Advanced programming lab evaluation-2

Week-5 lab

**Exercise 1: Student Grades Management System**

Create a simple student grades management system which perform the following functions

(Use a dictionary where the keys are student names, and the values are lists of grades.):

• **Add a student:** Add a student's name and their grades for multiple subjects.

• **Update a Grade:** Update a specific grade for a student in each subject.

• **Remove a student:** Remove a student from the system.

• **Get Average Grade:** Calculate and return the average grade for a student across all

subjects.

• **Get Subject Average:** Calculate and return the average grade for a specific subject.

• **List All Students:** List all students with their average grades for each subject and

overall.

• **Get Highest Grade:** Find the highest grade in a specific subject.

Solution

class StudentGrades:

def \_\_init\_\_(self):

self.students = {}

def add\_student(self, name, grades):

self.students[name] = grades

def update\_grade(self, name, subject, grade):

if name in self.students:

self.students[name][subject] = grade

else:

print(f"Student {name} not found.")

def remove\_student(self, name):

if name in self.students:

del self.students[name]

else:

print(f"Student {name} not found.")

def get\_average\_grade(self, name):

if name in self.students:

grades = self.students[name].values()

return sum(grades) / len(grades)

else:

print(f"Student {name} not found.")

return None

def get\_subject\_average(self, subject):

total, count = 0, 0

for grades in self.students.values():

if subject in grades:

total += grades[subject]

count += 1

if count == 0:

return None

return total / count

def list\_all\_students(self):

for student, grades in self.students.items():

avg\_grade = self.get\_average\_grade(student)

print(f"{student}: Grades {grades}, Average {avg\_grade:.2f}")

def get\_highest\_grade(self, subject):

highest\_grade = None

top\_student = None

for student, grades in self.students.items():

if subject in grades and (highest\_grade is None or grades[subject] > highest\_grade):

highest\_grade = grades[subject]

top\_student = student

if highest\_grade is not None:

return top\_student, highest\_grade

else:

return None, None

# Example usage:

system = StudentGrades()

system.add\_student("Alice", {"Math": 90, "Science": 85})

system.add\_student("Bob", {"Math": 92, "Science": 88})

system.update\_grade("Alice", "Math", 95)

system.list\_all\_students()

print("Alice's Average:", system.get\_average\_grade("Alice"))

print("Math Subject Average:", system.get\_subject\_average("Math"))

print("Highest Grade in Science:", system.get\_highest\_grade("Science"))

output

Alice: Grades {'Math': 95, 'Science': 85}, Average 90.00

Bob: Grades {'Math': 92, 'Science': 88}, Average 90.00

Alice's Average: 90.0

Math Subject Average: 93.5

Highest Grade in Science: ('Bob', 88)

**Exercise 2: Employee Management System**

Implement Employee Management System using nested dictionaries and lists and

implement following functions to handle different operations.

• **add\_employee():** Adds a new employee or updates an existing employee's details.

• **update\_salary():** Updates the salary of an existing employee.

• **add\_performance\_score():** Adds a performance score to an employee's record.

• **remove\_employee():** Removes an employee from the records.

• **get\_average\_salary\_by\_department():** Computes the average salary of employees

in a specified department.

• **get\_employee\_with\_highest\_performance():** Finds the employee with the highest

average performance score.

• **list\_employees\_by\_department():** Lists all employees in a specified department.

Solution

class EmployeeManagementSystem:

def \_\_init\_\_(self):

self.employees = {}

def add\_employee(self, emp\_id, name, department, salary, performance\_scores=None):

if performance\_scores is None:

performance\_scores = []

self.employees[emp\_id] = {

"name": name,

"department": department,

"salary": salary,

"performance\_scores": performance\_scores

}

def update\_salary(self, emp\_id, new\_salary):

if emp\_id in self.employees:

self.employees[emp\_id]['salary'] = new\_salary

else:

print(f"Employee with ID {emp\_id} not found.")

def add\_performance\_score(self, emp\_id, score):

if emp\_id in self.employees:

self.employees[emp\_id]['performance\_scores'].append(score)

else:

print(f"Employee with ID {emp\_id} not found.")

def remove\_employee(self, emp\_id):

if emp\_id in self.employees:

del self.employees[emp\_id]

else:

print(f"Employee with ID {emp\_id} not found.")

def get\_average\_salary\_by\_department(self, department):

total\_salary, count = 0, 0

for emp in self.employees.values():

if emp['department'] == department:

total\_salary += emp['salary']

count += 1

if count == 0:

print(f"No employees found in the {department} department.")

return None

return total\_salary / count

def get\_employee\_with\_highest\_performance(self):

highest\_avg\_score = None

top\_employee = None

for emp\_id, emp in self.employees.items():

if emp['performance\_scores']:

avg\_score = sum(emp['performance\_scores']) / len(emp['performance\_scores'])

if highest\_avg\_score is None or avg\_score > highest\_avg\_score:

highest\_avg\_score = avg\_score

top\_employee = emp\_id

if top\_employee is not None:

return self.employees[top\_employee]['name'], highest\_avg\_score

else:

print("No performance scores found for any employees.")

return None

def list\_employees\_by\_department(self, department):

employees\_in\_dept = [emp['name'] for emp in self.employees.values() if emp['department'] == department]

if employees\_in\_dept:

return employees\_in\_dept

else:

print(f"No employees found in the {department} department.")

return []

ems = EmployeeManagementSystem()

ems.add\_employee(1, "Alice", "Sales", 50000)

ems.add\_employee(2, "Bob", "Sales", 55000)

ems.add\_employee(3, "Charlie", "Engineering", 60000)

ems.add\_employee(4, "David", "Engineering", 62000)

ems.update\_salary(2, 58000)

ems.add\_performance\_score(1, 90)

ems.add\_performance\_score(1, 85)

ems.add\_performance\_score(2, 88)

ems.add\_performance\_score(3, 92)

ems.add\_performance\_score(4, 75)

ems.remove\_employee(4)

average\_salary\_sales = ems.get\_average\_salary\_by\_department("Sales")

average\_salary\_engineering = ems.get\_average\_salary\_by\_department("Engineering")

top\_performer = ems.get\_employee\_with\_highest\_performance()

employees\_in\_sales = ems.list\_employees\_by\_department("Sales")

employees\_in\_engineering = ems.list\_employees\_by\_department("Engineering")

average\_salary\_sales, average\_salary\_engineering, top\_performer, employees\_in\_sales, employees\_in\_engineering

output

**Average salary in Sales department**: $54,000

**Average salary in Engineering department**: $60,000

**Top performer**: Charlie with an average performance score of 92.0

**Employees in Sales department**: Alice, Bob

**Employees in Engineering department**: Charlie

Week-6 lab

**Exercise 1:**

You are tasked with designing a **Library Management System** for a local library. The

library has both **EBooks** (digital format) and **Printed Books** (physical copies), and the

system should allow members (both **students** and **teachers**) to borrow books. The library

also has a **librarian** who manages the addition and removal of books.

Your system should include the following features:

1. **Book Management**:

o Books can be either **EBooks** or **Printed Books**.

o Each book should have a **title**, **author**, and **ISBN**.

o EBooks should have a **file format**, while Printed Books should have a **page**

**count**.

2. **Member Management**:

o The library has **members** who can either be **students** or **teachers**.

o Each member has a **name** and **member ID**.

o Members should be able to **borrow books**.

3. **Librarian Management**:

o A **librarian** can add or remove books from the library.

o A librarian can be both a **student** and a **teacher**.

4. **Library Operations**:

o The system should allow the librarian to:

▪ **Add new books** to the library.

▪ **Remove books** from the library using their ISBN.

▪ **Search for books** by title or author.

**Requirements:**

Using Python and Object-Oriented Programming principles, implement the following:

1. Create a class hierarchy to represent **books** (including EBooks and Printed Books).

2. Create a class hierarchy to represent **members** (students and teachers).

3. Implement the functionalities for adding, removing, and searching for books.

4. Demonstrate the following types of inheritance:

o **Single Inheritance** for books.

o **Multiple Inheritance** for the librarian, who is both a student and a teacher.

o **Hierarchical Inheritance** for members (students and teachers).

**Tasks:**

1. **Book Management**:

o Define a base class Book with attributes for title, author, and ISBN.

o Define a subclass EBook that adds the attribute for file format.

o Define another subclass PrintedBook that adds the attribute for page count.

2. **Member and Librarian Management**:

o Define a base class Member with attributes for name and member ID.

o Define two subclasses: Student and Teacher, which inherit from Member.

o Create a Librarian class that inherits from both Student and Teacher (multiple

inheritance).

3. **Library Class**:o Implement a Library class to manage the collection of books.

o Add methods to the Library class to:

▪ Add new books.

▪ Remove a book by its ISBN.

▪ Search for books by title or author.

4. **Demonstration**:

o Instantiate a library and add books (both EBooks and Printed Books) to it.

o Demonstrate searching for books using keywords.

o Show how a librarian can add and remove books from the system.

Solution

class Book:

def \_\_init\_\_(self, title, author, isbn):

self.title = title

self.author = author

self.isbn = isbn

class EBook(Book):

def \_\_init\_\_(self, title, author, isbn, file\_format):

super().\_\_init\_\_(title, author, isbn)

self.file\_format = file\_format

def \_\_str\_\_(self):

return f"EBook: {self.title}, Author: {self.author}, ISBN: {self.isbn}, Format: {self.file\_format}"

class PrintedBook(Book):

def \_\_init\_\_(self, title, author, isbn, page\_count):

super().\_\_init\_\_(title, author, isbn)

self.page\_count = page\_count

def \_\_str\_\_(self):

return f"Printed Book: {self.title}, Author: {self.author}, ISBN: {self.isbn}, Pages: {self.page\_count}"

class Member:

def \_\_init\_\_(self, name, member\_id):

self.name = name

self.member\_id = member\_id

self.borrowed\_books = []

def borrow\_book(self, book):

self.borrowed\_books.append(book)

print(f"{self.name} borrowed '{book.title}'.")

class Student(Member):

def \_\_init\_\_(self, name, member\_id):

super().\_\_init\_\_(name, member\_id)

class Teacher(Member):

def \_\_init\_\_(self, name, member\_id):

super().\_\_init\_\_(name, member\_id)

class Librarian(Student, Teacher):

def \_\_init\_\_(self, name, member\_id):

super().\_\_init\_\_(name, member\_id)

class Library:

def \_\_init\_\_(self):

self.books = {}

def add\_book(self, book):

self.books[book.isbn] = book

print(f"Book '{book.title}' added to the library.")

def remove\_book(self, isbn):

if isbn in self.books:

removed\_book = self.books.pop(isbn)

print(f"Book '{removed\_book.title}' removed from the library.")

else:

print(f"No book with ISBN {isbn} found in the library.")

def search\_by\_title(self, title):

found\_books = [book for book in self.books.values() if title.lower() in book.title.lower()]

if found\_books:

print(f"Books found with title '{title}':")

for book in found\_books:

print(book)

else:

print(f"No books found with the title '{title}'.")

def search\_by\_author(self, author):

found\_books = [book for book in self.books.values() if author.lower() in book.author.lower()]

if found\_books:

print(f"Books found by author '{author}':")

for book in found\_books:

print(book)

else:

print(f"No books found by the author '{author}'.")

if \_\_name\_\_ == "\_\_main\_\_":

library = Library()

ebook1 = EBook("Python Programming", "John Doe", "1111", "PDF")

ebook2 = EBook("Data Science Handbook", "Jane Smith", "2222", "EPUB")

printed\_book1 = PrintedBook("Learning Python", "Mark Lutz", "3333", 1600)

printed\_book2 = PrintedBook("Introduction to Algorithms", "Cormen", "4444", 1312)

library.add\_book(ebook1)

library.add\_book(ebook2)

library.add\_book(printed\_book1)

library.add\_book(printed\_book2)

library.search\_by\_title("Python")

library.search\_by\_author("Cormen")

library.remove\_book("2222")

student = Student("Alice", "S123")

teacher = Teacher("Bob", "T456")

librarian = Librarian("Clara", "L789")

student.borrow\_book(printed\_book1)

teacher.borrow\_book(ebook1)

librarian.borrow\_book(ebook2)

library.remove\_book("1111")

output

Book 'Python Programming' added to the library.

Book 'Data Science Handbook' added to the library.

Book 'Learning Python' added to the library.

Book 'Introduction to Algorithms' added to the library.

Books found with title 'Python':

EBook: Python Programming, Author: John Doe, ISBN: 1111, Format: PDF

Printed Book: Learning Python, Author: Mark Lutz, ISBN: 3333, Pages: 1600

Books found by author 'Cormen':

Printed Book: Introduction to Algorithms, Author: Cormen, ISBN: 4444, Pages: 1312

Book 'Data Science Handbook' removed from the library.

Alice borrowed 'Learning Python'.

Bob borrowed 'Python Programming'.

Clara borrowed 'Data Science Handbook'.

Book 'Python Programming' removed from the library.

**Exercise 2: Advanced E-Commerce System Utilizing Polymorphism**

A rapidly growing online retail company is looking to upgrade its **E-Commerce System**.

They need the system to manage various types of products, allow users to add them to their

shopping carts, apply discounts, and handle different payment methods. To make the system

robust, flexible, and scalable, you decide to use **Object-Oriented Programming (OOP)**

principles.

**Objectives:**

1. **Product Management**:

o Products in the system belong to different categories such as **Electronics** and

**Clothing**. Each product category has its own discount logic.

o Discounts should be applied based on product types, demonstrating **method**

**overriding**.

2. **Shopping Cart Management**:

o Users should be able to add multiple items to their shopping carts and see the

total cost after discounts.

o The system should allow **merging two shopping carts** using operator

overloading.

3. **Payment Processing**:

o Customers should be able to process payments using various methods (e.g.,

credit card, PayPal). Even though Python does not natively support **method**

**overloading**, it should be simulated to handle different payment methods

efficiently.

**Functional Requirements:**

1. **Products**:

o Implement a base Product class that represents general products, containing

attributes like **name** and **price**.

o Create derived classes such as Electronics and Clothing that **override** the base

class method for calculating product discounts.

2. **Shopping Cart**:

o Implement a ShoppingCart class that can hold a collection of products.

o Overload the + operator to merge two shopping carts into one.

3. **Payment Processing**:

o Implement a PaymentProcessor class that simulates **method overloading** to

handle different payment methods (e.g., credit card and PayPal) using variable

arguments

solution

from datetime import datetime

class Product:

    def \_\_init\_\_(self, name, price, quantity, sku):

        self.name = name

        self.price = price

        self.quantity = quantity

        self.sku = sku

    def update\_stock(self, quantity):

        self.quantity += quantity

    def is\_low\_stock(self, threshold=5):

        return self.quantity < threshold

    def apply\_discount(self, discount\_percentage):

        self.price -= self.price \* (discount\_percentage / 100)

    def inventory\_value(self):

        return self.price \* self.quantity

    def \_\_str\_\_(self):

        return f"{self.name} (SKU: {self.sku}) - Price: {self.price:.2f}, Quantity: {self.quantity}"

class Electronics(Product):

    def \_\_init\_\_(self, name, price, quantity, sku, warranty\_period, brand):

        super().\_\_init\_\_(name, price, quantity, sku)

        self.warranty\_period = warranty\_period

        self.brand = brand

    def \_\_str\_\_(self):

        return f"Electronics: {super().\_\_str\_\_()}, Brand: {self.brand}, Warranty: {self.warranty\_period} months"

class Clothing(Product):

    def \_\_init\_\_(self, name, price, quantity, sku, size, material):

        super().\_\_init\_\_(name, price, quantity, sku)

        self.size = size

        self.material = material

    def \_\_str\_\_(self):

        return f"Clothing: {super().\_\_str\_\_()}, Size: {self.size}, Material: {self.material}"

class Groceries(Product):

    def \_\_init\_\_(self, name, price, quantity, sku, expiration\_date, is\_organic):

        super().\_\_init\_\_(name, price, quantity, sku)

        self.expiration\_date = expiration\_date

        self.is\_organic = is\_organic

    def is\_expired(self):

        return datetime.now().date() > self.expiration\_date

    def \_\_str\_\_(self):

        organic\_status = "Organic" if self.is\_organic else "Non-Organic"

        return f"Groceries: {super().\_\_str\_\_()}, Expiration Date: {self.expiration\_date}, {organic\_status}"

class Inventory:

    def \_\_init\_\_(self):

        self.products = []

    def add\_product(self, product):

        self.products.append(product)

        print(f"Product added: {product.name}")

    def calculate\_inventory\_value(self):

        total\_value = sum(product.inventory\_value() for product in self.products)

        return total\_value

    def check\_low\_stock(self, threshold=5):

        low\_stock\_products = [product for product in self.products if product.is\_low\_stock(threshold)]

        if low\_stock\_products:

            print("Low stock products:")

            for product in low\_stock\_products:

                print(product)

        else:

            print("No products are low in stock.")

    def apply\_discount\_by\_type(self, product\_type, discount\_percentage):

        for product in self.products:

            if isinstance(product, product\_type):

                product.apply\_discount(discount\_percentage)

                print(f"Discount applied to {product.name}: {discount\_percentage}%")

    def print\_inventory(self):

        if not self.products:

            print("No products in inventory.")

        else:

            for product in self.products:

                print(product)

if \_\_name\_\_ == "\_\_main\_\_":

    inventory = Inventory()

    phone = Electronics("iPhone 12", 999.99, 10, "E123", 24, "Apple")

    laptop = Electronics("Dell XPS", 1299.99, 5, "E124", 36, "Dell")

    shirt = Clothing("T-shirt", 19.99, 50, "C101", "M", "Cotton")

    pants = Clothing("Jeans", 49.99, 30, "C102", "L", "Denim")

    apple = Groceries("Apple", 0.99, 100, "G001", datetime(2024, 10, 31).date(), True)

    banana = Groceries("Banana", 0.69, 80, "G002", datetime(2024, 11, 5).date(), False)

    inventory.add\_product(phone)

    inventory.add\_product(laptop)

    inventory.add\_product(shirt)

    inventory.add\_product(pants)

    inventory.add\_product(apple)

    inventory.add\_product(banana)

    print("\nCurrent Inventory:")

    inventory.print\_inventory()

    inventory.check\_low\_stock(threshold=10)

    print("\nApplying 10% discount to Electronics:")

    inventory.apply\_discount\_by\_type(Electronics, 10)

    total\_value = inventory.calculate\_inventory\_value()

    print(f"\nTotal inventory value: ${total\_value:.2f}")

    print("\nInventory after applying discounts:")

    inventory.print\_inventory()

output

Product added: iPhone 12

Product added: Dell XPS

Product added: T-shirt

Product added: Jeans

Product added: Apple

Product added: Banana

Current Inventory:

Electronics: iPhone 12 (SKU: E123) - Price: 999.99, Quantity: 10, Brand: Apple, Warranty: 24 months

Electronics: Dell XPS (SKU: E124) - Price: 1299.99, Quantity: 5, Brand: Dell, Warranty: 36 months

Clothing: T-shirt (SKU: C101) - Price: 19.99, Quantity: 50, Size: M, Material: Cotton

Clothing: Jeans (SKU: C102) - Price: 49.99, Quantity: 30, Size: L, Material: Denim

Groceries: Apple (SKU: G001) - Price: 0.99, Quantity: 100, Expiration Date: 2024-10-31, Organic

Groceries: Banana (SKU: G002) - Price: 0.69, Quantity: 80, Expiration Date: 2024-11-05, Non-Organic

Low stock products:

Electronics: Dell XPS (SKU: E124) - Price: 1299.99, Quantity: 5, Brand: Dell, Warranty: 36 months

Applying 10% discount to Electronics:

Discount applied to iPhone 12: 10%

Discount applied to Dell XPS: 10%

Total inventory value: $17503.26

Inventory after applying discounts:

Electronics: iPhone 12 (SKU: E123) - Price: 899.99, Quantity: 10, Brand: Apple, Warranty: 24 months

Electronics: Dell XPS (SKU: E124) - Price: 1169.99, Quantity: 5, Brand: Dell, Warranty: 36 months

Clothing: T-shirt (SKU: C101) - Price: 19.99, Quantity: 50, Size: M, Material: Cotton

Clothing: Jeans (SKU: C102) - Price: 49.99, Quantity: 30, Size: L, Material: Denim

Groceries: Apple (SKU: G001) - Price: 0.99, Quantity: 100, Expiration Date: 2024-10-31, Organic

Groceries: Banana (SKU: G002) - Price: 0.69, Quantity: 80, Expiration Date: 2024-11-05, Non-Organic

Week-6 lab

**Exercise 1:**

**Inventory Management System using Python Inheritance**

**Scenario: Designing an Inventory Management System**

You are tasked with designing an **Inventory Management System** that handles various types

of products. Each product shares common attributes but has specific attributes based on the

category of the product. The system must support operations such as adding products,

calculating inventory value, applying discounts, and checking stock levels.

**Problem Definition:**

We need to manage three types of products:

1. **Electronics** (e.g., phones, laptops).

2. **Clothing** (e.g., shirts, pants).

3. **Groceries** (e.g., fruits, vegetables).

All product types share basic attributes like **name**, **price**, **quantity**, and **SKU** (Stock Keeping

Unit). However, they also have specific attributes:

• **Electronics** may have **warranty period** and **brand**.

• **Clothing** has attributes like **size** and **material**.

• **Groceries** include **expiration date** and **organic status**.

Additionally, the system needs to:

1. Add new products.

2. Update stock.

3. Calculate the total value of inventory.

4. Apply discounts based on product type.

5. Check if products are low in stock.

Solution

from datetime import datetime

class Product:

    def \_\_init\_\_(self, name, price, quantity, sku):

        self.name = name

        self.price = price

        self.quantity = quantity

        self.sku = sku

    def update\_stock(self, quantity):

        self.quantity += quantity

    def is\_low\_stock(self, threshold=5):

        return self.quantity < threshold

    def apply\_discount(self, discount\_percentage):

        self.price -= self.price \* (discount\_percentage / 100)

    def inventory\_value(self):

        return self.price \* self.quantity

    def \_\_str\_\_(self):

        return f"{self.name} (SKU: {self.sku}) - Price: {self.price:.2f}, Quantity: {self.quantity}"

class Electronics(Product):

    def \_\_init\_\_(self, name, price, quantity, sku, warranty\_period, brand):

        super().\_\_init\_\_(name, price, quantity, sku)

        self.warranty\_period = warranty\_period

        self.brand = brand

    def \_\_str\_\_(self):

        return f"Electronics: {super().\_\_str\_\_()}, Brand: {self.brand}, Warranty: {self.warranty\_period} months"

class Clothing(Product):

    def \_\_init\_\_(self, name, price, quantity, sku, size, material):

        super().\_\_init\_\_(name, price, quantity, sku)

        self.size = size

        self.material = material

    def \_\_str\_\_(self):

        return f"Clothing: {super().\_\_str\_\_()}, Size: {self.size}, Material: {self.material}"

class Groceries(Product):

    def \_\_init\_\_(self, name, price, quantity, sku, expiration\_date, is\_organic):

        super().\_\_init\_\_(name, price, quantity, sku)

        self.expiration\_date = expiration\_date

        self.is\_organic = is\_organic

    def is\_expired(self):

        return datetime.now().date() > self.expiration\_date

    def \_\_str\_\_(self):

        organic\_status = "Organic" if self.is\_organic else "Non-Organic"

        return f"Groceries: {super().\_\_str\_\_()}, Expiration Date: {self.expiration\_date}, {organic\_status}"

class Inventory:

    def \_\_init\_\_(self):

        self.products = []

    def add\_product(self, product):

        self.products.append(product)

        print(f"Product added: {product.name}")

    def calculate\_inventory\_value(self):

        total\_value = sum(product.inventory\_value() for product in self.products)

        return total\_value

    def check\_low\_stock(self, threshold=5):

        low\_stock\_products = [product for product in self.products if product.is\_low\_stock(threshold)]

        if low\_stock\_products:

            print("Low stock products:")

            for product in low\_stock\_products:

                print(product)

        else:

            print("No products are low in stock.")

    def apply\_discount\_by\_type(self, product\_type, discount\_percentage):

        for product in self.products:

            if isinstance(product, product\_type):

                product.apply\_discount(discount\_percentage)

                print(f"Discount applied to {product.name}: {discount\_percentage}%")

    def print\_inventory(self):

        if not self.products:

            print("No products in inventory.")

        else:

            for product in self.products:

                print(product)

if \_\_name\_\_ == "\_\_main\_\_":

    inventory = Inventory()

    phone = Electronics("iPhone 12", 999.99, 10, "E123", 24, "Apple")

    laptop = Electronics("Dell XPS", 1299.99, 5, "E124", 36, "Dell")

    shirt = Clothing("T-shirt", 19.99, 50, "C101", "M", "Cotton")

    pants = Clothing("Jeans", 49.99, 30, "C102", "L", "Denim")

    apple = Groceries("Apple", 0.99, 100, "G001", datetime(2024, 10, 31).date(), True)

    banana = Groceries("Banana", 0.69, 80, "G002", datetime(2024, 11, 5).date(), False)

    inventory.add\_product(phone)

    inventory.add\_product(laptop)

    inventory.add\_product(shirt)

    inventory.add\_product(pants)

    inventory.add\_product(apple)

    inventory.add\_product(banana)

    print("\nCurrent Inventory:")

    inventory.print\_inventory()

    inventory.check\_low\_stock(threshold=10)

    print("\nApplying 10% discount to Electronics:")

    inventory.apply\_discount\_by\_type(Electronics, 10)

    total\_value = inventory.calculate\_inventory\_value()

    print(f"\nTotal inventory value: ${total\_value:.2f}")

    print("\nInventory after applying discounts:")

    inventory.print\_inventory()

output

Product added: iPhone 12

Product added: Dell XPS

Product added: T-shirt

Product added: Jeans

Product added: Apple

Product added: Banana

Current Inventory:

Electronics: iPhone 12 (SKU: E123) - Price: 999.99, Quantity: 10, Brand: Apple, Warranty: 24 months

Electronics: Dell XPS (SKU: E124) - Price: 1299.99, Quantity: 5, Brand: Dell, Warranty: 36 months

Clothing: T-shirt (SKU: C101) - Price: 19.99, Quantity: 50, Size: M, Material: Cotton

Clothing: Jeans (SKU: C102) - Price: 49.99, Quantity: 30, Size: L, Material: Denim

Groceries: Apple (SKU: G001) - Price: 0.99, Quantity: 100, Expiration Date: 2024-10-31, Organic

Groceries: Banana (SKU: G002) - Price: 0.69, Quantity: 80, Expiration Date: 2024-11-05, Non-Organic

Low stock products:

Electronics: Dell XPS (SKU: E124) - Price: 1299.99, Quantity: 5, Brand: Dell, Warranty: 36 months

Applying 10% discount to Electronics:

Discount applied to iPhone 12: 10%

Discount applied to Dell XPS: 10%

Total inventory value: $17503.26

Inventory after applying discounts:

Electronics: iPhone 12 (SKU: E123) - Price: 899.99, Quantity: 10, Brand: Apple, Warranty: 24 months

Electronics: Dell XPS (SKU: E124) - Price: 1169.99, Quantity: 5, Brand: Dell, Warranty: 36 months

Clothing: T-shirt (SKU: C101) - Price: 19.99, Quantity: 50, Size: M, Material: Cotton

Clothing: Jeans (SKU: C102) - Price: 49.99, Quantity: 30, Size: L, Material: Denim

Groceries: Apple (SKU: G001) - Price: 0.99, Quantity: 100, Expiration Date: 2024-10-31, Organic

Groceries: Banana (SKU: G002) - Price: 0.69, Quantity: 80, Expiration Date: 2024-11-05, Non-Organic

**Exercise 2**

**Building a Payment Processing System**

**You are tasked with designing a Payment Processing System that handles multiple payment**

**methods (Credit Card, PayPal, Bank Transfer). Each payment method has unique steps**

**involved in processing payments, but they all share the common interface of processing a**

**payment and issuing a refund.**

**Problem Definition:**

**The system must support the following payment methods:**

**1. Credit Card Payment: Requires card number, expiry date, and CVV to process**

**payments.**

**2. PayPal Payment: Uses a PayPal account email and password.**

**3. Bank Transfer Payment: Processes payments using a bank account number and a sort**

**code.**

**Each payment method has:**

**• A method to process payments.**

**• A method to issue refunds.**

**• Error handling for failed payments.**

**Solution**

**from abc import ABC, abstractmethod**

**class PaymentMethod(ABC):**

**@abstractmethod**

**def process\_payment(self, amount):**

**pass**

**@abstractmethod**

**def issue\_refund(self, amount):**

**pass**

**class CreditCardPayment(PaymentMethod):**

**def \_\_init\_\_(self, card\_number, expiry\_date, cvv):**

**self.card\_number = card\_number**

**self.expiry\_date = expiry\_date**

**self.cvv = cvv**

**def process\_payment(self, amount):**

**try:**

**if len(self.card\_number) == 16 and len(self.cvv) == 3:**

**print(f"Processed credit card payment of ${amount}.")**

**else:**

**raise ValueError("Invalid card details.")**

**except Exception as e:**

**print(f"Failed to process credit card payment: {e}")**

**def issue\_refund(self, amount):**

**try:**

**print(f"Issued a refund of ${amount} to credit card ending in {self.card\_number[-4:]}.")**

**except Exception as e:**

**print(f"Failed to issue refund on credit card: {e}")**

**class PayPalPayment(PaymentMethod):**

**def \_\_init\_\_(self, email, password):**

**self.email = email**

**self.password = password**

**def process\_payment(self, amount):**

**try:**

**if "@" in self.email and len(self.password) > 5:**

**print(f"Processed PayPal payment of ${amount} for {self.email}.")**

**else:**

**raise ValueError("Invalid PayPal credentials.")**

**except Exception as e:**

**print(f"Failed to process PayPal payment: {e}")**

**def issue\_refund(self, amount):**

**try:**

**print(f"Issued a refund of ${amount} to PayPal account {self.email}.")**

**except Exception as e:**

**print(f"Failed to issue refund on PayPal: {e}")**

**class BankTransferPayment(PaymentMethod):**

**def \_\_init\_\_(self, account\_number, sort\_code):**

**self.account\_number = account\_number**

**self.sort\_code = sort\_code**

**def process\_payment(self, amount):**

**try:**

**if len(self.account\_number) == 10 and len(self.sort\_code) == 6:**

**print(f"Processed bank transfer payment of ${amount}.")**

**else:**

**raise ValueError("Invalid bank details.")**

**except Exception as e:**

**print(f"Failed to process bank transfer payment: {e}")**

**def issue\_refund(self, amount):**

**try:**

**print(f"Issued a refund of ${amount} to bank account {self.account\_number}.")**

**except Exception as e:**

**print(f"Failed to issue refund via bank transfer: {e}")**

**def process\_payments():**

**cc\_payment = CreditCardPayment("1234567812345678", "12/24", "123")**

**cc\_payment.process\_payment(100)**

**cc\_payment.issue\_refund(50)**

**paypal\_payment = PayPalPayment("user@example.com", "password123")**

**paypal\_payment.process\_payment(200)**

**paypal\_payment.issue\_refund(100)**

**bank\_payment = BankTransferPayment("1234567890", "654321")**

**bank\_payment.process\_payment(300)**

**bank\_payment.issue\_refund(150)**

**process\_payments()**

**output**

**Processed credit card payment of $100.**

**Issued a refund of $50 to credit card ending in 5678.**

**Processed PayPal payment of $200 for user@example.com.**

**Issued a refund of $100 to PayPal account user@example.com.**

**Processed bank transfer payment of $300.**

**Issued a refund of $150 to bank account 1234567890.**

**Week-8 lab**

**Exercise 1:**

**File Handling in Python with Error and Exception Handling**

**You are tasked with building a simple data processing system for an e-commerce company**

**that handles product inventories and customer reviews. The company uses two types of files to**

**store data:**

**1. CSV Files: The product inventory is stored in a CSV file with the following columns:**

***Product ID, Product Name, Category, Price, and Stock Quantity.***

**2. JSON Files: Customer reviews for each product are stored in separate JSON files. Each**

**review file contains a list of reviews for a product, where each review has the following**

**fields: *Review ID, User ID, Rating, Comment, and Date.***

**Your system needs to process these files to provide insights and updates on product availability,**

**pricing trends, and customer feedback. Additionally, the system should allow adding new**

**products and updating product information in the CSV file, while handling errors and**

**exceptions gracefully.**

**System Requirements:**

**1. Product Management (CSV File):**

**• Add New Products: Write a function to add a new product to the inventory CSV file.**

**• Update Stock and Price: Implement a function to update the stock quantity and price**

**of an existing product in the CSV file.**

**• Check Product Availability: Write a function that reads the CSV file to check if a**

**product is in stock and, if so, how many units are available.**

**2. Customer Review Management (JSON Files):**

**• Add Customer Review: Implement a function that adds a new customer review to the**

**corresponding JSON file for a product.**

**• Average Rating Calculation: Write a function that reads a product’s JSON review file**

**and calculates its average rating.**

**• Review Search: Implement a search function that reads the JSON file and allows users**

**to search for reviews containing specific keywords in the comment section.**

**3. Data Analysis:**

**• Top Rated Products: Create a function that reads all product review JSON files,**

**calculates the average rating for each product, and lists the top 5 highest-rated products.**

**• Out of Stock Products: Write a function that reads the CSV inventory file and lists all**

**products that are out of stock.**

**• Price Trends: Implement a function to read the CSV file and identify products whose**

**prices have increased or decreased in the past month.**

**4. Error and Exception Handling:**

**• Implement error handling for situations such as:**

**o File Not Found: When the CSV or JSON file is missing.**

**o Invalid Data Format: If the data in the files does not match the expected**

**structure.**

**o File Corruption: When a file contains corrupted data.**

**o Read/Write Permissions: When the system does not have permission to read**

**or write to a file.**

**Solution**

**import csv**

**import json**

**import os**

**from datetime import datetime**

**CSV\_FILE = 'inventory.csv'**

**def add\_new\_product(product\_id, product\_name, category, price, stock\_quantity):**

**try:**

**with open(CSV\_FILE, mode='a', newline='') as file:**

**writer = csv.writer(file)**

**writer.writerow([product\_id, product\_name, category, price, stock\_quantity])**

**print(f"Product {product\_name} added successfully.")**

**except Exception as e:**

**print(f"Error adding new product: {e}")**

**def update\_stock\_and\_price(product\_id, new\_price, new\_stock\_quantity):**

**try:**

**updated = False**

**products = []**

**with open(CSV\_FILE, mode='r') as file:**

**reader = csv.reader(file)**

**for row in reader:**

**if row[0] == product\_id:**

**row[3] = new\_price**

**row[4] = new\_stock\_quantity**

**updated = True**

**products.append(row)**

**if updated:**

**with open(CSV\_FILE, mode='w', newline='') as file:**

**writer = csv.writer(file)**

**writer.writerows(products)**

**print(f"Product {product\_id} updated successfully.")**

**else:**

**print(f"Product {product\_id} not found.")**

**except Exception as e:**

**print(f"Error updating product: {e}")**

**def check\_product\_availability(product\_id):**

**try:**

**with open(CSV\_FILE, mode='r') as file:**

**reader = csv.reader(file)**

**for row in reader:**

**if row[0] == product\_id:**

**return f"Product {row[1]} is in stock: {row[4]} units available."**

**return f"Product {product\_id} not found."**

**except FileNotFoundError:**

**print("Inventory file not found.")**

**except Exception as e:**

**print(f"Error checking product availability: {e}")**

**def add\_customer\_review(product\_id, review\_id, user\_id, rating, comment):**

**json\_file = f"{product\_id}\_reviews.json"**

**review = {**

**"Review ID": review\_id,**

**"User ID": user\_id,**

**"Rating": rating,**

**"Comment": comment,**

**"Date": datetime.now().strftime("%Y-%m-%d")**

**}**

**try:**

**if os.path.exists(json\_file):**

**with open(json\_file, 'r+') as file:**

**reviews = json.load(file)**

**reviews.append(review)**

**file.seek(0)**

**json.dump(reviews, file, indent=4)**

**else:**

**with open(json\_file, 'w') as file:**

**json.dump([review], file, indent=4)**

**print("Review added successfully.")**

**except Exception as e:**

**print(f"Error adding review: {e}")**

**def calculate\_average\_rating(product\_id):**

**json\_file = f"{product\_id}\_reviews.json"**

**try:**

**with open(json\_file, 'r') as file:**

**reviews = json.load(file)**

**total\_rating = sum([review["Rating"] for review in reviews])**

**average\_rating = total\_rating / len(reviews) if reviews else 0**

**return average\_rating**

**except FileNotFoundError:**

**print("Review file not found.")**

**except Exception as e:**

**print(f"Error calculating average rating: {e}")**

**def search\_reviews(product\_id, keyword):**

**json\_file = f"{product\_id}\_reviews.json"**

**try:**

**with open(json\_file, 'r') as file:**

**reviews = json.load(file)**

**matching\_reviews = [review for review in reviews if keyword.lower() in review["Comment"].lower()]**

**return matching\_reviews**

**except FileNotFoundError:**

**print("Review file not found.")**

**except Exception as e:**

**print(f"Error searching reviews: {e}")**

**def top\_rated\_products():**

**product\_ratings = {}**

**try:**

**for file in os.listdir():**

**if file.endswith('\_reviews.json'):**

**product\_id = file.split('\_')[0]**

**average\_rating = calculate\_average\_rating(product\_id)**

**product\_ratings[product\_id] = average\_rating**

**top\_5 = sorted(product\_ratings.items(), key=lambda x: x[1], reverse=True)[:5]**

**return top\_5**

**except Exception as e:**

**print(f"Error calculating top-rated products: {e}")**

**def out\_of\_stock\_products():**

**try:**

**out\_of\_stock = []**

**with open(CSV\_FILE, mode='r') as file:**

**reader = csv.reader(file)**

**for row in reader:**

**if int(row[4]) == 0:**

**out\_of\_stock.append(row[1])**

**return out\_of\_stock**

**except FileNotFoundError:**

**print("Inventory file not found.")**

**except Exception as e:**

**print(f"Error finding out-of-stock products: {e}")**

**def price\_trends():**

**pass**

**try:**

**add\_new\_product("P001", "Laptop", "Electronics", 1200, 10)**

**update\_stock\_and\_price("P001", 1150, 8)**

**print(check\_product\_availability("P001"))**

**add\_customer\_review("P001", "R001", "U001", 5, "Excellent product!")**

**add\_customer\_review("P001", "R002", "U002", 4, "Great value for money.")**

**print("Average Rating:", calculate\_average\_rating("P001"))**

**print("Reviews containing 'great':", search\_reviews("P001", "great"))**

**print("Top Rated Products:", top\_rated\_products())**

**print("Out of Stock Products:", out\_of\_stock\_products())**

**except Exception as e:**

**print(f"An error occurred: {e}")**

**output**

**Product Laptop added successfully.**

**Product P001 updated successfully.**

**Product Laptop is in stock: 8 units available.**

**Review added successfully.**

**Review added successfully.**

**Average Rating: 4.5**

**Reviews containing 'great': [{'Review ID': 'R002', 'User ID': 'U002', 'Rating': 4, 'Comment': 'Great value for money.', 'Date': '2024-10-18'}]**

**Top Rated Products: [('P001', 4.5)]**

**Out of Stock Products: []**

**Week -9 lab**

**Exercise 1:**

**Sales Performance Analysis of XYZ Company**

**Data Overview:**

**The sales data of XYZ Company contains the following attributes for each transaction:**

**• Date: The date of the sale.**

**• Product Name: The name of the product sold.**

**• Units Sold: The number of units sold in the transaction.**

**• Revenue: The total revenue generated by the sale.**

**• Region: The geographical region where the sale occurred.**

**• Discount Offered (%): The percentage of discount offered on the sale.**

**• Salesperson: The name of the salesperson responsible for the sale.**

**Insert At least 20 data and create the dataframe.**

**1. What are the top 3 sales transactions with the highest revenue?**

**2. How many units of each product were sold?**

**3. What is the total revenue after applying discounts?**

**4. Which sales transaction had the highest discount offered, and how much revenue did**

**it generate after applying the discount?**

**5. Which salesperson generated the highest total revenue?**

**6. What is the average discount offered by each salesperson?**

**7. How much revenue was generated in each region?**

**8. In which region did Alice generate the highest sales?**

**9. Which product generated the highest revenue per unit sold?**

**10. How many transactions were rated as "High" performance?**

**11. Which salesperson sold the most units in the North region without offering any**

**discount?**

**12. What is the average revenue per unit sold in each region for each product?**

**13. Which salesperson has the highest average revenue after discounts, and how does it**

**compare between regions?**

**14. What is the cumulative total revenue over time for each salesperson?15. For each salesperson, rank the transactions by revenue, and find the top 2 transactions**

**for each.**

**16. How has the total revenue generated by each product changed over time? Show**

**cumulative revenue for each product per day.**

**17. Analyze how discounts affect revenue. For each product, what is the average revenue**

**generated with a discount compared to without a discount?**

**18. What is the weighted average discount offered by each salesperson, weighted by the**

**revenue they generated?**

**19. What percentage of the total revenue does each region contribute? Compare it to the**

**total revenue per region.**

**Solution**

import pandas as pd

# Sample Data

data = {

"Date": ["2024-01-01", "2024-01-02", "2024-01-03", "2024-01-04", "2024-01-05",

"2024-01-06", "2024-01-07", "2024-01-08", "2024-01-09", "2024-01-10",

"2024-01-11", "2024-01-12", "2024-01-13", "2024-01-14", "2024-01-15",

"2024-01-16", "2024-01-17", "2024-01-18", "2024-01-19", "2024-01-20"],

"Product Name": ["Product A", "Product B", "Product C", "Product D", "Product E",

"Product A", "Product B", "Product C", "Product D", "Product E",

"Product A", "Product B", "Product C", "Product D", "Product E",

"Product A", "Product B", "Product C", "Product D", "Product E"],

"Units Sold": [10, 20, 30, 15, 25, 5, 35, 45, 25, 30, 20, 10, 50, 40, 15, 25, 35, 45, 10, 20],

"Revenue": [100, 200, 300, 150, 250, 50, 350, 450, 250, 300, 200, 100, 500, 400, 150, 250, 350, 450, 100, 200],

"Region": ["North", "South", "East", "West", "North", "South", "East", "West", "North", "South",

"East", "West", "North", "South", "East", "West", "North", "South", "East", "West"],

"Discount Offered (%)": [5, 10, 0, 20, 15, 5, 10, 0, 20, 15, 5, 10, 0, 20, 15, 5, 10, 0, 20, 15],

"Salesperson": ["Alice", "Bob", "Charlie", "David", "Eve", "Alice", "Bob", "Charlie", "David", "Eve",

"Alice", "Bob", "Charlie", "David", "Eve", "Alice", "Bob", "Charlie", "David", "Eve"]

}

# Create DataFrame

df = pd.DataFrame(data)

# Solutions to questions

# 1. Top 3 sales transactions with the highest revenue

top\_3\_revenue = df.nlargest(3, 'Revenue')

print("Top 3 sales transactions with the highest revenue:")

print(top\_3\_revenue)

# 2. Units sold per product

units\_sold\_per\_product = df.groupby('Product Name')['Units Sold'].sum()

print("\nUnits sold per product:")

print(units\_sold\_per\_product)

# 3. Total revenue after applying discounts

df['Revenue After Discount'] = df['Revenue'] \* (1 - df['Discount Offered (%)'] / 100)

total\_revenue\_after\_discounts = df['Revenue After Discount'].sum()

print("\nTotal revenue after applying discounts:")

print(total\_revenue\_after\_discounts)

# 4. Sales transaction with the highest discount offered and revenue after discount

highest\_discount\_transaction = df.loc[df['Discount Offered (%)'].idxmax()]

print("\nSales transaction with the highest discount offered:")

print(highest\_discount\_transaction)

# 5. Salesperson with the highest total revenue

total\_revenue\_per\_salesperson = df.groupby('Salesperson')['Revenue'].sum()

highest\_revenue\_salesperson = total\_revenue\_per\_salesperson.idxmax()

print("\nSalesperson with the highest total revenue:")

print(highest\_revenue\_salesperson)

# 6. Average discount offered by each salesperson

average\_discount\_per\_salesperson = df.groupby('Salesperson')['Discount Offered (%)'].mean()

print("\nAverage discount offered by each salesperson:")

print(average\_discount\_per\_salesperson)

# 7. Revenue generated in each region

revenue\_per\_region = df.groupby('Region')['Revenue'].sum()

print("\nRevenue generated in each region:")

print(revenue\_per\_region)

# 8. Region where Alice generated the highest sales

alice\_sales\_per\_region = df[df['Salesperson'] == 'Alice'].groupby('Region')['Revenue'].sum()

region\_highest\_sales\_alice = alice\_sales\_per\_region.idxmax()

print("\nRegion where Alice generated the highest sales:")

print(region\_highest\_sales\_alice)

# 9. Product that generated the highest revenue per unit sold

revenue\_per\_unit = df.groupby('Product Name')['Revenue'].sum() / df.groupby('Product Name')['Units Sold'].sum()

product\_highest\_revenue\_per\_unit = revenue\_per\_unit.idxmax()

print("\nProduct that generated the highest revenue per unit sold:")

print(product\_highest\_revenue\_per\_unit)

# 10. Number of transactions rated as "High" performance

high\_performance\_transactions = df[df['Revenue After Discount'] > df['Revenue After Discount'].mean()]

num\_high\_performance\_transactions = len(high\_performance\_transactions)

print("\nNumber of transactions rated as 'High' performance:")

print(num\_high\_performance\_transactions)

# 11. Salesperson who sold the most units in the North region without offering any discount

north\_no\_discount = df[(df['Region'] == 'North') & (df['Discount Offered (%)'] == 0)]

salesperson\_most\_units\_north\_no\_discount = north\_no\_discount.groupby('Salesperson')['Units Sold'].sum().idxmax()

print("\nSalesperson who sold the most units in the North region without offering any discount:")

print(salesperson\_most\_units\_north\_no\_discount)

# 12. Average revenue per unit sold in each region for each product

average\_revenue\_per\_unit\_region\_product = df.groupby(['Region', 'Product Name'])['Revenue'].sum() / df.groupby(['Region', 'Product Name'])['Units Sold'].sum()

print("\nAverage revenue per unit sold in each region for each product:")

print(average\_revenue\_per\_unit\_region\_product)

# 13. Salesperson with the highest average revenue after discounts

average\_revenue\_after\_discounts\_per\_salesperson = df.groupby('Salesperson')['Revenue After Discount'].mean()

highest\_avg\_revenue\_after\_discounts\_salesperson = average\_revenue\_after\_discounts\_per\_salesperson.idxmax()

print("\nSalesperson with the highest average revenue after discounts:")

print(highest\_avg\_revenue\_after\_discounts\_salesperson)

# 14. Cumulative total revenue over time for each salesperson

df['Cumulative Revenue'] = df.groupby('Salesperson')['Revenue'].cumsum()

print("\nCumulative total revenue over time for each salesperson:")

print(df[['Salesperson', 'Date', 'Cumulative Revenue']])

# 15. Rank transactions by revenue for each salesperson and find the top 2 transactions for each

ranked\_transactions\_by\_salesperson = df.groupby('Salesperson').apply(lambda x: x.nlargest(2, 'Revenue'))

print("\nTop 2 transactions by revenue for each salesperson:")

print(ranked\_transactions\_by\_salesperson)

# 16. Cumulative revenue for each product per day

df['Cumulative Product Revenue'] = df.groupby(['Product Name'])['Revenue'].cumsum()

print("\nCumulative revenue for each product per day:")

print(df[['Product Name', 'Date', 'Cumulative Product Revenue']])

# 17. Analyze how discounts affect revenue for each product

revenue\_with\_discount = df[df['Discount Offered (%)'] > 0].groupby('Product Name')['Revenue After Discount'].mean()

revenue\_without\_discount = df[df['Discount Offered (%)'] == 0].groupby('Product Name')['Revenue'].mean()

print("\nAverage revenue with discount vs without discount:")

print(pd.DataFrame({'With Discount': revenue\_with\_discount, 'Without Discount': revenue\_without\_discount}))

# 18. Weighted average discount offered by each salesperson

weighted\_avg\_discount = df.groupby('Salesperson').apply(lambda x: (x['Discount Offered (%)'] \* x['Revenue']).sum() / x['Revenue'].sum())

print("\nWeighted average discount offered by each salesperson:")

print(weighted\_avg\_discount)

# 19. Percentage of total revenue contributed by each region

total\_revenue = df['Revenue'].sum()

revenue\_percentage\_per\_region = (df.groupby('Region')['Revenue'].sum() / total\_revenue) \* 100

print("\nPercentage of total revenue contributed by each region:")

print(revenue\_percentage\_per\_region)

solution

Top 3 sales transactions with the highest revenue:

          Date Product Name  ...  Discount Offered (%)  Salesperson

12  2024-01-13    Product C  ...                     0      Charlie

7   2024-01-08    Product C  ...                     0      Charlie

17  2024-01-18    Product C  ...                     0      Charlie

[3 rows x 7 columns]

Units sold per product:

Product Name

Product A     60

Product B    100

Product C    170

Product D     90

Product E     90

Name: Units Sold, dtype: int64

Total revenue after applying discounts:

4655.0

Sales transaction with the highest discount offered:

Date                      2024-01-04

Product Name               Product D

Units Sold                        15

Revenue                          150

Region                          West

Discount Offered (%)              20

Salesperson                    David

Revenue After Discount         120.0

Name: 3, dtype: object

Salesperson with the highest total revenue:

Charlie

Average discount offered by each salesperson:

Salesperson

Alice       5.0

Bob        10.0

Charlie     0.0

David      20.0

Eve        15.0

Name: Discount Offered (%), dtype: float64

Revenue generated in each region:

Region

East     1100

North    1450

South    1400

West     1150

Name: Revenue, dtype: int64

Region where Alice generated the highest sales:

West

Product that generated the highest revenue per unit sold:

Product A

Number of transactions rated as 'High' performance:

9

Salesperson who sold the most units in the North region without offering any discount:

Charlie

Average revenue per unit sold in each region for each product:

Region  Product Name

East    Product A       10.0

        Product B       10.0

        Product C       10.0

        Product D       10.0

        Product E       10.0

North   Product A       10.0

        Product B       10.0

        Product C       10.0

        Product D       10.0

        Product E       10.0

South   Product A       10.0

        Product B       10.0

        Product C       10.0

        Product D       10.0

        Product E       10.0

West    Product A       10.0

        Product B       10.0

        Product C       10.0

        Product D       10.0

        Product E       10.0

dtype: float64

Salesperson with the highest average revenue after discounts:

Charlie

Cumulative total revenue over time for each salesperson:

   Salesperson        Date  Cumulative Revenue

0        Alice  2024-01-01                 100

1          Bob  2024-01-02                 200

2      Charlie  2024-01-03                 300

3        David  2024-01-04                 150

4          Eve  2024-01-05                 250

5        Alice  2024-01-06                 150

6          Bob  2024-01-07                 550

7      Charlie  2024-01-08                 750

8        David  2024-01-09                 400

9          Eve  2024-01-10                 550

10       Alice  2024-01-11                 350

11         Bob  2024-01-12                 650

12     Charlie  2024-01-13                1250

13       David  2024-01-14                 800

14         Eve  2024-01-15                 700

15       Alice  2024-01-16                 600

16         Bob  2024-01-17                1000

17     Charlie  2024-01-18                1700

18       David  2024-01-19                 900

19         Eve  2024-01-20                 900

C:\Users\ojasa\AppData\Local\Temp\tempCodeRunnerFile.python:104: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include\_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to silence this warning.

  ranked\_transactions\_by\_salesperson = df.groupby('Salesperson').apply(lambda x: x.nlargest(2, 'Revenue'))

Top 2 transactions by revenue for each salesperson:

                      Date  ... Cumulative Revenue

Salesperson                 ...

Alice       15  2024-01-16  ...                600

            10  2024-01-11  ...                350

Bob         6   2024-01-07  ...                550

            16  2024-01-17  ...               1000

Charlie     12  2024-01-13  ...               1250

            7   2024-01-08  ...                750

David       13  2024-01-14  ...                800

            8   2024-01-09  ...                400

Eve         9   2024-01-10  ...                550

            4   2024-01-05  ...                250

[10 rows x 9 columns]

Cumulative revenue for each product per day:

   Product Name        Date  Cumulative Product Revenue

0     Product A  2024-01-01                         100

1     Product B  2024-01-02                         200

2     Product C  2024-01-03                         300

3     Product D  2024-01-04                         150

4     Product E  2024-01-05                         250

5     Product A  2024-01-06                         150

6     Product B  2024-01-07                         550

7     Product C  2024-01-08                         750

8     Product D  2024-01-09                         400

9     Product E  2024-01-10                         550

10    Product A  2024-01-11                         350

11    Product B  2024-01-12                         650

12    Product C  2024-01-13                        1250

13    Product D  2024-01-14                         800

14    Product E  2024-01-15                         700

15    Product A  2024-01-16                         600

16    Product B  2024-01-17                        1000

17    Product C  2024-01-18                        1700

18    Product D  2024-01-19                         900

19    Product E  2024-01-20                         900

Average revenue with discount vs without discount:

              With Discount  Without Discount

Product Name

Product A            142.50               NaN

Product B            225.00               NaN

Product C               NaN             425.0

Product D            180.00               NaN

Product E            191.25               NaN

C:\Users\ojasa\AppData\Local\Temp\tempCodeRunnerFile.python:120: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include\_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to silence this warning.

  weighted\_avg\_discount = df.groupby('Salesperson').apply(lambda x: (x['Discount Offered (%)'] \* x['Revenue']).sum() / x['Revenue'].sum())

Weighted average discount offered by each salesperson:

Salesperson

Alice       5.0

Bob        10.0

Charlie     0.0

David      20.0

Eve        15.0

dtype: float64

Percentage of total revenue contributed by each region:

Region

East     21.568627

North    28.431373

South    27.450980

West     22.549020

Name: Revenue, dtype: float64