horror/Mystery
```

```
1                                             Psycho        Horror/Thriller

    IMDb_Rating
13          8.8
12          9.0
9           8.5
8           8.7
36          8.1
10          8.8
18          8.0
11          8.6
27          8.9
24          8.6
2           9.2
25          9.3
37          8.5
31          8.5
6           8.8
47          7.9
44          8.3
15          8.4
4           7.8
20          7.7
1           8.5
```

[239]:

```
# List Movie Name, Category, IMDb_Rating by Grouping for each Year with indexing
import pandas as pd

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Netflix-Movies-Sample-Data2.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

df['Year'] = df['Year'].astype(int) # Converting 'Year' column to integer type␣
 ↪(if necessary)
grouped = df.groupby('Year')

output = pd.DataFrame(columns=['Name', 'Category', 'IMDb_Rating', 'Year'])

# Iterating through each group (year) and populate the output DataFrame
for year, group in grouped:
    year_data = group[['Name', 'Category', 'IMDb_Rating']].
 ↪reset_index(drop=True)
    year_data['Year'] = year
    output = pd.concat([output, year_data], ignore_index=True)

# Sorting by 'Year' and within each group by 'IMDB_Rating' in descending order
```

```python
output = output.sort_values(by=['Year', 'IMDb_Rating'], ascending=[True, False])
output.index += 1

# Setting multi-index with 'Year' and a cumulative count within each year
result = output.set_index(['Year', output.groupby('Year').cumcount() + 1])
result
```

```
C:\Users\ashwi\AppData\Local\Temp\ipykernel_20008\2535481366.py:17:
FutureWarning: The behavior of DataFrame concatenation with empty or all-NA
entries is deprecated. In a future version, this will no longer exclude empty or
all-NA columns when determining the result dtypes. To retain the old behavior,
exclude the relevant entries before the concat operation.
  output = pd.concat([output, year_data], ignore_index=True)
```

[239]:

```
                                                     Name  \
Year
1942 1                                         Casablanca
1960 1                                             Psycho
1972 1                                      The Godfather
1977 1                 Star Wars: Episode IV - A New Hope
1982 1                         E.T. the Extra-Terrestrial
1991 1                           The Silence of the Lambs
     2                         Terminator 2: Judgment Day
1993 1                                   Schindler's List
1994 1                           The Shawshank Redemption
     2                                       Pulp Fiction
     3                                       Forrest Gump
1997 1                                            Titanic
1999 1                                         Fight Club
     2                                         The Matrix
     3                                     The Green Mile
2000 1                                          Gladiator
2001 1  The Lord of the Rings: The Fellowship of the R…
     2                                      Spirited Away
2006 1                                       The Prestige
     2                                       The Departed
2007 1                             No Country for Old Men
2008 1                                    The Dark Knight
     2                                Slumdog Millionaire
2010 1                                          Inception
     2                                 The Social Network
2012 1                                   Django Unchained
     2                                       The Avengers
2013 1                                                Her
     2                                             Frozen
2014 1                                       Interstellar
     2                                           Whiplash
     3                         The Grand Budapest Hotel
     4                                        Gone Girl
     5   Birdman or (The Unexpected Virtue of Ignorance)
2015 1                                        Spotlight
     2                                      The Martian
     3                                     The Revenant
2016 1                                       La La Land
     2                                          Arrival
     3                                        Moonlight
2017 1          Three Billboards Outside Ebbing, Missouri
     2                                          Dunkirk
     3                                          Get Out
     4                               The Shape of Water
2018 1                                             Roma
     2                                    Black Panther
2019 1                                         Parasite
     2                                            Joker
     3                                             1917
     4                    Once Upon a Time in Hollywood

                      Category  IMDb_Rating
Year
1942 1          Drama/Romance          8.5
1960 1        Horror/Thriller          8.5
1972 1            Crime/Drama          9.2
1977 1       Action/Adventure          8.6
1982 1          Family/Sci-Fi          7.8
1991 1            Crime/Drama          8.6
     2          Action/Sci-Fi          8.5
1993 1        Biography/Drama          8.9
1994 1                  Drama          9.3
     2            Crime/Drama          8.9
     3          Drama/Romance          8.8
1997 1          Drama/Romance          7.8
1999 1                  Drama          8.8
     2          Action/Sci-Fi          8.7
     3            Crime/Drama          8.6
2000 1           Action/Drama          8.5
2001 1        Adventure/Drama          8.8
     2    Animation/Adventure          8.6
2006 1          Drama/Mystery          8.5
     2            Crime/Drama          8.5
2007 1            Crime/Drama          8.1
2008 1           Action/Crime          9.0
     2          Drama/Romance          8.0
2010 1       Action/Adventure          8.8
     2        Biography/Drama          7.7
```

```
2012 1         Drama/Western           8.4
     2      Action/Adventure           8.0
2013 1         Drama/Romance           8.0
     2   Animation/Adventure           7.4
2014 1       Adventure/Drama           8.6
     2           Drama/Music           8.5
     3      Adventure/Comedy           8.1
     4         Drama/Mystery           8.1
     5          Comedy/Drama           7.7
2015 1           Crime/Drama           8.1
     2      Adventure/Sci-Fi           8.0
     3      Action/Adventure           8.0
2016 1          Comedy/Drama           8.0
     2          Drama/Sci-Fi           7.9
     3                 Drama           7.4
2017 1           Crime/Drama           8.2
     2          Action/Drama           7.8
     3        Horror/Mystery           7.7
     4       Adventure/Drama           7.3
2018 1                 Drama           7.7
     2      Action/Adventure           7.3
2019 1          Comedy/Drama           8.6
     2           Crime/Drama           8.4
     3             Drama/War           8.3
     4          Comedy/Drama           7.6
```

[243]:

```
# List Movie Name, Category, IMDb_Rating by Grouping for each Year and Age␣
 ↪Rating with indexing
import pandas as pd

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Netflix-Movies-Sample-Data2.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

df['Year'] = df['Year'].astype(int) # Converting 'Year' column to integer type␣
 ↪(if necessary)
grouped = df.groupby(['Year', 'Age_Rating'])

output = pd.DataFrame(columns=['Name', 'Category', 'IMDb_Rating', 'Age_Rating',␣
 ↪'Year'])

# Iterating through each group (year, age rating) and populate the output␣
 ↪DataFrame
for (year, age_rating), group in grouped:
    year_age_data = group[['Name', 'Category', 'IMDb_Rating', 'Age_Rating']].
 ↪reset_index(drop=True)
```

```
    year_age_data['Year'] = year
    output = pd.concat([output, year_age_data], ignore_index=True)

# Sorting the output DataFrame by 'Year', 'Age Rating', and then by 'Movie Name'
output = output.sort_values(by=['Year', 'Age_Rating', 'Name'])
output.index = output.index + 1

# Setting multi-index with 'Year' and 'Age Rating',cumulative counting index␣
 ↪from 1
output = output.set_index([ 'Year','Age_Rating', output.
 ↪groupby(['Year','Age_Rating']).cumcount() + 1])
output
```

```
C:\Users\ashwi\AppData\Local\Temp\ipykernel_20008\1994497655.py:17:
FutureWarning: The behavior of DataFrame concatenation with empty or all-NA
entries is deprecated. In a future version, this will no longer exclude empty or
all-NA columns when determining the result dtypes. To retain the old behavior,
exclude the relevant entries before the concat operation.
  output = pd.concat([output, year_age_data], ignore_index=True)
```

[243]:

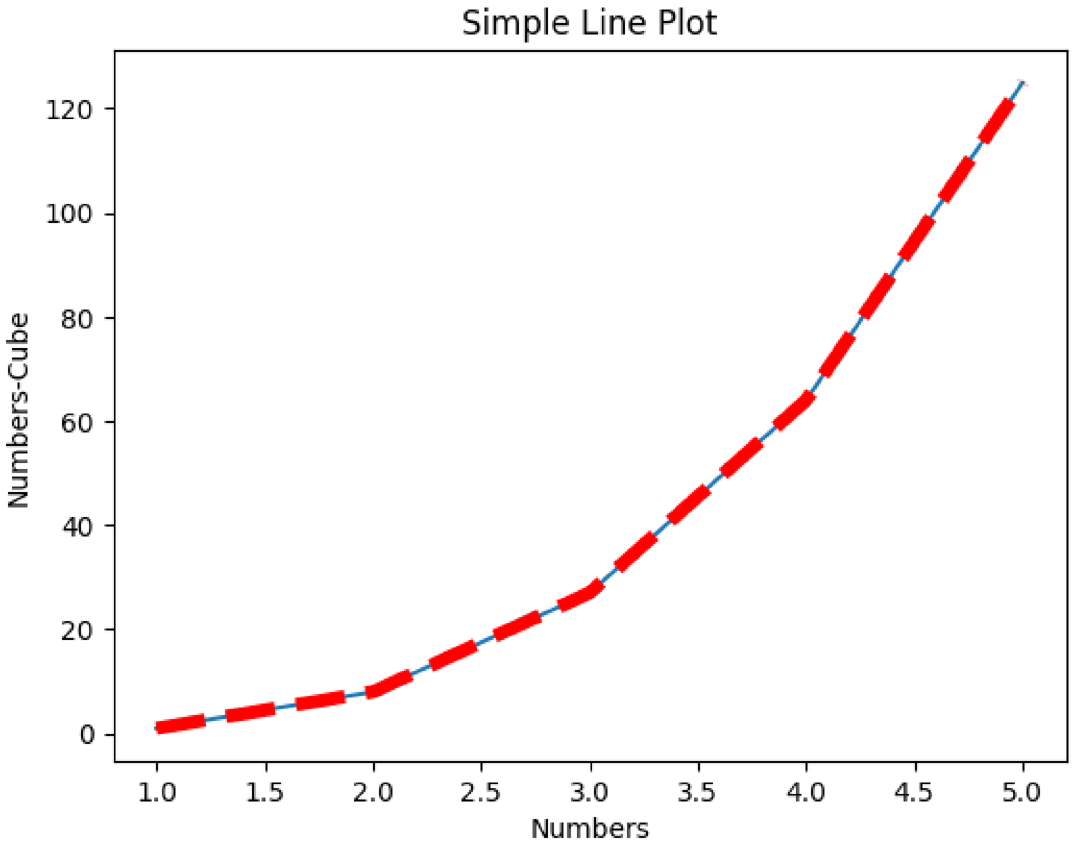
```
                                                                Name  \
Year Age_Rating
1942 PG         1                                         Casablanca
1960 R          1                                             Psycho
1972 R          1                                      The Godfather
1977 PG         1                 Star Wars: Episode IV - A New Hope
1982 PG         1                         E.T. the Extra-Terrestrial
1991 R          1                         Terminator 2: Judgment Day
                2                           The Silence of the Lambs
1993 R          1                                   Schindler's List
1994 PG-13      1                                       Forrest Gump
     R          1                                       Pulp Fiction
                2                           The Shawshank Redemption
1997 PG-13      1                                            Titanic
1999 R          1                                         Fight Club
                2                                     The Green Mile
                3                                         The Matrix
2000 R          1                                          Gladiator
2001 PG         1                                      Spirited Away
     PG-13      1  The Lord of the Rings: The Fellowship of the R…
2006 PG-13      1                                       The Prestige
     R          1                                       The Departed
2007 R          1                             No Country for Old Men
2008 PG-13      1                                    The Dark Knight
     R          1                                Slumdog Millionaire
2010 PG-13      1                                          Inception
                2                                 The Social Network
```

```
2012 PG-13      1                                      The Avengers
     R          1                                  Django Unchained
2013 PG         1                                            Frozen
     R          1                                               Her
2014 PG-13      1                                      Interstellar
     R          1   Birdman or (The Unexpected Virtue of Ignorance)
                2                                         Gone Girl
                3                          The Grand Budapest Hotel
                4                                          Whiplash
2015 PG-13      1                                       The Martian
     R          1                                         Spotlight
                2                                      The Revenant
2016 PG-13      1                                           Arrival
                2                                        La La Land
     R          1                                         Moonlight
2017 PG-13      1                                           Dunkirk
     R          1                                           Get Out
                2                                The Shape of Water
                3         Three Billboards Outside Ebbing, Missouri
2018 PG-13      1                                     Black Panther
     R          1                                              Roma
2019 R          1                                              1917
                2                                             Joker
                3                     Once Upon a Time in Hollywood
                4                                          Parasite

                                Category  IMDb_Rating
Year Age_Rating
1942 PG         1          Drama/Romance          8.5
1960 R          1        Horror/Thriller          8.5
1972 R          1            Crime/Drama          9.2
1977 PG         1       Action/Adventure          8.6
1982 PG         1          Family/Sci-Fi          7.8
1991 R          1          Action/Sci-Fi          8.5
                2            Crime/Drama          8.6
1993 R          1        Biography/Drama          8.9
1994 PG-13      1          Drama/Romance          8.8
     R          1            Crime/Drama          8.9
                2                  Drama          9.3
1997 PG-13      1          Drama/Romance          7.8
1999 R          1                  Drama          8.8
                2            Crime/Drama          8.6
                3          Action/Sci-Fi          8.7
2000 R          1           Action/Drama          8.5
2001 PG         1    Animation/Adventure          8.6
     PG-13      1        Adventure/Drama          8.8
2006 PG-13      1          Drama/Mystery          8.5
```

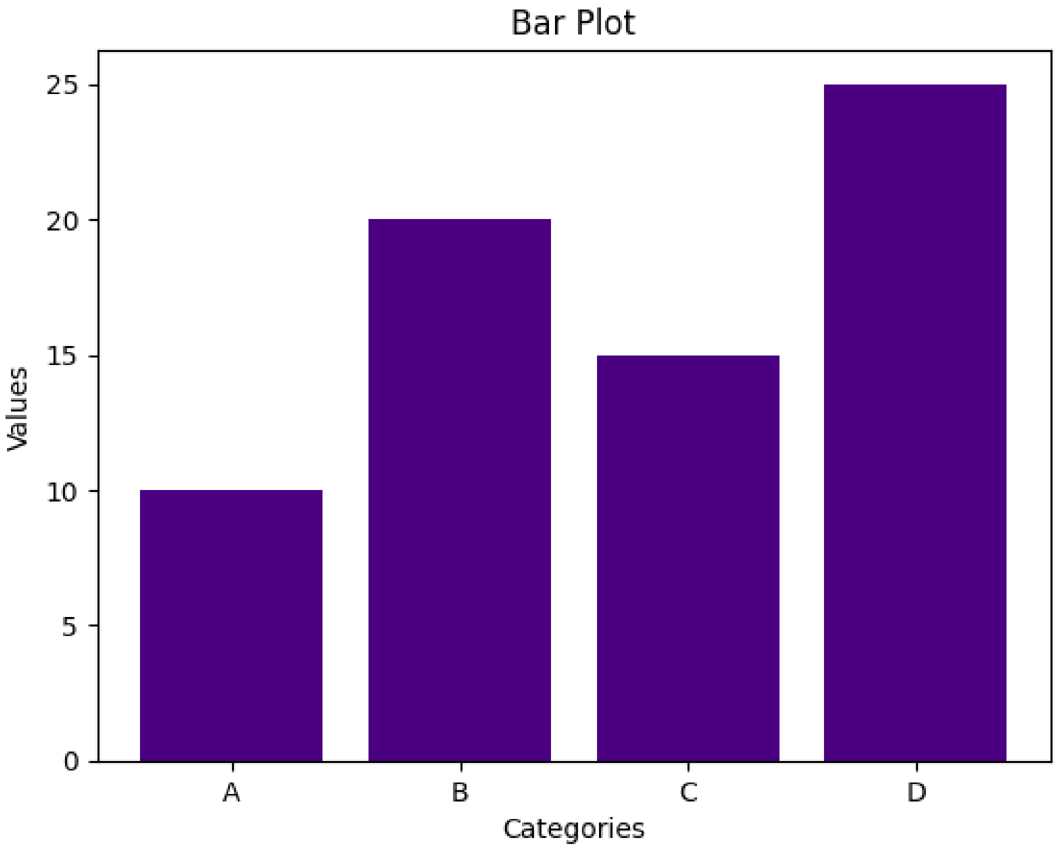
```
     R          1           Crime/Drama           8.5
2007 R          1           Crime/Drama           8.1
2008 PG-13      1          Action/Crime           9.0
     R          1         Drama/Romance           8.0
2010 PG-13      1      Action/Adventure           8.8
                2       Biography/Drama           7.7
2012 PG-13      1      Action/Adventure           8.0
     R          1         Drama/Western           8.4
2013 PG         1   Animation/Adventure           7.4
     R          1         Drama/Romance           8.0
2014 PG-13      1       Adventure/Drama           8.6
     R          1          Comedy/Drama           7.7
                2         Drama/Mystery           8.1
                3      Adventure/Comedy           8.1
                4           Drama/Music           8.5
2015 PG-13      1      Adventure/Sci-Fi           8.0
     R          1           Crime/Drama           8.1
                2      Action/Adventure           8.0
2016 PG-13      1          Drama/Sci-Fi           7.9
                2          Comedy/Drama           8.0
     R          1                 Drama           7.4
2017 PG-13      1          Action/Drama           7.8
     R          1        Horror/Mystery           7.7
                2       Adventure/Drama           7.3
                3           Crime/Drama           8.2
2018 PG-13      1      Action/Adventure           7.3
     R          1                 Drama           7.7
2019 R          1             Drama/War           8.3
                2           Crime/Drama           8.4
                3          Comedy/Drama           7.6
                4          Comedy/Drama           8.6
```

[272]:

```
# PYTHON MATPLOTLIB EXERCISE
# Create a simple Line Plot using given values of x and y
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [1, 8, 27, 64, 125]

plt.plot(x, y)
plt.xlabel('Numbers')
plt.ylabel('Numbers-Cube')
plt.title('Simple Line Plot')
plt.plot(x, y, color='red', linestyle='--', linewidth=5)
plt.show()
```

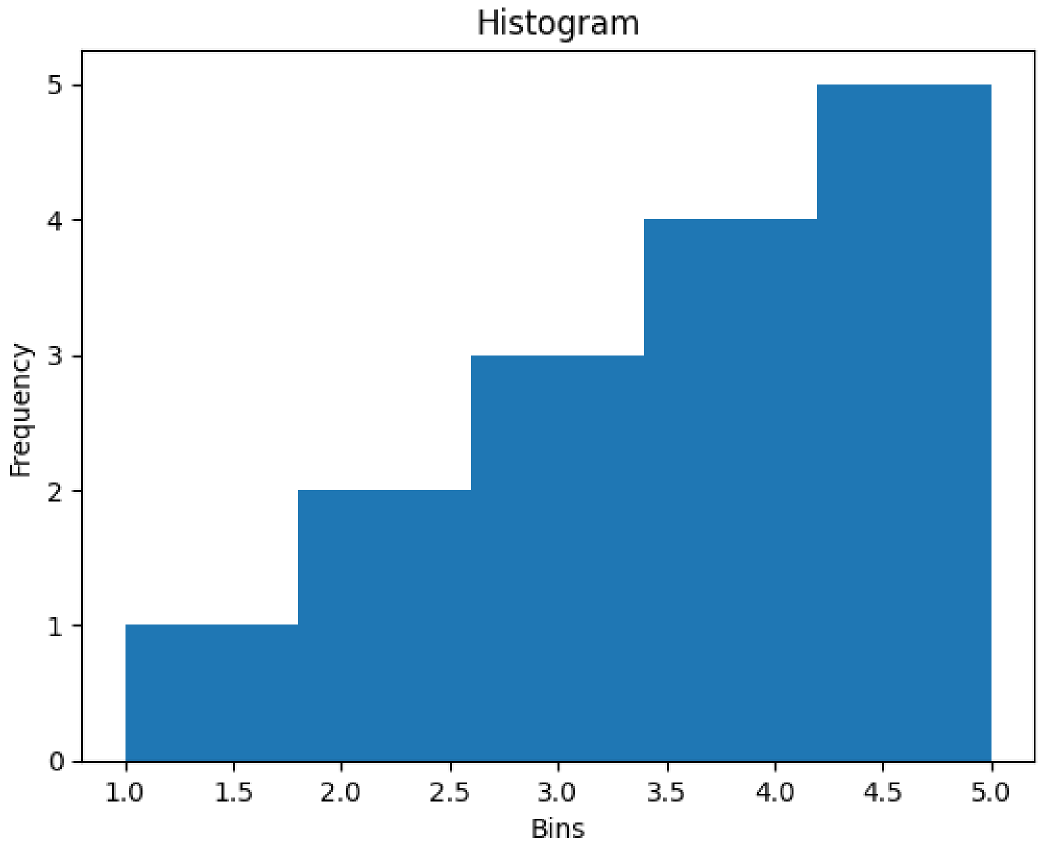


```
[336]: # Create a sample Bar Plot using given values of Categories and Values
import matplotlib.pyplot as plt
categories = ['A', 'B', 'C', 'D']
values = [10, 20, 15, 25]

plt.bar(categories, values, color='indigo')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Plot')
plt.show()
```



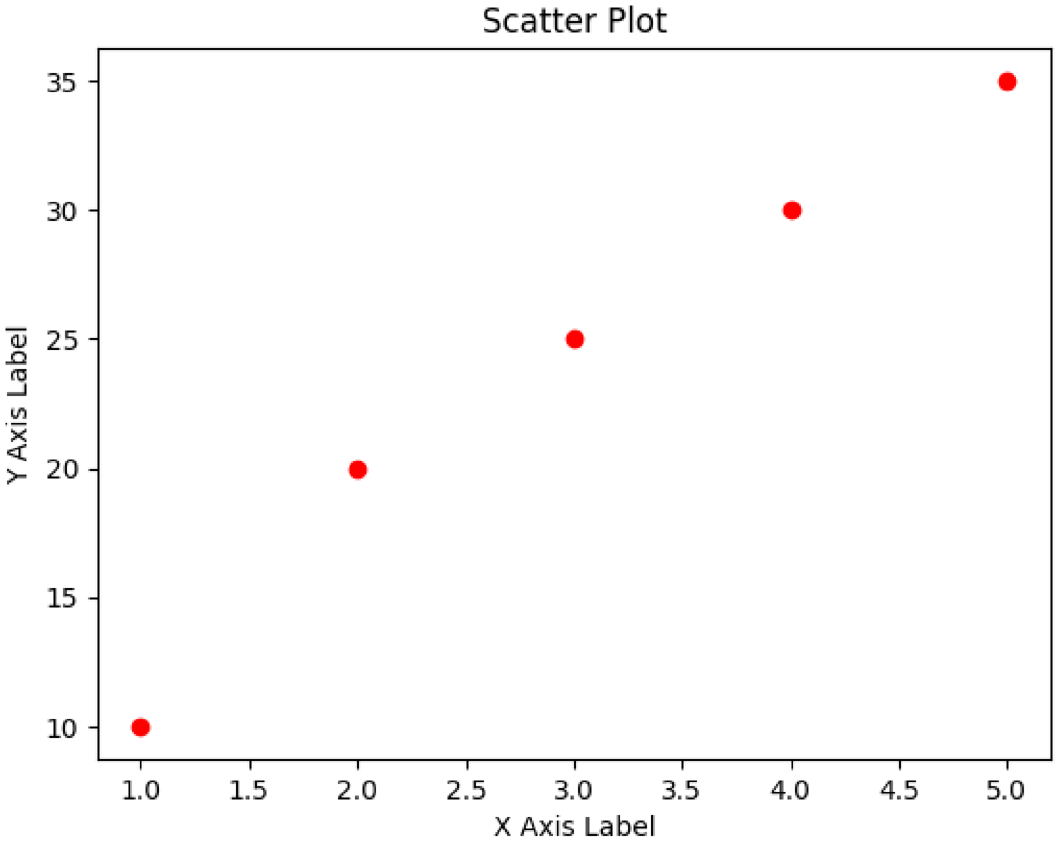
[267]:

```
# Represent the distribution of a dataset in a histogram plot
import matplotlib.pyplot as plt
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5]

plt.hist(data, bins=5)
plt.xlabel('Bins')
plt.ylabel('Frequency')
plt.title('Histogram')
plt.show()
```



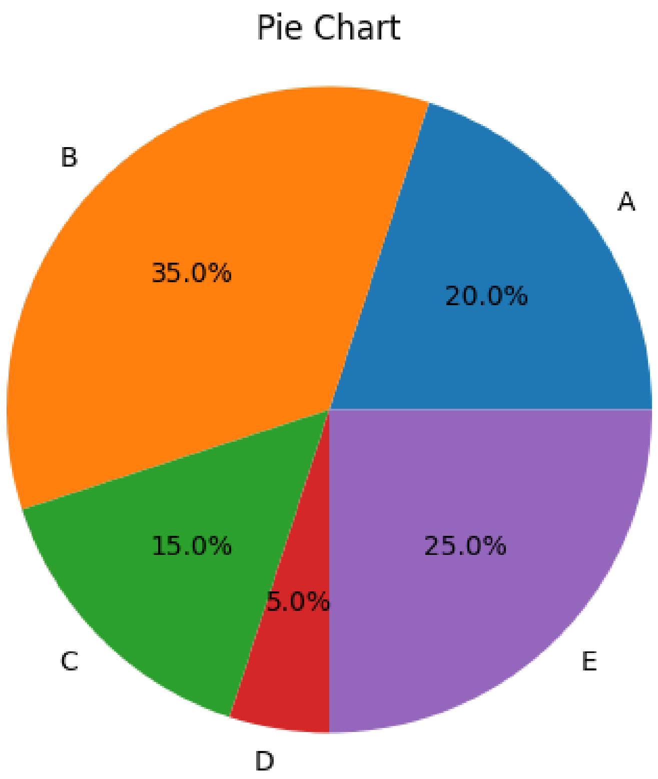
[281]:

```
# Show relationship between two variables using Scatter Plot
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 35]

plt.scatter(x, y, color='red')
plt.xlabel('X Axis Label')
plt.ylabel('Y Axis Label')
plt.title('Scatter Plot')
plt.show()
```



[285]:

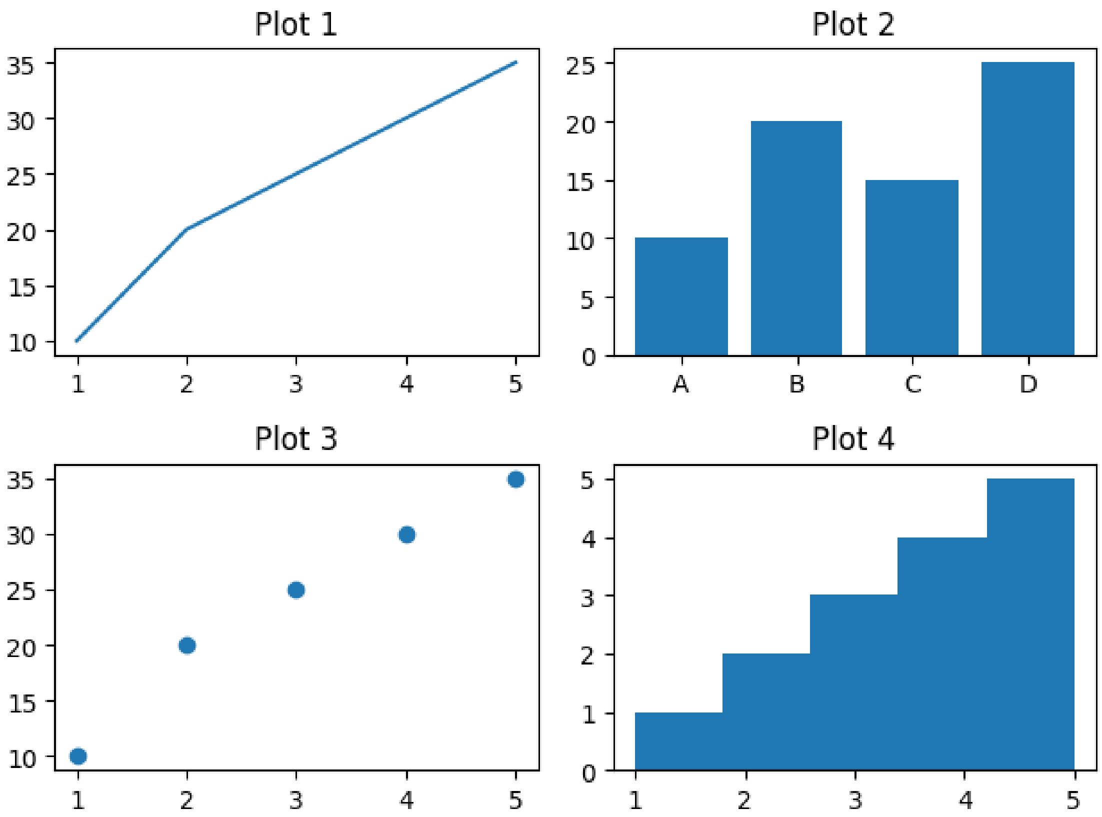
```
# Show the proportion of different categories in a Pie Chart
import matplotlib.pyplot as plt
labels = ['A', 'B', 'C', 'D', 'E']
sizes = [20, 35, 15, 5, 25]

plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=0)
plt.axis('equal')
plt.title('Pie Chart')
plt.show()

# plt.axis('equal') function call ensures that the pie chart is drawn as a␣
↪circle.
# By default, Matplotlib might distort the pie chart into an ellipse if the␣
↪aspect ratio of the plot window is not equal
```



[335]:

```
# Create multiple plots in a single figure  as Subplots
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 35]
categories = ['A', 'B', 'C', 'D']
values = [10, 20, 15, 25]
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5]

fig, axs = plt.subplots(2, 2)

# Plot 1: Line plot in the first subplot
axs[0, 0].plot(x, y)
axs[0, 0].set_title('Plot 1')

# Plot 2: Bar plot in the second subplot
axs[0, 1].bar(categories, values)
axs[0, 1].set_title('Plot 2')

# Plot 3: Scatter plot in the third subplot
axs[1, 0].scatter(x, y)
axs[1, 0].set_title('Plot 3')
```

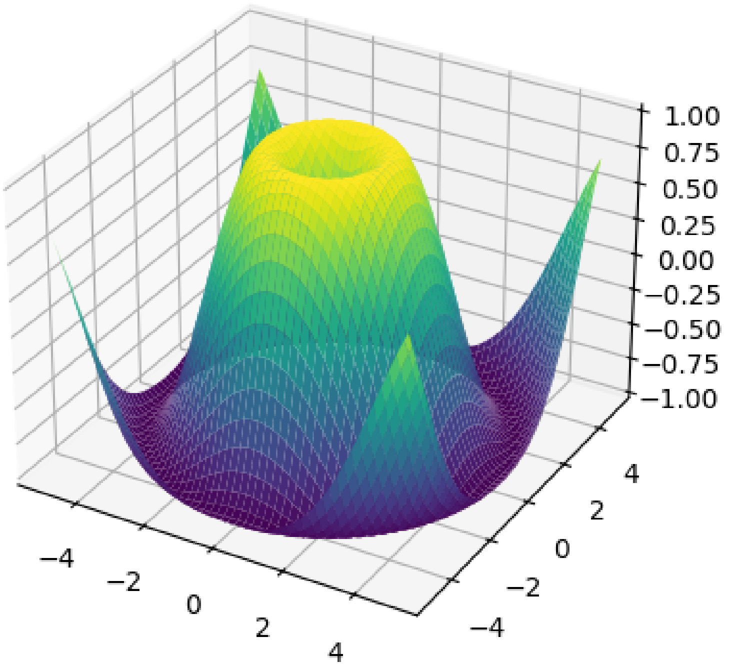
```
# Plot 4: Histogram in the fourth subplot
axs[1, 1].hist(data, bins=5)
axs[1, 1].set_title('Plot 4')

plt.tight_layout()
plt.show()

# tight_layout() method adjusts the spacing between subplots to prevent overlap␣
 ↪of labels, titles, and axes
```



[277]:

```
# Create a 3-D Plot
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure() # Creating a new figure for plotting
ax = fig.add_subplot(111, projection='3d')

x = np.linspace(-5, 5, 100) # Lines generating 100 evenly spaced values between␣
 ↪-5 and 5 for both x and y
```

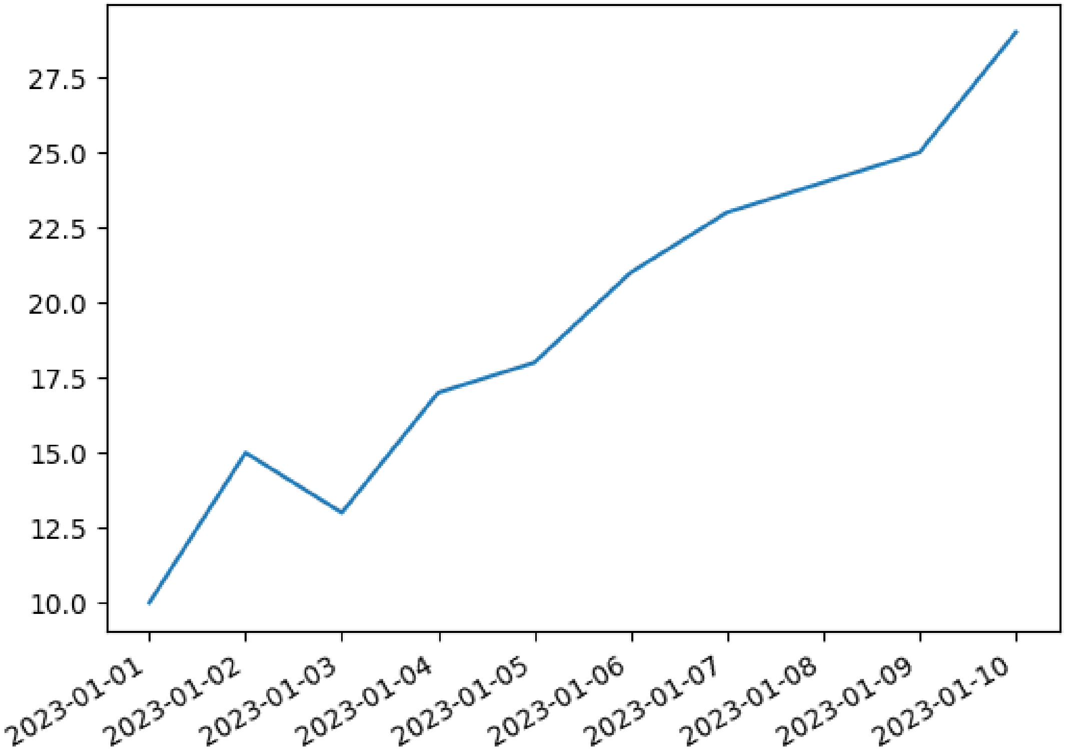
```
y = np.linspace(-5, 5, 100) # Lines generate 100 evenly spaced values between␣
 ↪-5 and 5 for both x and y
x, y = np.meshgrid(x, y) # Creating a coordinate grid from the x and y arrays
z = np.sin(np.sqrt(x**2 + y**2)) # Creating a wave-like surface

ax.plot_surface(x, y, z, cmap='viridis')
plt.show()

# Importing Axes3D from mpl_toolkits.mplot3d to enable 3D plotting
# 3D subplot to the figure. 111 means "1x1 grid, first subplot" . And␣
 ↪projection='3d' argument tells Matplotlib to create a 3D plot.
# np.meshgrid function creates a coordinate grid from the x and y arrays
# z data as a function of x and y computing sine of the square root of the sum␣
 ↪of the squares of x and y. This creates a wave-like surface
# .plot_surface() method creates a 3D surface plot. cmap='viridis' argument␣
 ↪sets the colormap to 'viridis', which is a visually appealing color gradient
```



[286]:

```
# Working with Dates
import matplotlib.dates as mdates
import datetime
import matplotlib.pyplot as plt

dates = [datetime.datetime(2023, 1, i) for i in range(1, 11)]
```

```
values = [10, 15, 13, 17, 18, 21, 23, 24, 25, 29]

plt.plot(dates, values)
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.DayLocator())
plt.gcf().autofmt_xdate()
plt.show()

# plt.gca() stands for "get current axes" and returns the current axes instance
# .xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d')) means the dates␣
 ↪will be displayed in the format "YYYY-MM-DD"
# .xaxis.set_major_locator(mdates.DayLocator()) sets the locator for the major␣
 ↪ticks on the x-axis to a DayLocator ensuring a major tick (and thus a date␣
 ↪label) for each day in the range of dates.
# .autofmt_xdate() automatically formats the x-axis labels to make them more␣
 ↪readable, especially if they are long or if there are many of them.
# It typically rotates the labels and aligns them so they don't overlap
#
```



[289]:

```
# Data Visualization using a CSV Dataset
# Read total data of this CSV Datasheet
import pandas as pd
```

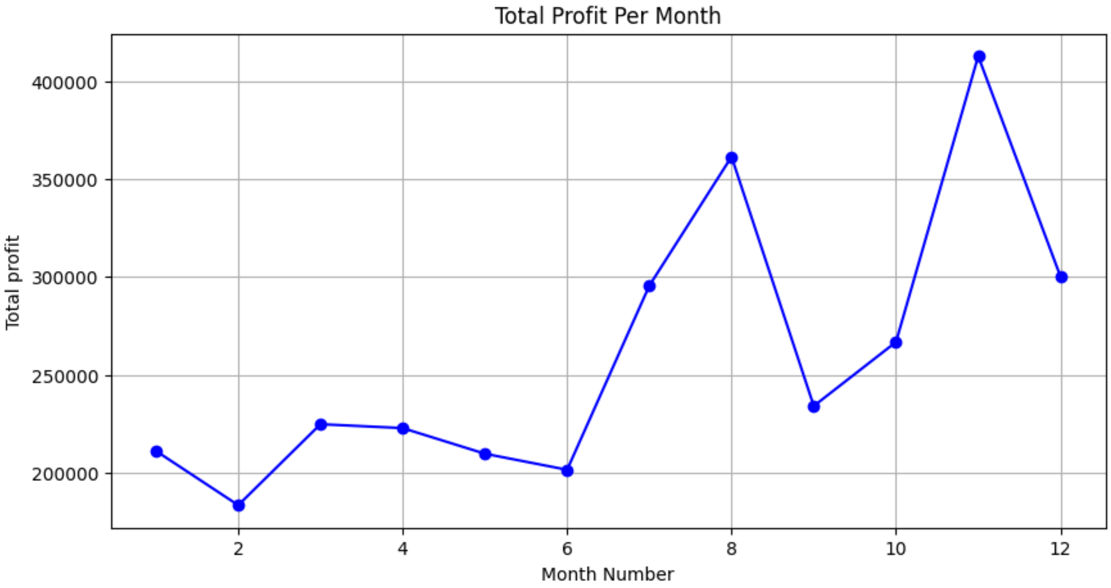
```
file_path = r'c:\Users\ashwi\Downloads\company_sales_data.csv'
df = pd.read_csv(file_path)
df.index += 1
df
```

[289]:

```
    month_number  facecream  facewash  toothpaste  bathingsoap  shampoo  \
1              1       2500      1500        5200         9200     1200
2              2       2630      1200        5100         6100     2100
3              3       2140      1340        4550         9550     3550
4              4       3400      1130        5870         8870     1870
5              5       3600      1740        4560         7760     1560
6              6       2760      1555        4890         7490     1890
7              7       2980      1120        4780         8980     1780
8              8       3700      1400        5860         9960     2860
9              9       3540      1780        6100         8100     2100
10            10       1990      1890        8300        10300     2300
11            11       2340      2100        7300        13300     2400
12            12       2900      1760        7400        14400     1800

    moisturizer  total_units  total_profit
1          1500        21100        211000
2          1200        18330        183300
3          1340        22470        224700
4          1130        22270        222700
5          1740        20960        209600
6          1555        20140        201400
7          1120        29550        295500
8          1400        36140        361400
9          1780        23400        234000
10         1890        26670        266700
11         2100        41280        412800
12         1760        30020        300200
```

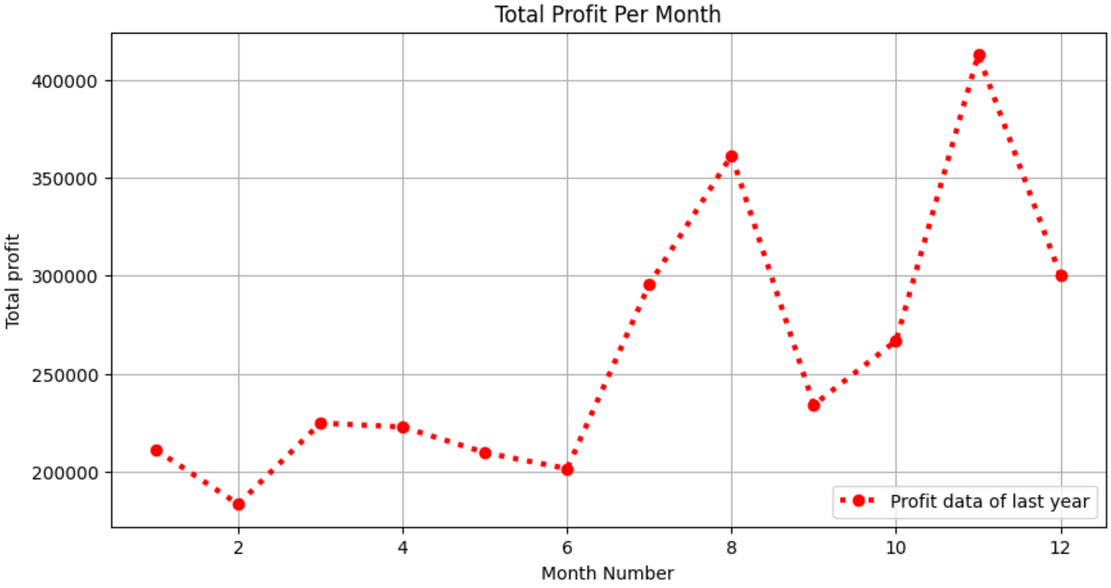
[329]:

```
# Read Total profit of all months and show it using a line plot. Total profit␣
 ↪data provided for each month
import pandas as pd
import matplotlib.pyplot as plt

file_path = r'c:\Users\ashwi\Downloads\company_sales_data.csv'
df = pd.read_csv(file_path)

months = df['month_number']
total_profit = df['total_profit']

# Plotting the line plot
plt.figure(figsize=(10, 5))
plt.plot(months, total_profit, marker='o', linestyle='-', color='blue')
plt.xlabel('Month Number')
plt.ylabel('Total profit')
plt.title('Total Profit Per Month')
plt.grid(True)
plt.show()
```



[295]:

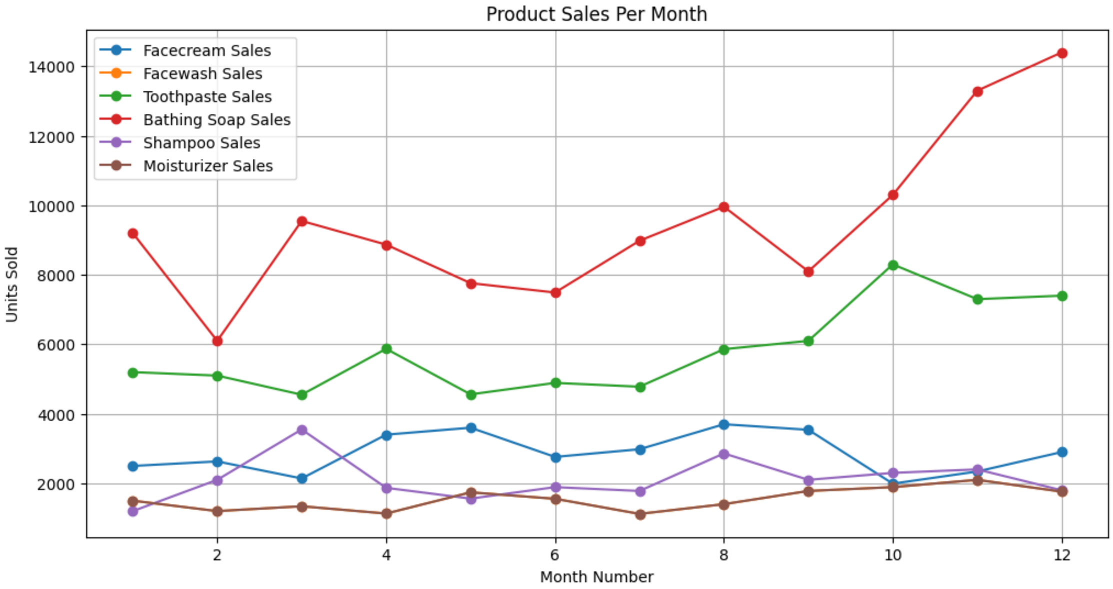
```
# Get total profit of all months and show line plot with the following Style␣
↪properties
# Generated line plot must include following Style properties: -
# Line Style dotted and Line-color should be red
# Show legend at the lower right location.
# X label name = Month Number
# Y label name = Sold units number
# Add a circle marker.
# Line marker color as read
# Line width should be 3
import pandas as pd
import matplotlib.pyplot as plt

file_path = r'c:\Users\ashwi\Downloads\company_sales_data.csv'
df = pd.read_csv(file_path)

plt.figure(figsize=(10, 5))
plt.plot(months, total_profit, linestyle=':', color='r', marker='o',␣
↪markerfacecolor='r', linewidth=3, label='Profit data of last year')
plt.xlabel('Month Number')
plt.ylabel('Total profit')
plt.title('Total Profit Per Month')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



[294]:

```
# Read all product sales data and show it  using a multiline plot
# Display the number of units sold per month for each product using multiline␣
 ↪plots. (i.e., Separate Plotline for each product)
import pandas as pd
import matplotlib.pyplot as plt

file_path = r'c:\Users\ashwi\Downloads\company_sales_data.csv'
df = pd.read_csv(file_path)

facecream_sales = df['facecream']
facewash_sales = df['facewash']
toothpaste_sales = df['toothpaste']
bathingsoap_sales = df['bathingsoap']
shampoo_sales = df['shampoo']
moisturizer_sales = df['moisturizer']

# Plotting the multiline plot
plt.figure(figsize=(12, 6))
plt.plot(months, facecream_sales, marker='o', linestyle='-', label='Facecream␣
 ↪Sales')
```

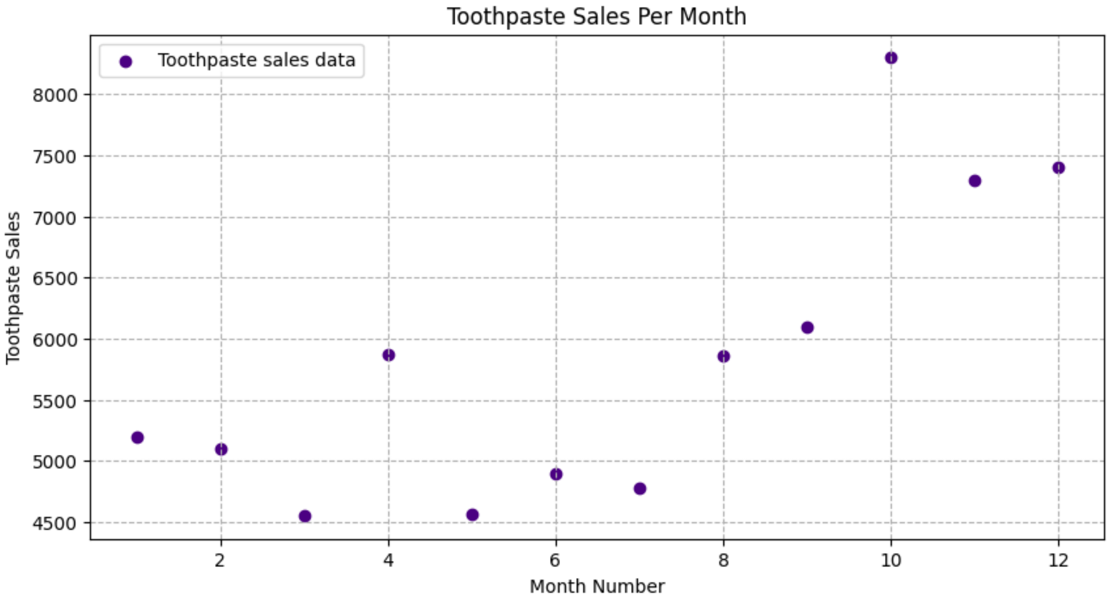
```
plt.plot(months, facewash_sales, marker='o', linestyle='-', label='Facewash Sales')
plt.plot(months, toothpaste_sales, marker='o', linestyle='-', label='Toothpaste Sales')
plt.plot(months, bathingsoap_sales, marker='o', linestyle='-', label='Bathing Soap Sales')
plt.plot(months, shampoo_sales, marker='o', linestyle='-', label='Shampoo Sales')
plt.plot(months, moisturizer_sales, marker='o', linestyle='-', label='Moisturizer Sales')

plt.xlabel('Month Number')
plt.ylabel('Units Sold')
plt.title('Product Sales Per Month')
plt.legend()
plt.grid(True)
plt.show()
```



[328]:

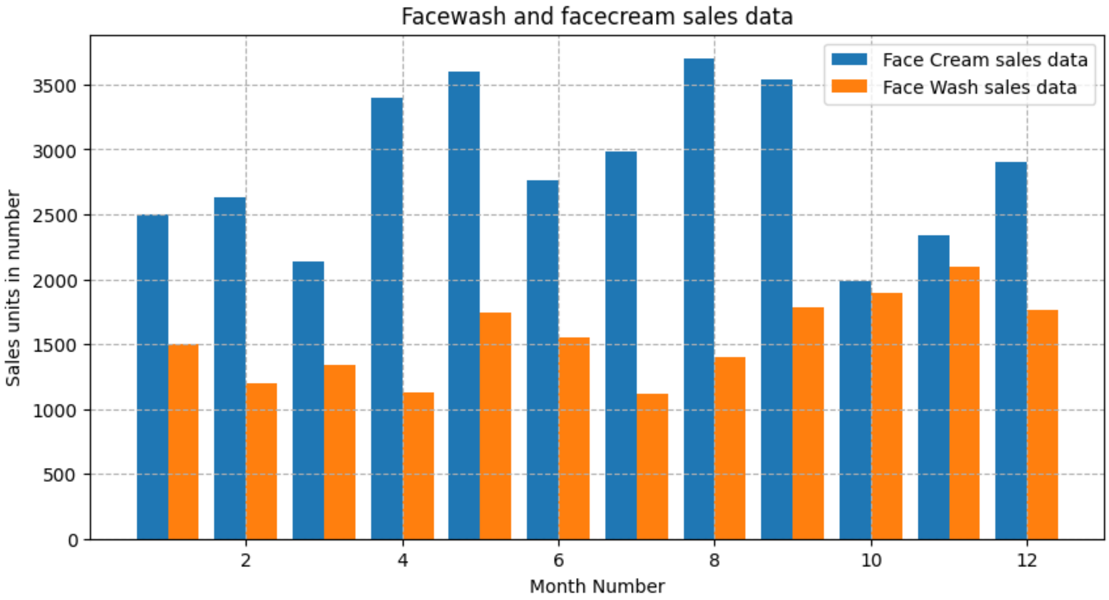
```
# Read toothpaste sales data of each month and show it using a scatter plot
# Also, add a grid in the plot. gridline style should "-"
import pandas as pd
import matplotlib.pyplot as plt

file_path = r'c:\Users\ashwi\Downloads\company_sales_data.csv'
df = pd.read_csv(file_path)

toothpaste_sales = df['toothpaste']
```

```
plt.figure(figsize=(10, 5))
plt.scatter(months, toothpaste_sales, color='indigo',label='Toothpaste sales␣
 ↪data')
plt.xlabel('Month Number')
plt.ylabel('Toothpaste Sales')
plt.title('Toothpaste Sales Per Month')
plt.grid(True, linestyle='--')
plt.legend()
plt.show()
```



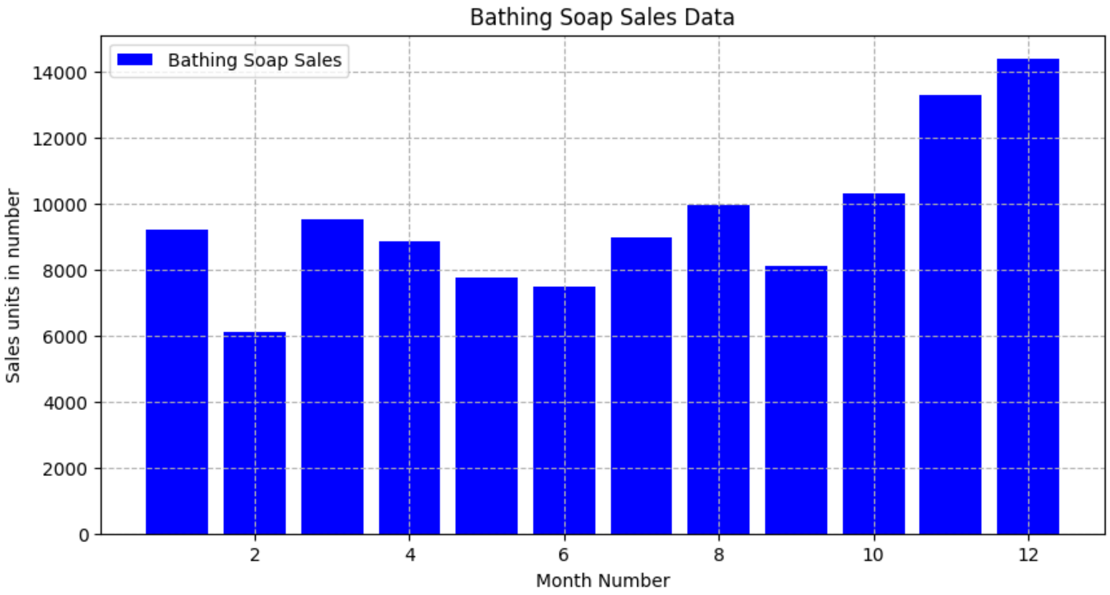
[302]:

```
# Read face cream and facewash product sales data and show it using the bar␣
 ↪chart
# The bar chart should display the number of units sold per month for each␣
 ↪product. Add a separate bar for each product in the same chart
import pandas as pd
import matplotlib.pyplot as plt

file_path = r'c:\Users\ashwi\Downloads\company_sales_data.csv'
df = pd.read_csv(file_path)

months = df['month_number']
facecream_sales = df['facecream']
facewash_sales = df['facewash']

plt.figure(figsize=(10, 5))
plt.bar(months - 0.2, facecream_sales, width=0.4, label='Face Cream sales␣
 ↪data', align='center')
plt.bar(months + 0.2, facewash_sales, width=0.4, label='Face Wash sales data',␣
 ↪align='center')
plt.xlabel('Month Number')
plt.ylabel('Sales units in number')
plt.title('Facewash and facecream sales data')
plt.legend()
plt.grid(True, linestyle='--')
plt.show()
```

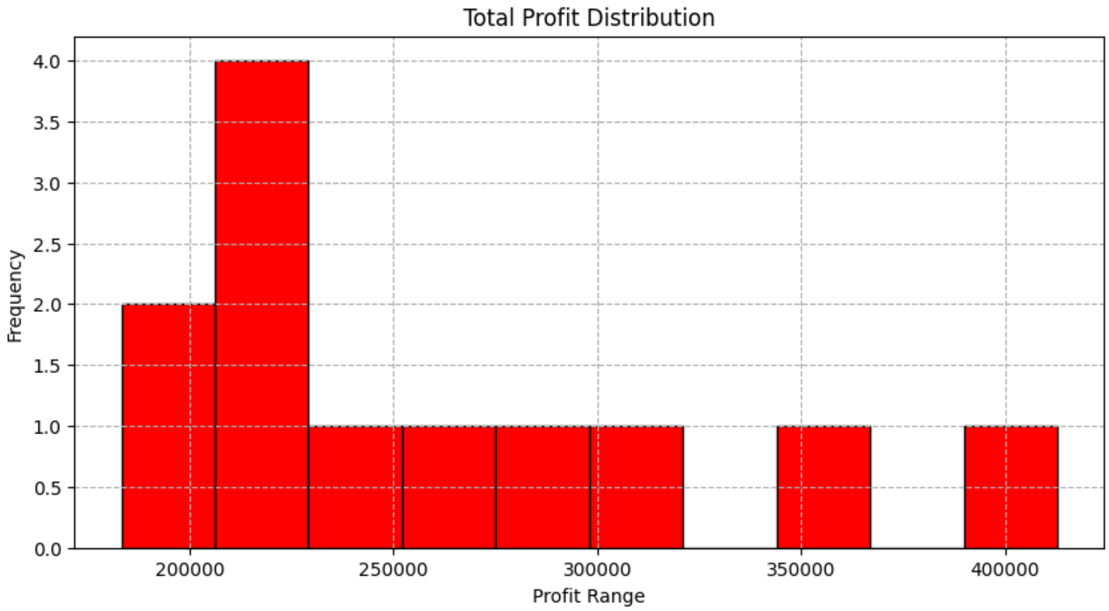


[303]:

```
# Read sales data of bathing soap of all months and show it using a bar chart
import pandas as pd
import matplotlib.pyplot as plt

file_path = r'c:\Users\ashwi\Downloads\company_sales_data.csv'
df = pd.read_csv(file_path)

bathingsoap_sales = df['bathingsoap']

plt.figure(figsize=(10, 5))
plt.bar(months, bathingsoap_sales, color='b', label='Bathing Soap Sales')
plt.xlabel('Month Number')
plt.ylabel('Sales units in number')
plt.title('Bathing Soap Sales Data')
plt.legend()
plt.grid(True, linestyle='--')
```

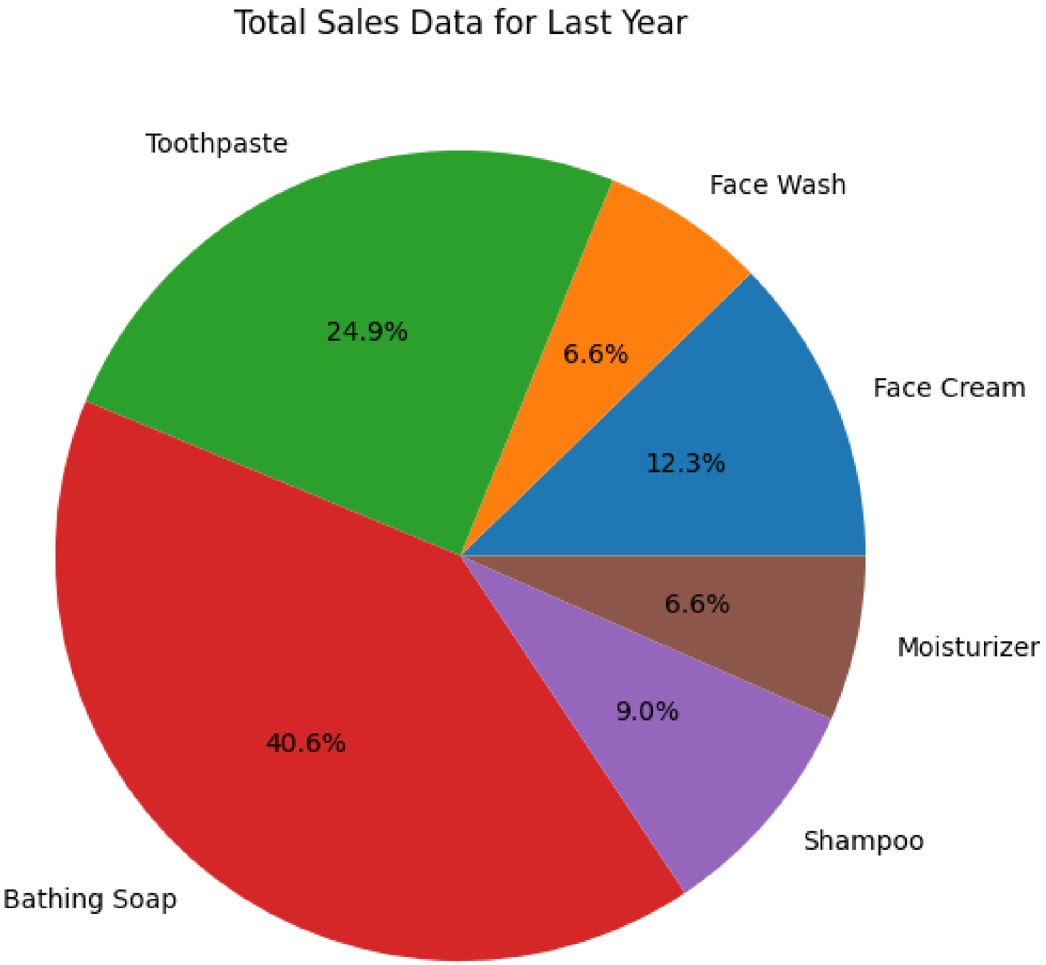


[316]:
```
# Read the total profit of each month and show it using the histogram to see␣
↪the most common profit ranges
import pandas as pd
import matplotlib.pyplot as plt

file_path = r'c:\Users\ashwi\Downloads\company_sales_data.csv'
df = pd.read_csv(file_path)

total_profit = df['total_profit']

plt.figure(figsize=(10, 5))
plt.hist(total_profit, bins=10, color='red', edgecolor='black')
plt.xlabel('Profit Range')
plt.ylabel('Frequency')
plt.title('Total Profit Distribution')
plt.grid(True, linestyle='--')
plt.show()
```



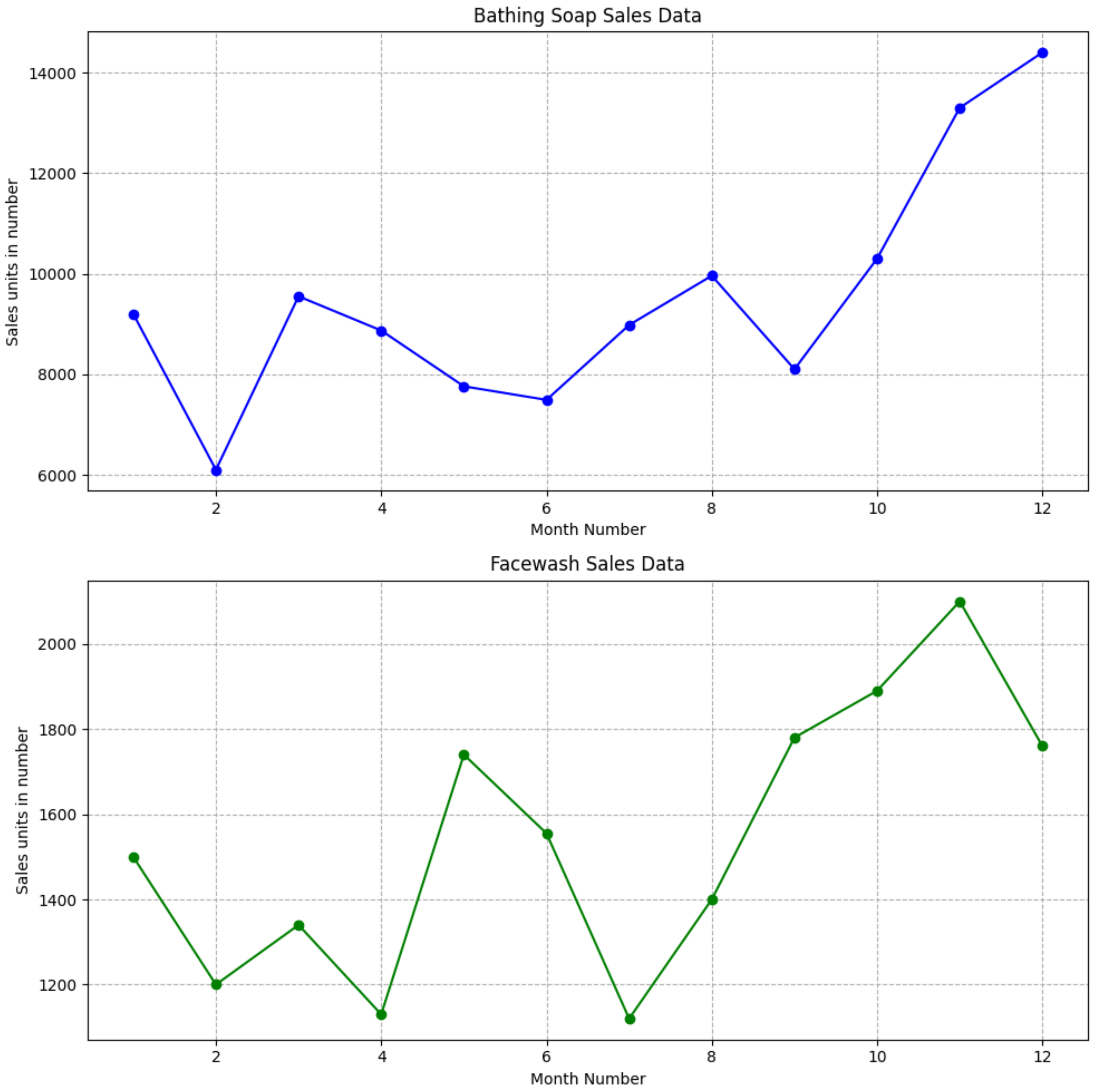
```
[313]: # Calculate total sale data for last year for each product and show it using a␣
         ↪Pie chart
       # Note: In Pie chart display Number of units sold per year for each product in␣
         ↪percentage
       import pandas as pd
       import matplotlib.pyplot as plt

       file_path = r'c:\Users\ashwi\Downloads\company_sales_data.csv'
       df = pd.read_csv(file_path)

       total_facecream_sales = df['facecream'].sum()
       total_facewash_sales = df['facewash'].sum()
       total_toothpaste_sales = df['toothpaste'].sum()
       total_bathingsoap_sales = df['bathingsoap'].sum()
       total_shampoo_sales = df['shampoo'].sum()
       total_moisturizer_sales = df['moisturizer'].sum()

       # Creating a list of total sales
       sales = [
           total_facecream_sales,
           total_facewash_sales,
           total_toothpaste_sales,
           total_bathingsoap_sales,
           total_shampoo_sales,
           total_moisturizer_sales
       ]
```

```
# Labels for each product
labels = [
    'Face Cream',
    'Face Wash',
    'Toothpaste',
    'Bathing Soap',
    'Shampoo',
    'Moisturizer'
]

# Plotting the pie chart
plt.figure(figsize=(10, 7))
plt.pie(sales, labels=labels, autopct='%1.1f%%', startangle=0)
plt.title('Total Sales Data for Last Year')
plt.show()
```



[314]:

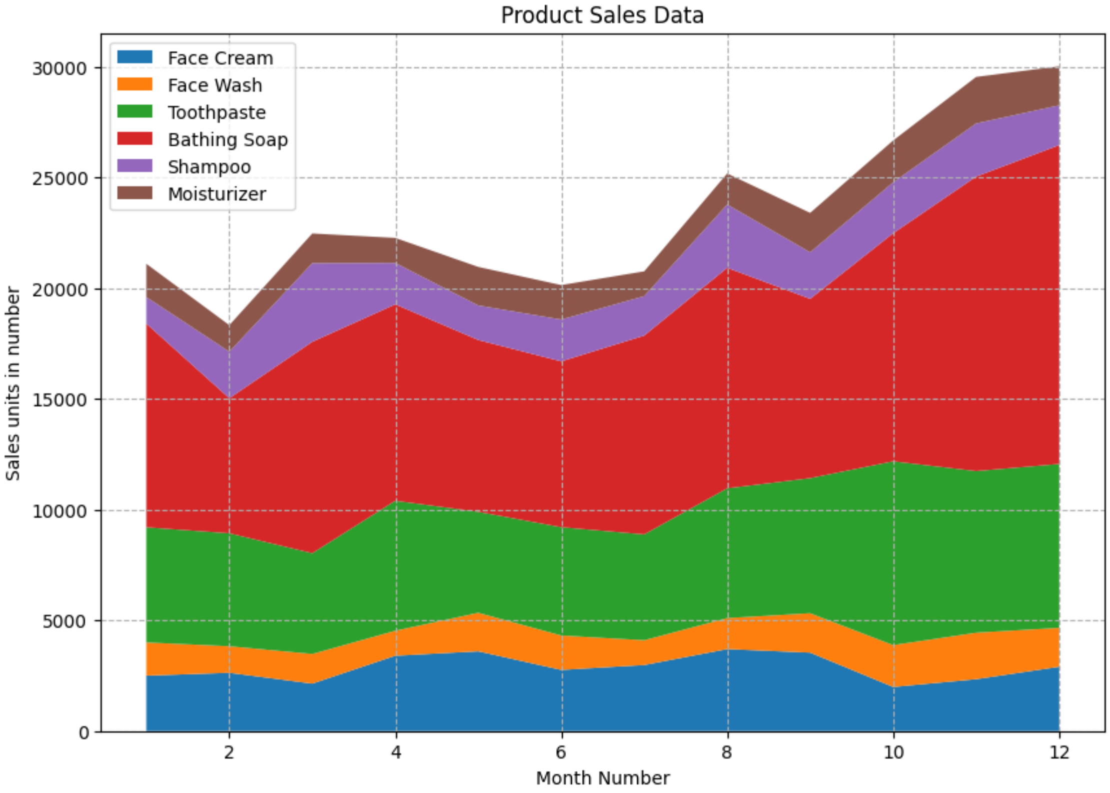
```python
# Read Bathing soap, facewash of all months and display it using the Subplot
import pandas as pd
import matplotlib.pyplot as plt

file_path = r'c:\Users\ashwi\Downloads\company_sales_data.csv'
df = pd.read_csv(file_path)

months = df['month_number']
bathingsoap_sales = df['bathingsoap']
facewash_sales = df['facewash']

fig, axs = plt.subplots(2, 1, figsize=(10, 10))

axs[0].plot(months, bathingsoap_sales, marker='o', linestyle='-', color='b')
axs[0].set_title('Bathing Soap Sales Data')
axs[0].set_xlabel('Month Number')
axs[0].set_ylabel('Sales units in number')
axs[0].grid(True, linestyle='--')

axs[1].plot(months, facewash_sales, marker='o', linestyle='-', color='g')
axs[1].set_title('Facewash Sales Data')
axs[1].set_xlabel('Month Number')
axs[1].set_ylabel('Sales units in number')
axs[1].grid(True, linestyle='--')

plt.tight_layout()
plt.show()
```



[315]:

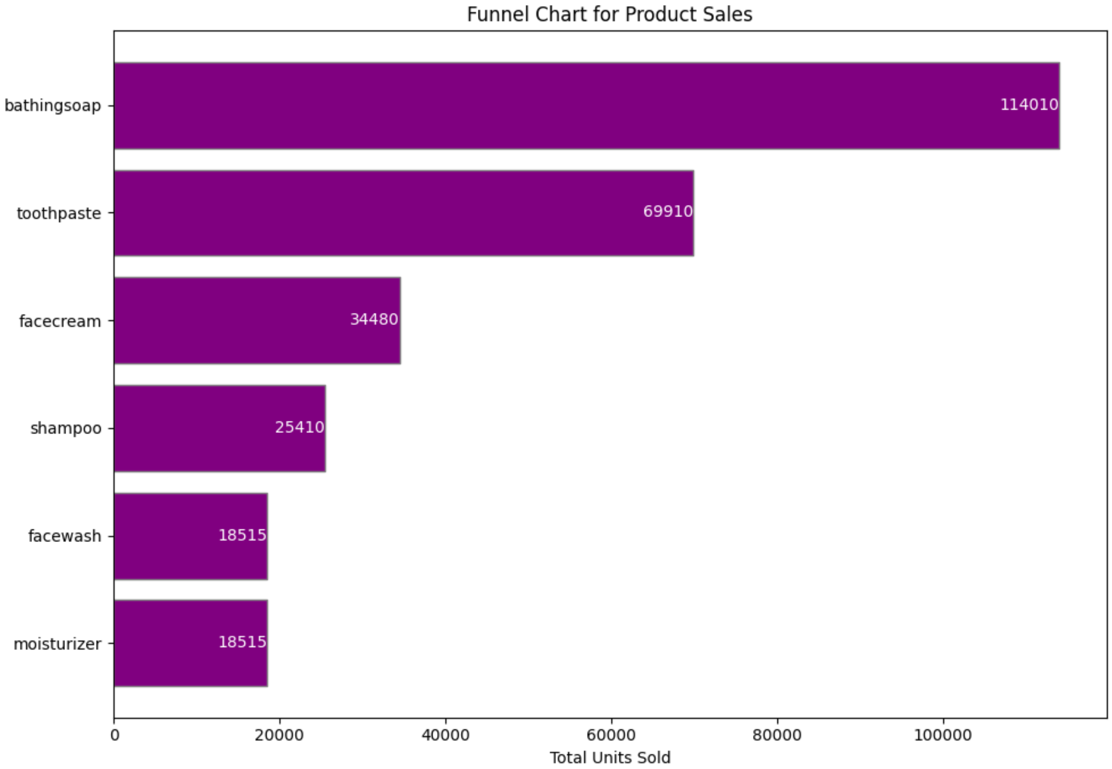
```
# Read all product sales data and show it using the stack plot
import pandas as pd
import matplotlib.pyplot as plt

file_path = r'c:\Users\ashwi\Downloads\company_sales_data.csv'
df = pd.read_csv(file_path)

months = df['month_number']
facecream_sales = df['facecream']
facewash_sales = df['facewash']
toothpaste_sales = df['toothpaste']
bathingsoap_sales = df['bathingsoap']
shampoo_sales = df['shampoo']
```

```
moisturizer_sales = df['moisturizer']

plt.figure(figsize=(10, 7))
plt.stackplot(months,
              facecream_sales, facewash_sales, toothpaste_sales,
              bathingsoap_sales, shampoo_sales, moisturizer_sales,
              labels=['Face Cream', 'Face Wash', 'Toothpaste', 'Bathing Soap',␣
 ↪'Shampoo', 'Moisturizer'])

plt.xlabel('Month Number')
plt.ylabel('Sales units in number')
plt.title('Product Sales Data')
plt.legend(loc='upper left')
plt.grid(True, linestyle='--')
plt.show()
```



[417]:
```
#
import pandas as pd
import matplotlib.pyplot as plt

file_path = r'c:\Users\ashwi\Downloads\company_sales_data.csv'
```

```
df = pd.read_csv(file_path)

products = ['facecream', 'facewash', 'toothpaste', 'bathingsoap', 'shampoo',␣
 ↪'moisturizer']
total_units = df[products].sum()

fig, ax = plt.subplots(figsize=(10, 7))

total_units_sorted = total_units.sort_values(ascending=False)
products_sorted = total_units_sorted.index

bars = ax.barh(products_sorted, total_units_sorted, color='purple',␣
 ↪edgecolor='grey')
ax.set_xlabel('Total Units Sold')
ax.set_title('Funnel Chart for Product Sales')
ax.invert_yaxis()

for bar in bars:
    width = bar.get_width()
    label_x_pos = width - max(total_units_sorted) * 0.05  # Adjust label␣
 ↪position
    ax.text(width, bar.get_y() + bar.get_height()/2, f'{width}', va='center',␣
 ↪ha='right', color='white')

plt.tight_layout()
plt.show()
```



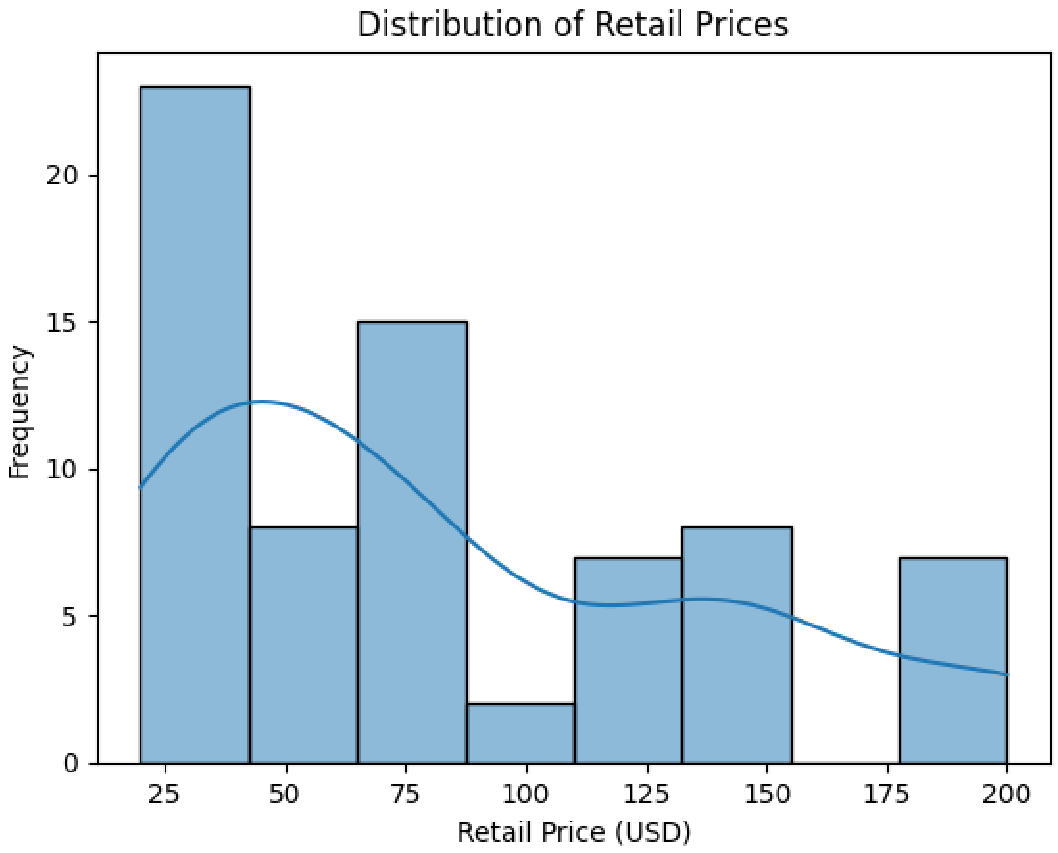
```
[426]: # PYTHON SEABORN DATASET EXERCISE
       # Read Excel Dataset and display it as a table
       import pandas as pd

       file_path = r'e:\AYUSH\Analytics books\EXCEL␣
         ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
       excel_data = pd.ExcelFile(file_path)
       df  = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)
       df.index += 1
       df
```

```
[426]:     Order_No Order_Date      Customer_Name  Ship_Date  Retail_Price(USD)  \
       1       1001 2024-01-01         John Smith 2024-01-03              49.99
       2       1002 2024-01-01           Jane Doe 2024-01-04              29.99
       3       1003 2024-01-02    Michael Johnson 2024-01-07              99.99
       4       1004 2024-01-02        Emily Brown 2024-01-03              19.99
       5       1005 2024-01-03       David Wilson 2024-01-08             149.99
       ..       …          …                  …          …                …
       66      1066 2024-02-02     Sarah Gonzalez 2024-02-06              79.99
       67      1067 2024-02-03      Matthew Smith 2024-02-06              49.99
       68      1068 2024-02-04      Emily Johnson 2024-02-05             129.99
       69      1069 2024-02-04       Daniel Brown 2024-02-08              19.99
```

```
70      1070 2024-02-04  Jennifer Hernandez 2024-02-07             149.99

    Order_Quantity  Tax(USD)  Total(USD)
1                2         0       99.98
2                1         0       29.99
3                3         0      299.97
4                4         0       79.96
5                1         0      149.99
..             …       …         …
66               2         0      159.98
67               3         0      149.97
68               1         0      129.99
69               4         0       79.96
70               1         0      149.99

[70 rows x 8 columns]
```

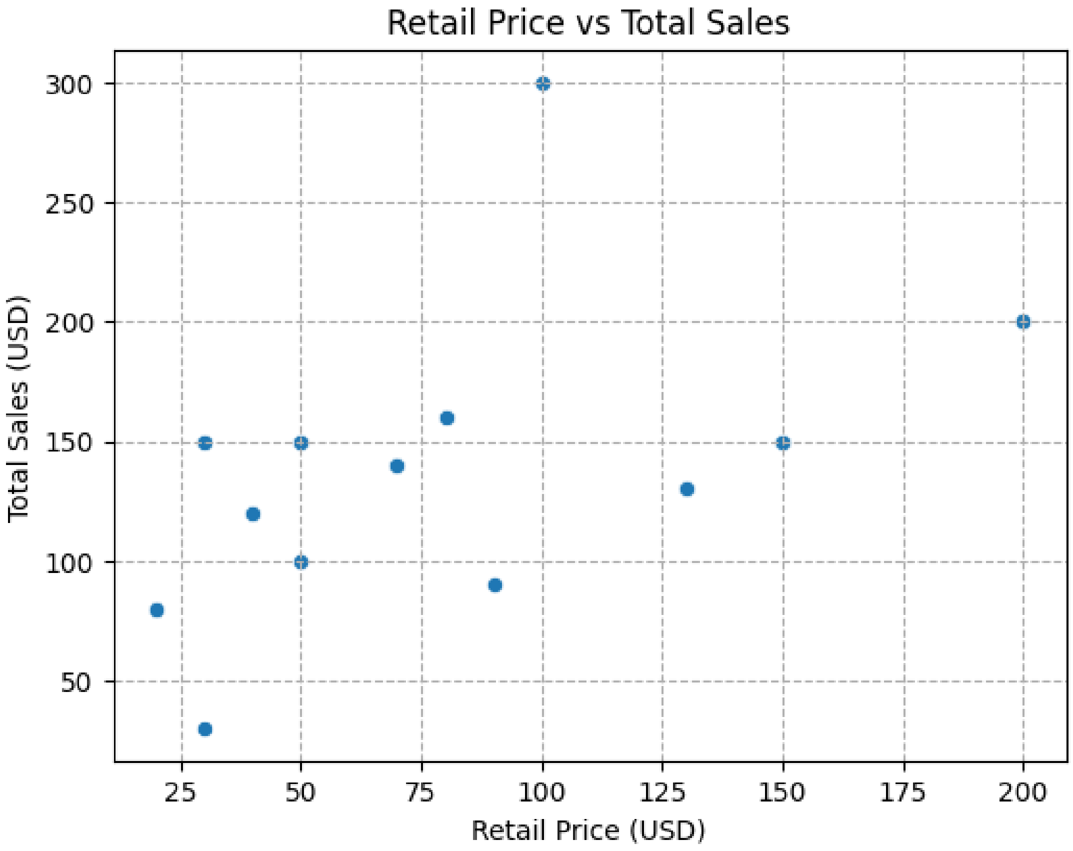
[429]:

```
# What is the distribution of retail prices ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

sns.histplot(df['Retail_Price(USD)'], kde=True)
plt.title('Distribution of Retail Prices')
plt.xlabel('Retail Price (USD)')
plt.ylabel('Frequency')
plt.show()
```



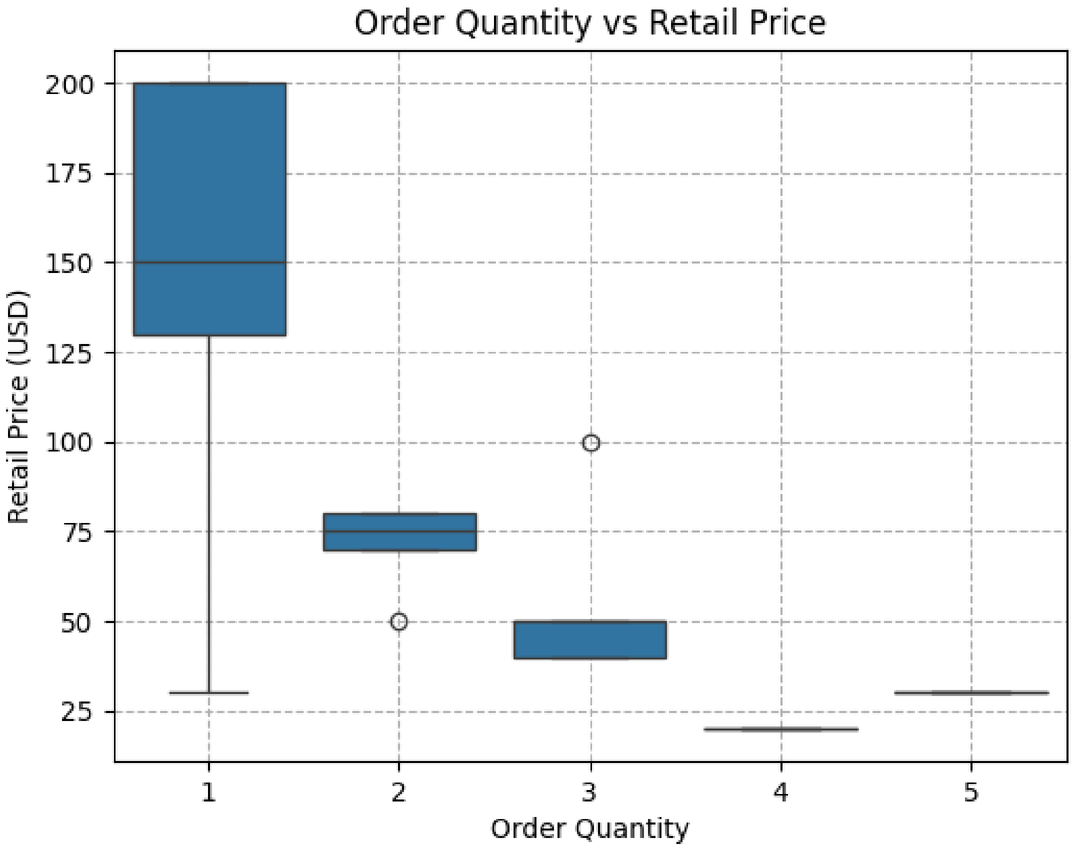
[431]:

```
# What is the relationship between retail price and total sales ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

sns.scatterplot(data=df, x='Retail_Price(USD)', y='Total(USD)')
plt.title('Retail Price vs Total Sales')
plt.xlabel('Retail Price (USD)')
plt.ylabel('Total Sales (USD)')
plt.grid(True, linestyle='--')
plt.show()
```



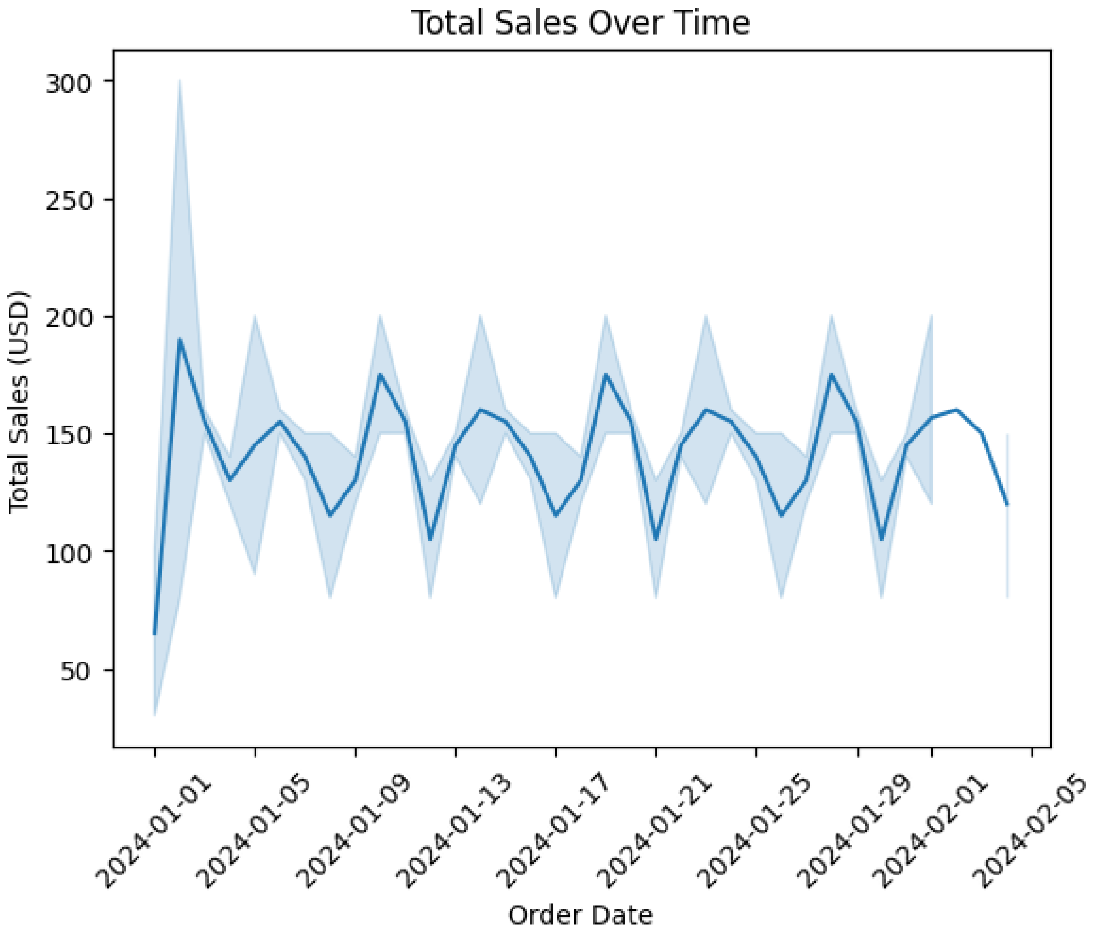
[434]:

```
# How does the order quantity vary with retail price ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

sns.boxplot(data=df, x='Order_Quantity', y='Retail_Price(USD)')
plt.title('Order Quantity vs Retail Price')
plt.xlabel('Order Quantity')
plt.ylabel('Retail Price (USD)')
plt.grid(True, linestyle='--')
plt.show()
```

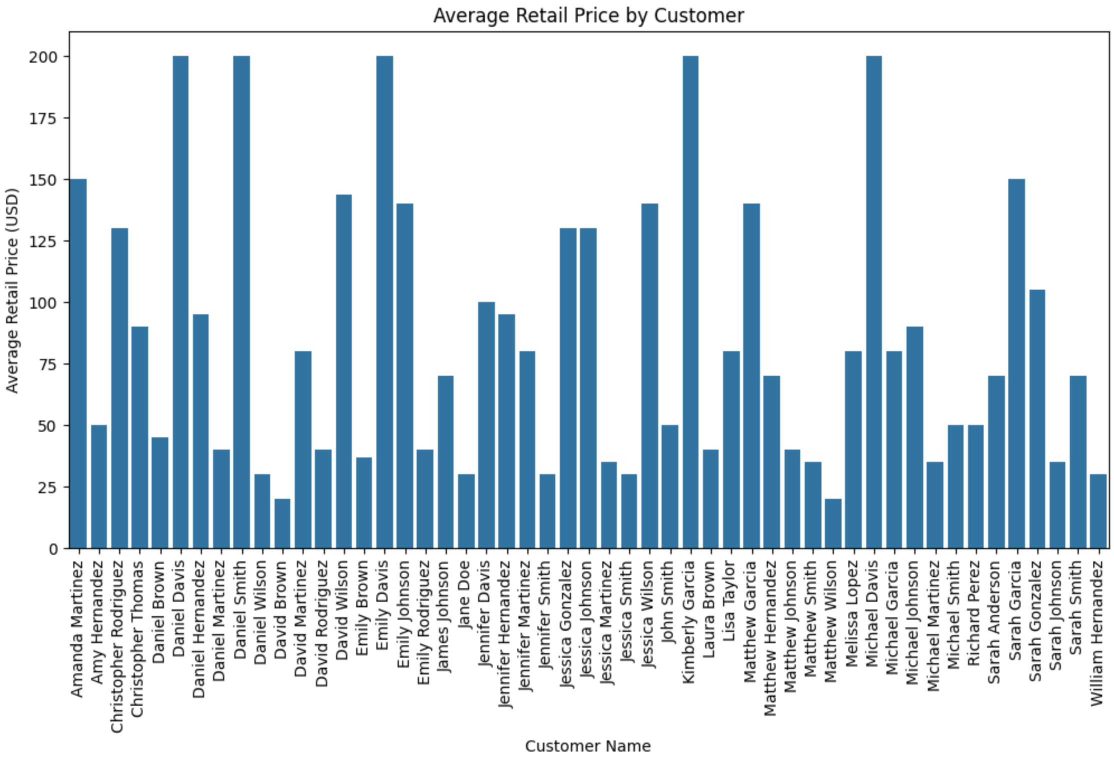


[437]:

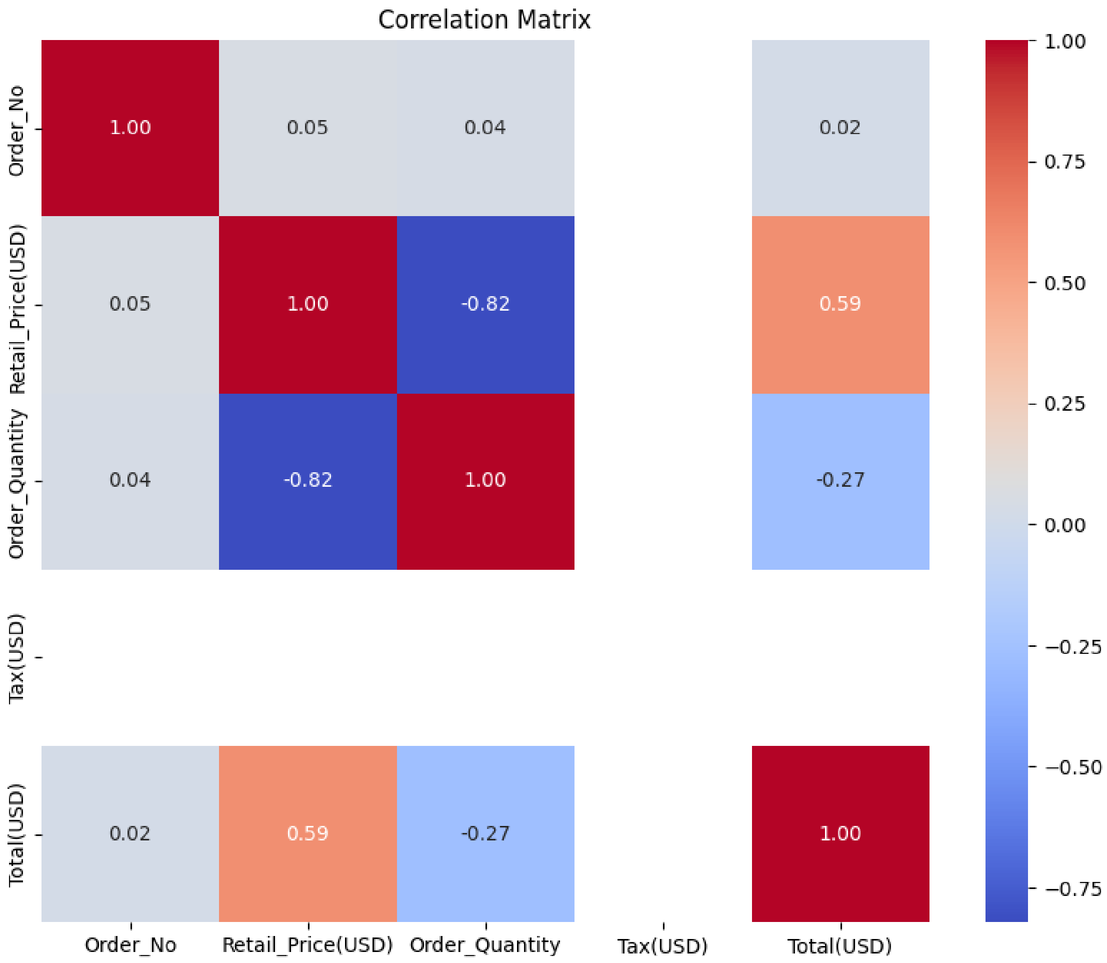
```
# What is the trend of total sales over time ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

df['Order_Date'] = pd.to_datetime(df['Order_Date'])
sns.lineplot(data=df, x='Order_Date', y='Total(USD)')
plt.title('Total Sales Over Time')
plt.xlabel('Order Date')
plt.xticks(rotation=45)
plt.ylabel('Total Sales (USD)')
plt.show()
```



[445]:
```
# What are the average retail prices per customer ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

avg_retail_price = df.groupby('Customer_Name')['Retail_Price(USD)'].mean().
 ↪reset_index()

plt.figure(figsize=(12, 6))
sns.barplot(data=avg_retail_price, x='Customer_Name', y='Retail_Price(USD)')
plt.title('Average Retail Price by Customer')
plt.xlabel('Customer Name')
plt.ylabel('Average Retail Price (USD)')
plt.xticks(rotation=90)
plt.show()
```

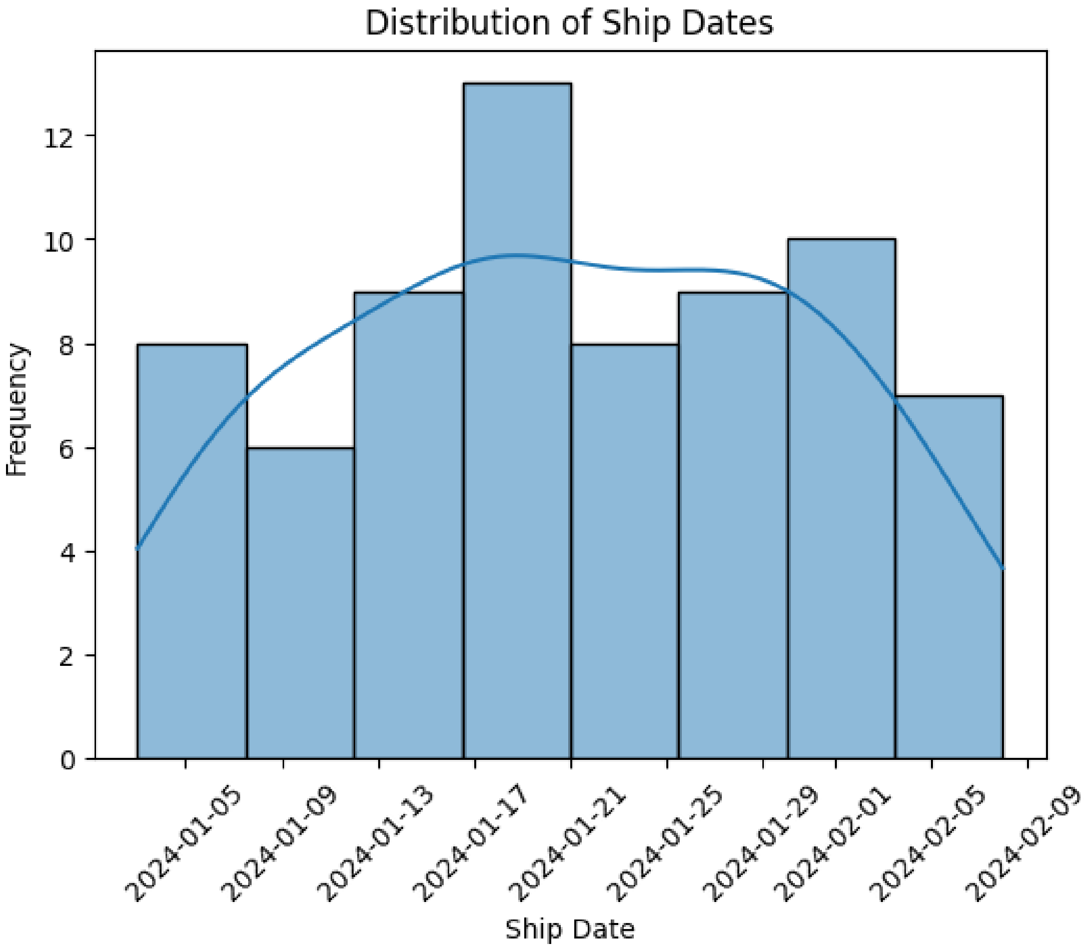


```
[447]: # What is the correlation matrix of numerical features ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

numerical_features = df.select_dtypes(include=['float64', 'int64'])
correlation_matrix = numerical_features.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

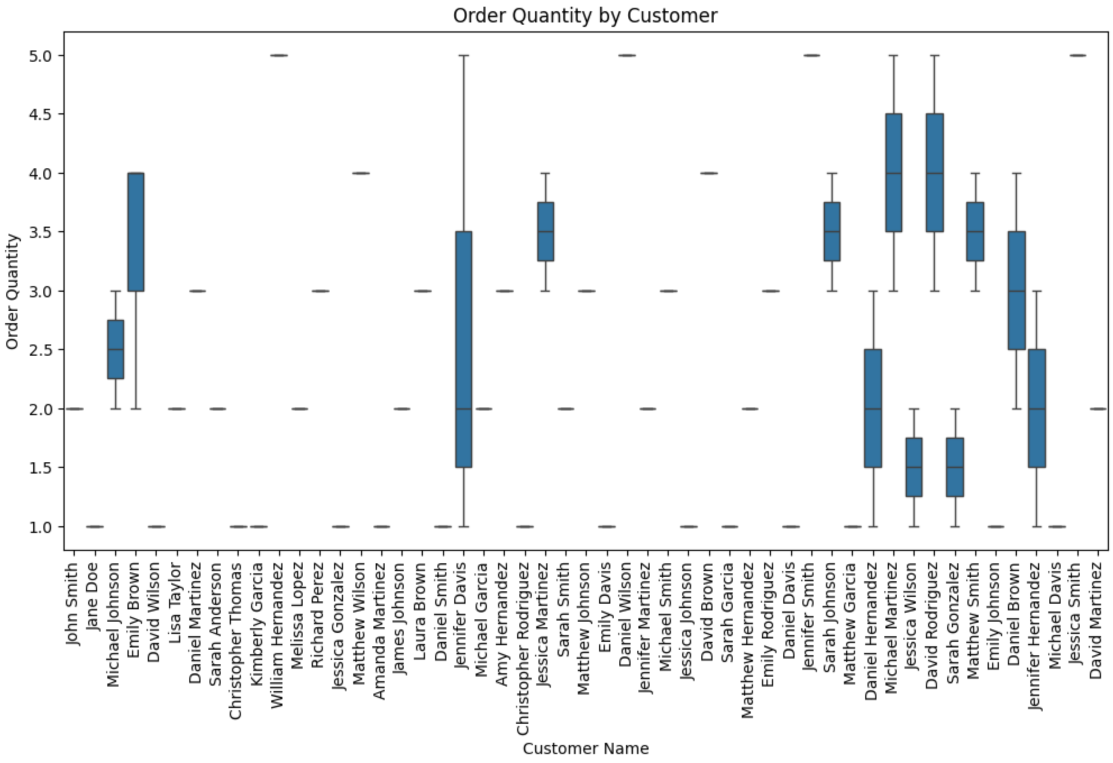


[451]:

```
# What is the distribution of ship dates ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

df['Ship_Date'] = pd.to_datetime(df['Ship_Date'])
sns.histplot(df['Ship_Date'], kde=True)
plt.title('Distribution of Ship Dates')
plt.xlabel('Ship Date')
plt.xticks(rotation=45)
plt.ylabel('Frequency')
plt.show()
```

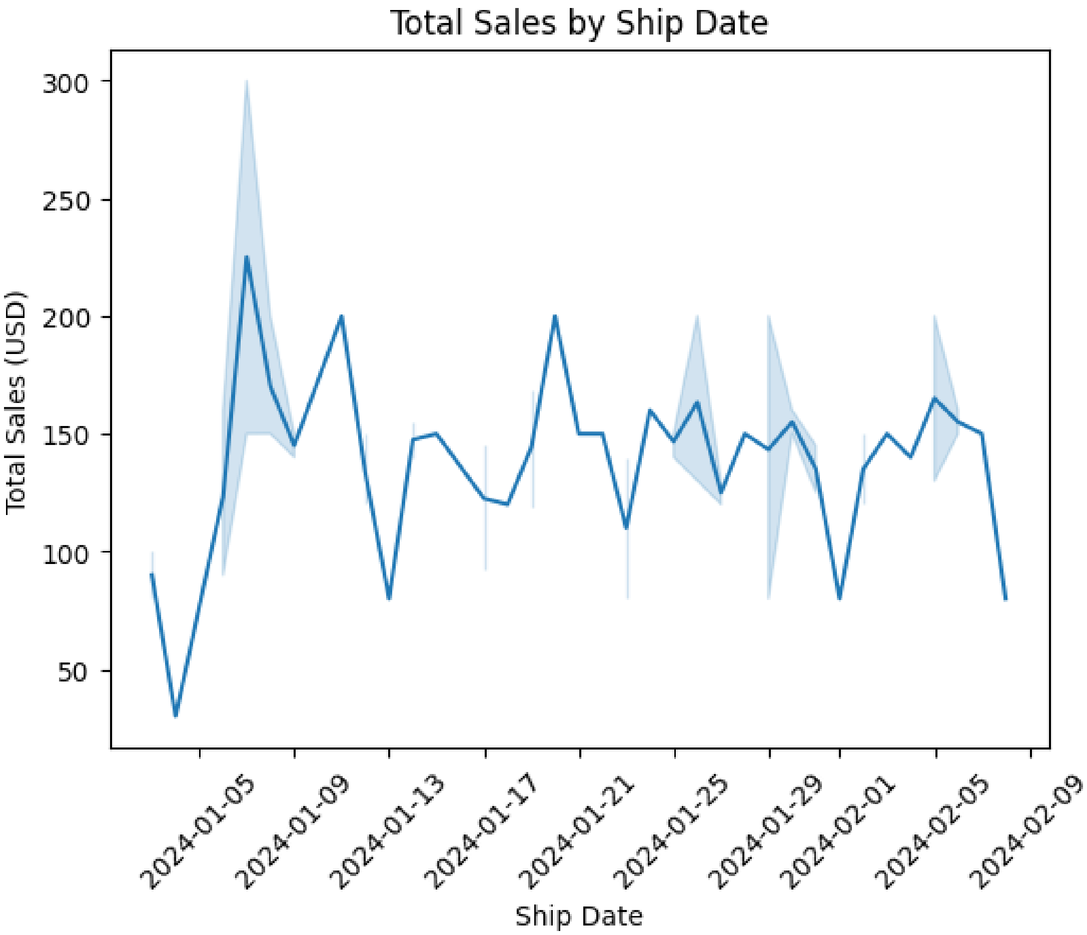


[453]:
```
# How does order quantity vary by customer ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x='Customer_Name', y='Order_Quantity')
plt.title('Order Quantity by Customer')
plt.xlabel('Customer Name')
plt.ylabel('Order Quantity')
plt.xticks(rotation=90)
plt.show()
```

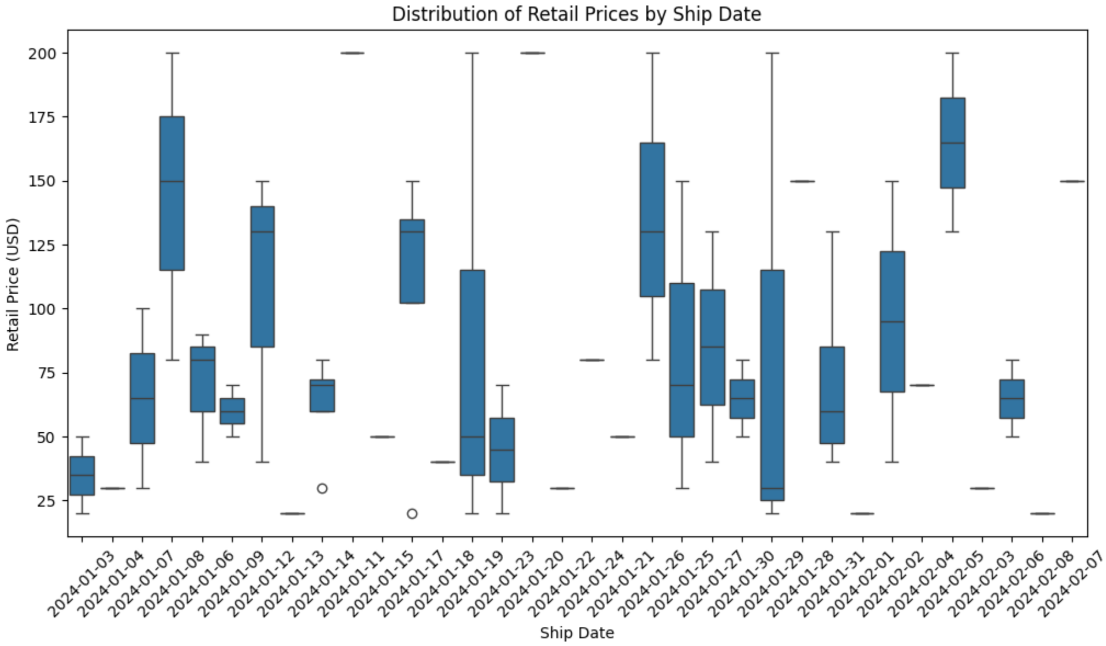


[459]:

```
# How does total sales vary with ship date ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

sns.lineplot(data=df, x='Ship_Date', y='Total(USD)')
plt.title('Total Sales by Ship Date')
plt.xlabel('Ship Date')
plt.xticks(rotation=45)
plt.ylabel('Total Sales (USD)')
plt.show()
```

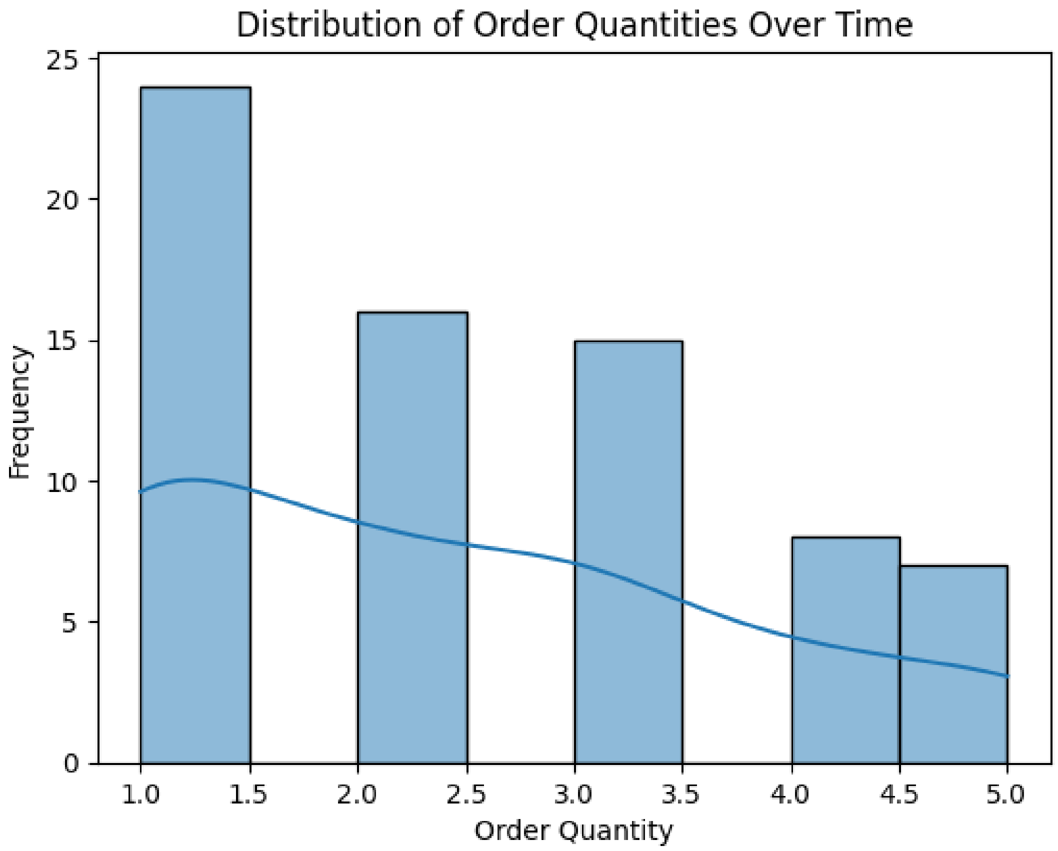


```
[460]: # What is the distribution of retail prices by ship date ?
       import pandas as pd
       import matplotlib.pyplot as plt
       import seaborn as sns

       file_path = r'e:\AYUSH\Analytics books\EXCEL␣
        ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
       excel_data = pd.ExcelFile(file_path)
       df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

       plt.figure(figsize=(12, 6))
       sns.boxplot(data=df, x='Ship_Date', y='Retail_Price(USD)')
       plt.title('Distribution of Retail Prices by Ship Date')
       plt.xlabel('Ship Date')
       plt.xticks(rotation=45)
       plt.ylabel('Retail Price (USD)')
       plt.show()
```



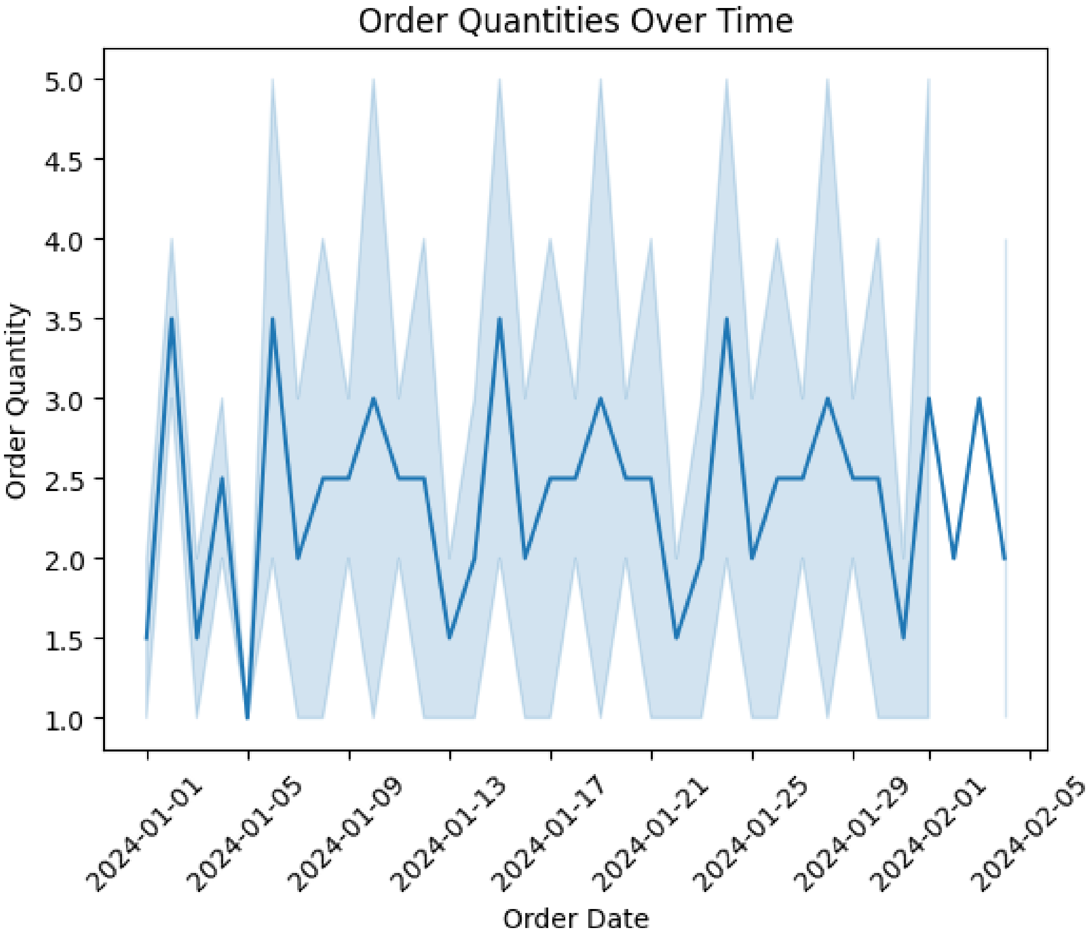
[463]:

```
# What is the distribution of order quantities over time ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

sns.histplot(df['Order_Quantity'], kde=True)
plt.title('Distribution of Order Quantities Over Time')
plt.xlabel('Order Quantity')
plt.ylabel('Frequency')
plt.show()
```

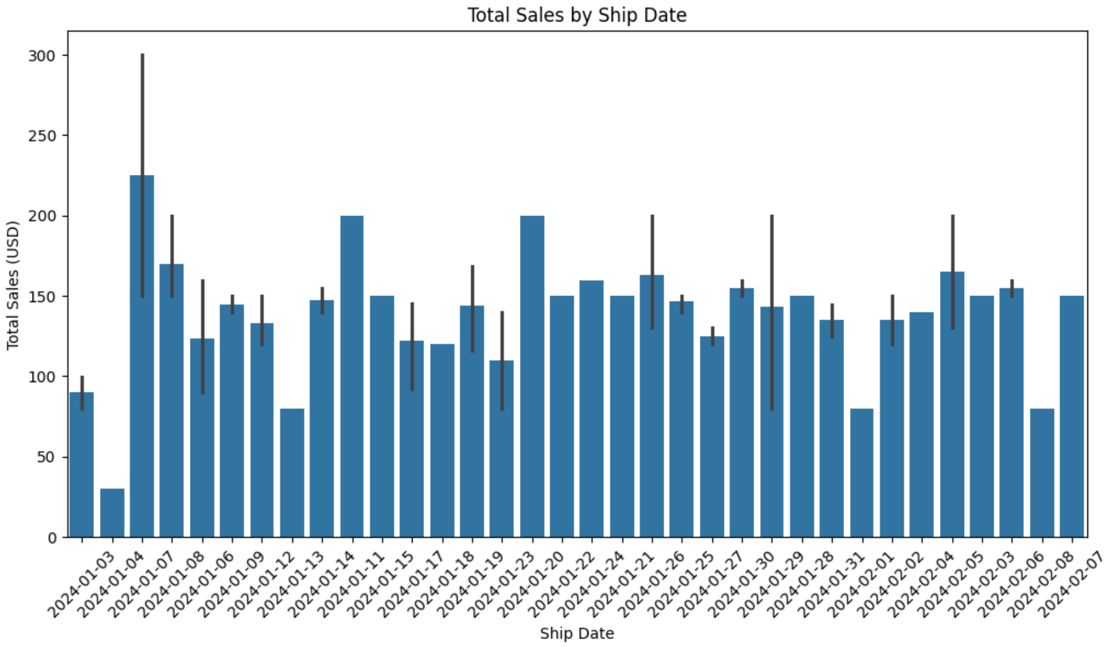


[467]:

```
# What is the trend of order quantities over time ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

sns.lineplot(data=df, x='Order_Date', y='Order_Quantity')
plt.title('Order Quantities Over Time')
plt.xlabel('Order Date')
plt.xticks(rotation=45)
plt.ylabel('Order Quantity')
plt.show()
```

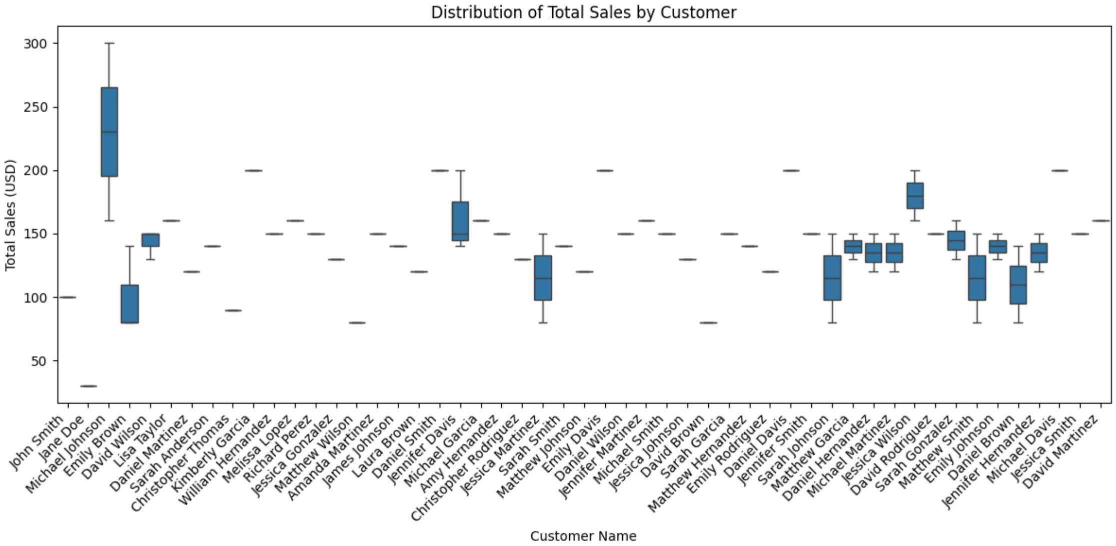


[468]:
```
# What is the total sales by ship date ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

plt.figure(figsize=(12, 6))
sns.barplot(data=df, x='Ship_Date', y='Total(USD)')
plt.title('Total Sales by Ship Date')
plt.xlabel('Ship Date')
plt.xticks(rotation=45)
plt.ylabel('Total Sales (USD)')
plt.show()
```



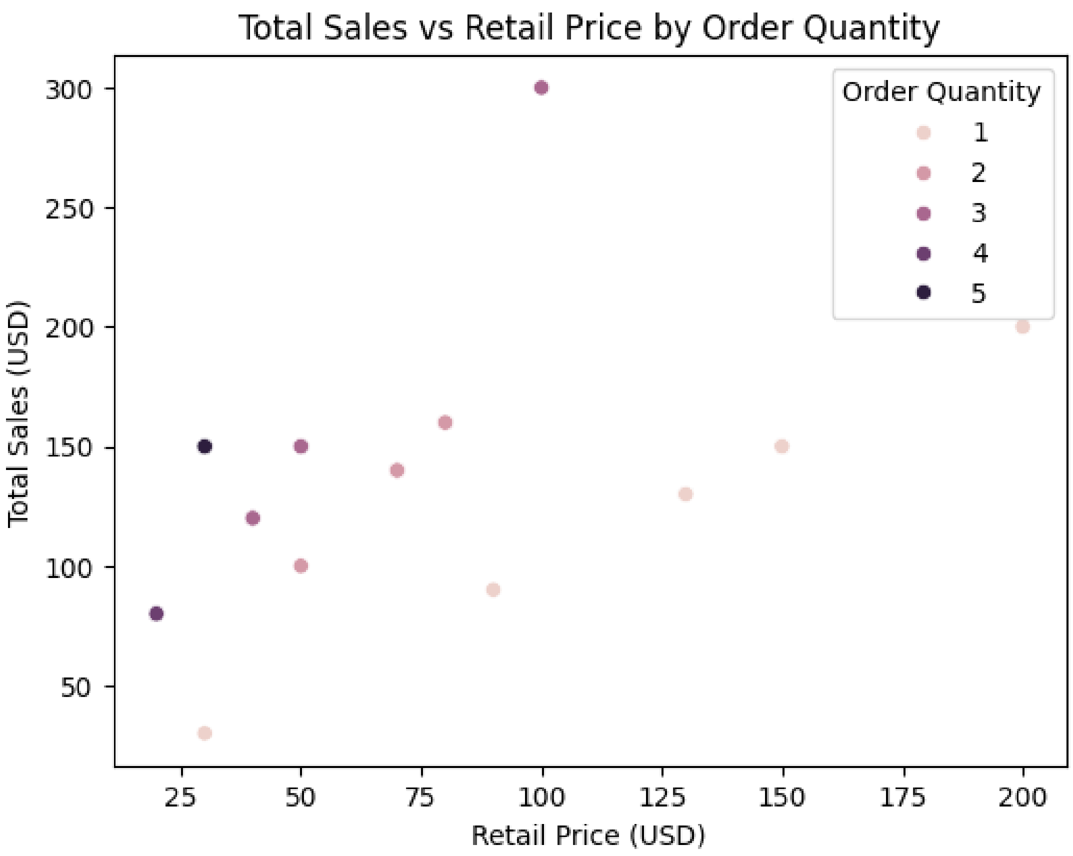
[472]:

```
# What is the distribution of total sales for each customer ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x='Customer_Name', y='Total(USD)')
plt.title('Distribution of Total Sales by Customer')
plt.xlabel('Customer Name')
plt.ylabel('Total Sales (USD)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

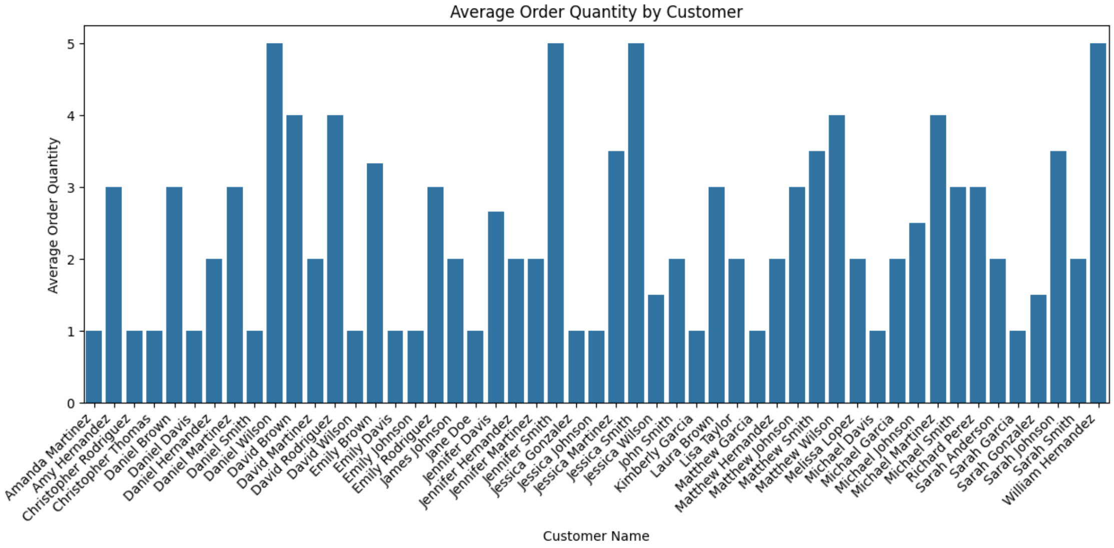


[475]:

```
# How does the total sales vary with the retail price for different order quantities ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

sns.scatterplot(data=df, x='Retail_Price(USD)', y='Total(USD)', hue='Order_Quantity')
plt.title('Total Sales vs Retail Price by Order Quantity')
plt.xlabel('Retail Price (USD)')
plt.ylabel('Total Sales (USD)')
plt.legend(title='Order Quantity')
plt.show()
```



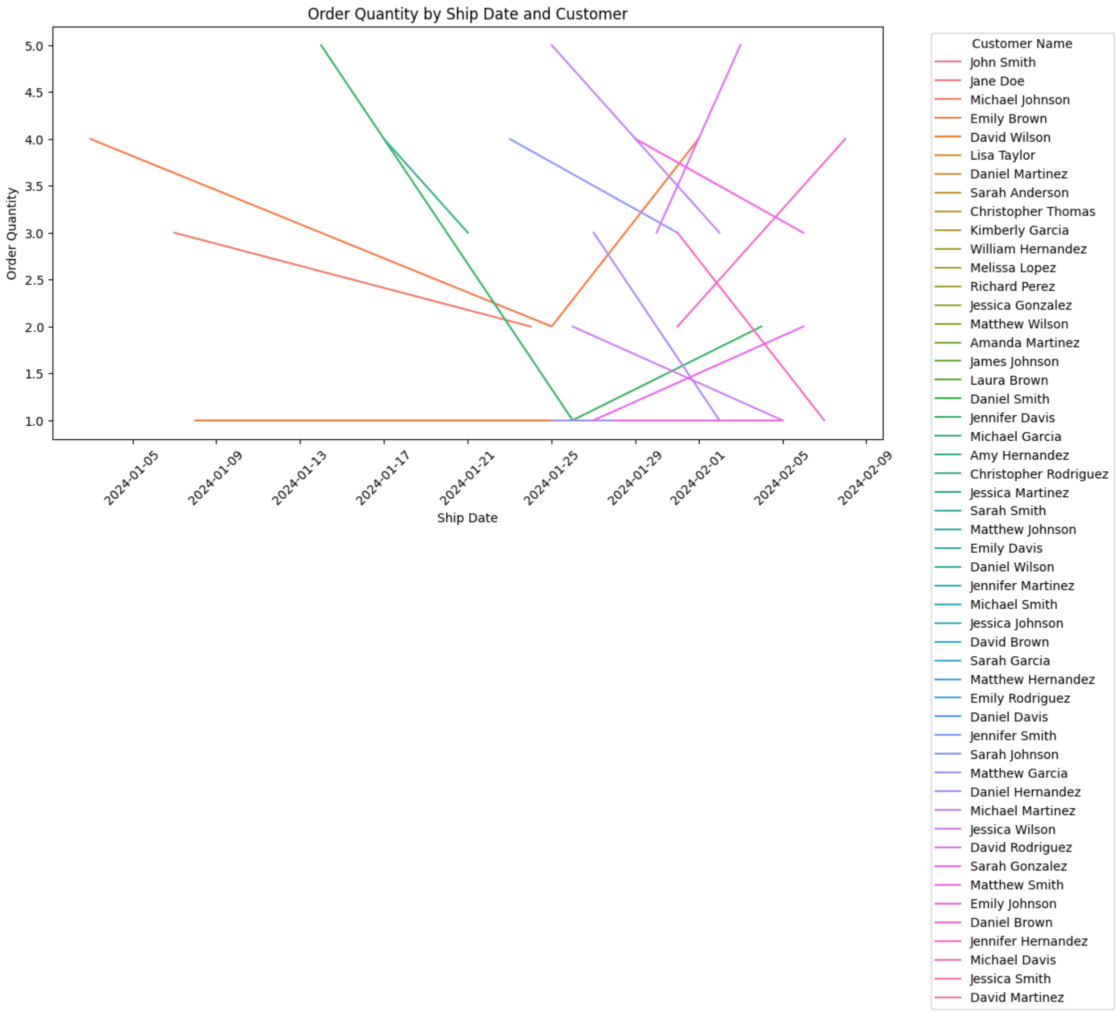
[476]:
```
# What is the average order quantity per customer ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

avg_order_quantity = df.groupby('Customer_Name')['Order_Quantity'].mean().
 ↪reset_index()

plt.figure(figsize=(12, 6))
sns.barplot(data=avg_order_quantity, x='Customer_Name', y='Order_Quantity')
plt.title('Average Order Quantity by Customer')
plt.xlabel('Customer Name')
plt.ylabel('Average Order Quantity')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



[477]:

```
# How does the order quantity vary with the ship date for different customers ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

plt.figure(figsize=(12, 6))
sns.lineplot(data=df, x='Ship_Date', y='Order_Quantity', hue='Customer_Name')
plt.title('Order Quantity by Ship Date and Customer')
plt.xlabel('Ship Date')
plt.ylabel('Order Quantity')
plt.xticks(rotation=45)
plt.legend(title='Customer Name', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

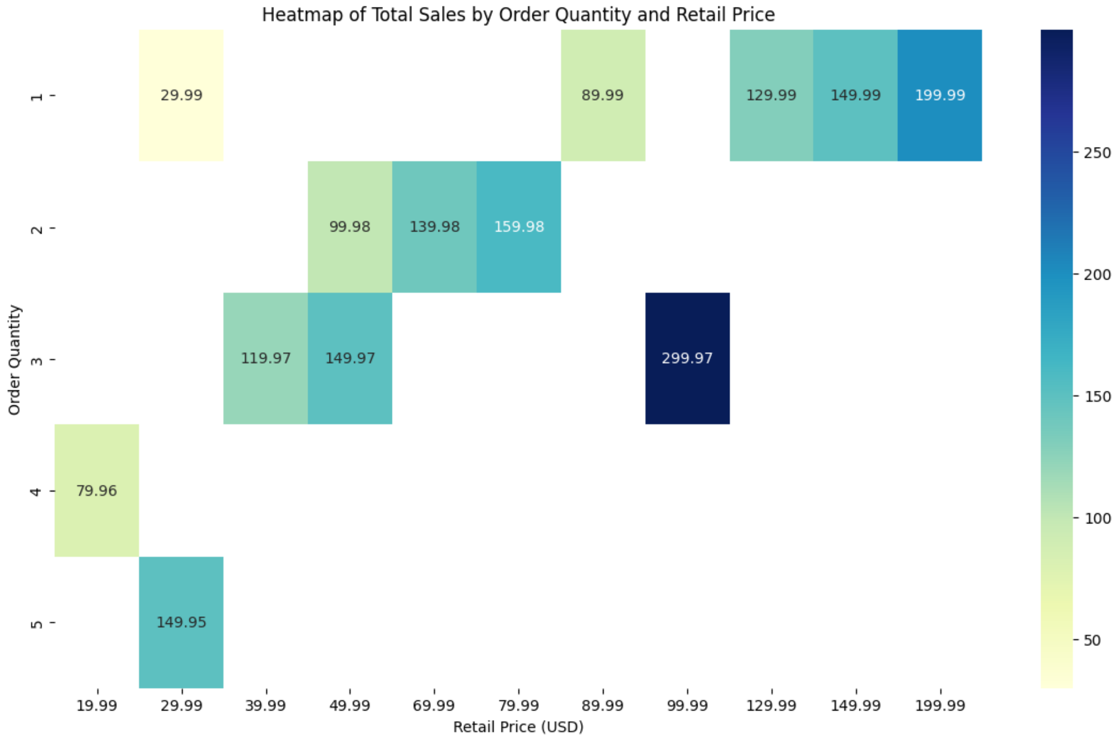
```
C:\Users\ashwi\AppData\Local\Temp\ipykernel_20008\1921627953.py:17: UserWarning:
Tight layout not applied. The bottom and top margins cannot be made large enough
to accommodate all Axes decorations.
  plt.tight_layout()
```



[478]:

```
# What is the relationship between total sales, order quantity, and retail␣
 ↪price ?
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = r'e:\AYUSH\Analytics books\EXCEL␣
 ↪Worksheets\Supermarket-Sales-Sample-Data.xlsx'
excel_data = pd.ExcelFile(file_path)
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0], header=0)

pivot_table = df.pivot_table(values='Total(USD)', index='Order_Quantity',␣
 ↪columns='Retail_Price(USD)', aggfunc='mean')

plt.figure(figsize=(14, 8))
```

```
sns.heatmap(pivot_table, cmap='YlGnBu', annot=True, fmt='.2f')
plt.title('Heatmap of Total Sales by Order Quantity and Retail Price')
plt.xlabel('Retail Price (USD)')
plt.ylabel('Order Quantity')
plt.show()
```



[357]:

```
# PYTHON RANDOM DATA GENERATION EXERCISE
# Generate 3 random integers between 100 and 999 which is divisible by 5
import random

def generate_random_integers(divisible_by, count):
    random_integers = []
    while len(random_integers) < count:
        num = random.randint(100, 999)
        if num % divisible_by == 0:
            random_integers.append(num)
    return random_integers

random_numbers = generate_random_integers(5,3)
print("3 random integers between 100 and 999 which is divisible by 5:",␣
 ↪random_numbers)
```

```
# 'len' function returns the number of items in a list, which starts counting␣
 ↪from zero
```

```
3 random integers between 100 and 999 which is divisible by 5: [625, 485, 570]
```

[347]:

```
# Random Lottery Pick. Generate 25 random lottery tickets and pick two lucky␣
 ↪tickets from it as a winner
import random

def generate_lottery_tickets(num_tickets=30):
    tickets = [random.randint(100, 999) for _ in range(num_tickets)]
    return tickets

def pick_lucky_tickets(tickets, num_winners=2):
    lucky_tickets = random.sample(tickets, num_winners)
    return lucky_tickets

lottery_tickets = generate_lottery_tickets()
lucky_tickets = pick_lucky_tickets(lottery_tickets)

print("Generated Lottery Tickets:")
print(lottery_tickets)
print("\nLucky Tickets:")
print(lucky_tickets)

# for _ in range(num_tickets) runs loop num_tickets times, adding a new random␣
 ↪integer to the list each time
```

```
Generated Lottery Tickets:
[641, 722, 158, 526, 544, 489, 450, 970, 961, 832, 451, 677, 454, 525, 307, 734,
886, 726, 272, 522, 833, 192, 229, 125, 128, 313, 259, 365, 352, 855]

Lucky Tickets:
[961, 970]
```

[352]:

```
# Generate 6 digit random secure OTP
import secrets

def generate_secure_otp(length=6):
    otp = ''.join([str(secrets.randbelow(10)) for _ in range(length)])
    return otp

secure_otp = generate_secure_otp()
print("Secure OTP:", secure_otp)

# secrets.randbelow(10) generates a random integer between 0 and 9, inclusive
```

```
Secure OTP: 622316
```

[359]:

```
# Pick a random character from a given String
import random

def pick_random_character(input_string):
    return random.choice(input_string)

user_input = input("Enter a string: ")
random_character = pick_random_character(user_input)

print("User Input String:", user_input)
print("Random Character:", random_character)

# random.choice(input_string) picks and returns a random character from the␣
 ↪input string
```

```
User Input String: Ayush, How are you ?
Random Character: w
```

[364]:

```
# Generate random String of length 5
# Note: String must be the combination of the UPPER case and lower case letters␣
 ↪only. No numbers and a special symbol
import random
import string

def generate_random_string(length=5):
    letters = string.ascii_letters
    random_string = ''.join(random.choice(letters) for _ in range(length))
    return random_string

random_string = generate_random_string()
print("Random String:", random_string)

# string.ascii_letters is a string containing all uppercase and lowercase␣
 ↪letters
```

```
Random String: WiHkd
```

[377]:

```
#  Generate a random Password which meets the following conditions :
# Password length must be 10 characters long.
# It must contain at least 2 upper case letters, 1 digit, and 1 special symbol.
import random
import string

def generate_random_password(length=10):
    if length < 4:  # Ensure minimum length for specified conditions
        raise ValueError("Password length must be at least 4 characters.")
    upper_case_letters = random.choices(string.ascii_uppercase, k=2) #␣
 ↪Selecting 2 random uppercase letters
    digits = random.choices(string.digits, k=1) # Selecting 1 random digit
    special_symbols = random.choices(string.punctuation, k=1) #  Selecting 1␣
 ↪random special symbol
    all_characters = string.ascii_letters + string.digits + string.punctuation␣
 ↪# Combine all possible characters

    remaining_length = length - len(upper_case_letters) - len(digits) -␣
 ↪len(special_symbols) # Getting remaining length to fill from all_characters
    remaining_characters = random.choices(all_characters, k=remaining_length)

    password_list = upper_case_letters + digits + special_symbols +␣
 ↪remaining_characters
    random.shuffle(password_list)  # Shuffle to ensure randomness
    password = ''.join(password_list)

    return password

random_password = generate_random_password()
print("Random Password Generation:", random_password)
```

```
Random Password Generation: X2E5~#5YU7
```

[389]:

```
# Calculate multiplication of two random float numbers :
# Note: First random float number must be between 0.1 and 1
#       Second random float number must be between 9.5 and 99.5
import random

# Generating random float numbers
first_random_float = random.uniform(0.1, 1)
second_random_float = random.uniform(9.5, 99.5)

result = first_random_float * second_random_float

print("First Random Float (0.1 to 1): {:.3f}".format(first_random_float))
print("Second Random Float (9.5 to 99.5): {:.3f}".format(second_random_float))
print("Multiplication Result: {:.3f}".format(result))

# "{:.3f}".format(first_random_float) formats first_random_float to 3 decimal␣
 ↪places
```

```
First Random Float (0.1 to 1): 0.262
Second Random Float (9.5 to 99.5): 46.104
Multiplication Result: 12.088
```

[397]:

```
# Generate random secure token of 64 bytes and random URL
import secrets

secure_token = secrets.token_hex(64) # Encoding as a hexadecimal string
print("Secure Token (64 bytes):", secure_token)
```

```
Secure Token (64 bytes): d24596a8a012f36e1dc885d863daed26522f68f7eb87272a1bacf2e
c09f2441bf8ad07e5db4800a60991424b4c17f2bc168f481c9ef0099653e0962c3daceb3c
```

[407]:

```
# Generate random secure random URL
import random
import string

def generate_random_string(length):
    letters_and_digits = string.ascii_letters + string.digits
    return ''.join(random.choices(letters_and_digits, k=length))

def generate_random_url():
    schemes = ['http', 'https'] # Protocol
    subdomains = ['www', 'app', 'api', 'blog'] # Subdomain
    domains = ['example', 'test', 'demo', 'sample'] # Domain
    tlds = ['com', 'net', 'org', 'io', 'co'] # Top-level Domains

    scheme = random.choice(schemes)
    subdomain = random.choice(subdomains)
    domain = random.choice(domains)
    tld = random.choice(tlds)
    port = random.choice(['', ':8080', ':8000', ':3000', ':5000'])

    path = '/' + generate_random_string(10)

    query_params = '?' # Initializes query_params with ?
    for _ in range(3):
        param_key = generate_random_string(5)
        param_value = generate_random_string(5)
        query_params += (f'{param_key}={param_value}&') # Appends each␣
 ↪key-value pair to query_params with = and &
    query_params = query_params.rstrip('&') # Removes the trailing &

    fragment = '#' + generate_random_string(5) # Generates a fragment (anchor)␣
 ↪with 5 random characters
    url = f'{scheme}://{subdomain}.{domain}.
 ↪{tld}{port}{path}{query_params}{fragment}'

    return url

random_url = generate_random_url()
```

```
print("Random URL:", random_url)

# path: Generates a random path of 10 characters. Uses the␣
 ↪'generate_random_string' function to create a 10-character string and␣
 ↪prepends it with '/'
# Constructs the full URL by concatenating all parts:
# scheme: Protocol (e.g., https)
# subdomain: Subdomain (e.g., www)
# domain: Domain name (e.g., example)
# tld: Top-level domain (e.g., com)
# port: Optional port (e.g., :8080)
# path: Path (e.g., /aBcDeFgHiJ)
# query_params: Query parameters (e.g., ?abcde=fghij&klmno=pqrst&uvwxy=zzzzz)
# fragment: Fragment (e.g., #abcde)
```

```
Random URL:
https://www.demo.net:5000/PGP18shjqw?DXsCc=8LTap&VEXeU=6dmGb&4l5vc=U3BjP#7kbE9
```

[412]:

```
# Roll dice in such a way that every time you get the same number for each turn
import random

fixed_roll = random.randint(1, 6)
for _ in range(5):
    print("Dice roll:", fixed_roll)
```

```
Dice roll: 6
Dice roll: 6
Dice roll: 6
Dice roll: 6
Dice roll: 6
```

[415]:

```
# Generate a random date between given start and end dates
import random
from datetime import datetime, timedelta

def generate_random_date(start_date, end_date):
    start_dt = datetime.strptime(start_date, "%Y-%m-%d")
    end_dt = datetime.strptime(end_date, "%Y-%m-%d")

    delta = end_dt - start_dt
    delta_seconds = delta.total_seconds()

    random_seconds = random.uniform(0, delta_seconds)
    random_date = start_dt + timedelta(seconds= random_seconds)

    return random_date.date()
```

```
start_date = input("Enter the start date (YYYY-MM-DD): ")
end_date = input("Enter the end date (YYYY-MM-DD): ")

print("Start Date:", start_date)
print("End Date:", end_date)

random_date = generate_random_date(start_date, end_date)
print("Random Date:", random_date)
```

```
Start Date: 2024-07-15
End Date: 2024-06-15
Random Date: 2024-06-20
```