**Question 1**

import datetime

import logging

import random

class Customer:

    def \_\_init\_\_(self, name: str, phone: str, email: str):

        self.name: str = name

        self.phone: str = phone

        self.email: str = email

    @property

    def name(self):

        return self.\_name

    @property

    def phone(self):

        return self.\_phone

    @property

    def email(self):

        return self.\_email

    @name.setter

    def name(self, new\_name):

        self.\_name = new\_name

    @phone.setter

    def phone(self, new\_phone):

        self.\_phone = new\_phone

    @email.setter

    def email(self, new\_email):

        self.\_email = new\_email

class Stock:

    def \_\_init\_\_(self, book\_name: str, author: str, price: float):

        self.book\_name: str = book\_name

        self.author: str = author

        self.price: float = price

    @property

    def book\_name(self):

        return self.\_book\_name

    @property

    def author(self):

        return self.\_author

    @property

    def price(self):

        return self.\_price

    @book\_name.setter

    def book\_name(self, new\_book\_name):

        self.\_book\_name = new\_book\_name

    @author.setter

    def author(self, new\_author):

        self.\_author = new\_author

    @price.setter

    def price(self, new\_price):

        self.\_price = new\_price

class Order:

    def \_\_init\_\_(self, customer: Customer, stock: Stock):

        self.customer: Customer = customer

        self.stock: Stock = stock

    @property

    def customer(self):

        return self.\_customer

    @property

    def stock(self):

        return self.\_stock

    @customer.setter

    def customer(self, new\_customer):

        self.\_customer = new\_customer

    @stock.setter

    def stock(self, new\_stock):

        self.\_stock = new\_stock

class Shipping:

    urgent\_cost: float = 5.45

    basic\_cost: float  = 3.95

    def \_\_init\_\_(self, order: Order, ship\_date: datetime):

        self.order: Order = order

        self.ship\_date: datetime = ship\_date

        self.ship\_cost: int = 0

        self.count\_urgent: int = 0

    def set\_ship\_cost(self, ship\_cost: float):

        self.ship\_cost = ship\_cost

    # calculating the shipping cost. 3.95 if not urgent, 5.45 if it is urgent

    def calc\_ship\_cost(self, is\_urgent: bool):

        if is\_urgent:

            self.ship\_cost = self.urgent\_cost

            self.add\_urgent\_shipment()

        else:

            self.ship\_cost = self.basic\_cost

        return self.ship\_cost

    # helper function to increment the count urgent for a given shipment. could allow us to extend our functionality in case we need have many levels of urgency

    def add\_urgent\_shipment(self):

        self.count\_urgent += 1

    @property

    def ship\_date(self):

        return self.\_ship\_date

    @property

    def ship\_cost(self):

        return self.\_ship\_cost

    @ship\_date.setter

    def ship\_date(self, new\_ship\_date):

        self.\_ship\_date = new\_ship\_date

    @ship\_cost.setter

    def ship\_cost(self, new\_ship\_cost):

        self.\_ship\_cost = new\_ship\_cost

class Invoice:

    def \_\_init\_\_(self, invoice\_nbr: str, stock: Stock, ship\_order: Shipping):

        self.invoice\_nbr: str = invoice\_nbr

        self.stock: Stock = stock

        self.ship\_order: Shipping = ship\_order

        self.total\_cost: int = 0

    # The invoice value is the price of the book + the price of the shipping

    def invoice(self):

        self.total\_cost = self.stock.price + self.ship\_order.ship\_cost

        return self.total\_cost

    @property

    def invoice\_nbr(self):

        return self.\_invoice\_nbr

    @invoice\_nbr.setter

    def invoice\_nbr(self, new\_invoice\_nbr):

        self.\_invoice\_nbr = new\_invoice\_nbr

class BookStore:

    def \_\_init\_\_(self):

        self.invoices: list[Invoice] = []

    # function to search through all the invoices in the bookstore looking for invoices matching the nbr number

    def search\_invoice(self, nbr: str):

        for invoice in self.invoices:

            if invoice.invoice\_nbr == nbr:

                return invoice

        else:

            print("Invoice not found")

    # helper function to return all invoices from the bookstore

    def get\_all\_invoices(self):

        return self.invoices

    @property

    def invoices(self):

        return self.\_invoices

    @invoices.setter

    def invoices(self, new\_invoices):

        self.\_invoices = new\_invoices

class Test:

    def main():

        logging.info("Creating 3 Test Customer Objects")

        customer1 = Customer("Mike", "".join([str(random.randint(0, 9)) for \_ in range(0, 11)]), "mike@gmail.com")

        customer2 = Customer("Alice", "".join([str(random.randint(0, 9)) for \_ in range(0, 11)]), "alice@hotmail.co.uk")

        customer3 = Customer("Bob", "".join([str(random.randint(0, 9)) for \_ in range(0, 11)]), "bob@btinternet.com")

        logging.info("Creating 3 Test Stock Objects")

        stock1 = Stock("A Christmas Carol", "Charles Dickens", 2.73)

        stock2 = Stock("To Kill a Mockingbird", "Harper Lee", 8.16)

        stock3 = Stock("The Great Gatsby", "F. Scott Fitzgerald", 57.17)

        logging.info("Creating 3 Test Order Objects")

        order1 = Order(customer1, stock1)

        order2 = Order(customer2, stock2)

        order3 = Order(customer3, stock3)

        logging.info("Creating 3 Test Shipping Objects")

        shipping1 = Shipping(order1, datetime.date.today())

        shipping2 = Shipping(order2, datetime.date.today())

        shipping3 = Shipping(order3, datetime.date.today())

        logging.info("Setting Shipping Costs")

        shipping1.set\_ship\_cost(shipping1.calc\_ship\_cost(True))

        shipping2.set\_ship\_cost(shipping2.calc\_ship\_cost(False))

        shipping3.set\_ship\_cost(shipping3.calc\_ship\_cost(True))

        logging.info("Creating 3 Invoices")

        invoice1 = Invoice("INV0001", stock1, shipping1)

        invoice2 = Invoice("INV0002", stock2, shipping2)

        invoice3 = Invoice("INV0003", stock3, shipping3)

        logging.info("Creating a Bookstore")

        bookstore = BookStore()

        logging.info("Adding the invoices to our bookstore")

        bookstore.invoices.append(invoice1)

        bookstore.invoices.append(invoice2)

        bookstore.invoices.append(invoice3)

        print(f"Number of urgent shipments: {shipping1.count\_urgent}")

        print(f"Invoice 1 total cost: {invoice1.invoice():.2f}")

        print(f"Invoice 2 total cost: {invoice2.invoice():.2f}")

        print(f"Invoice 3 total cost: {invoice3.invoice():.2f}")

        print(f"Number of Invoices in BookStore: {bookstore.get\_all\_invoices()}")

        assert bookstore.search\_invoice("INV0004") is None

        assert shipping1.count\_urgent == 1

        assert len(bookstore.get\_all\_invoices()) == 3

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

    Test.main()

**Test Cases:**

* Scenario 1 (Running as seen in the code above):

Output:

**Text

Description automatically generated**

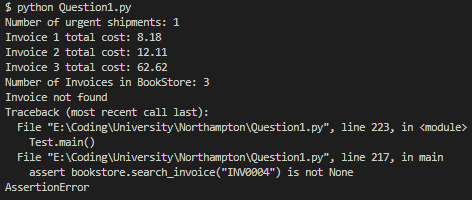
* Scenario 2 (Changing the assertions):

assert bookstore.search\_invoice("INV0004") is not None

        assert shipping1.count\_urgent == 1

        assert len(bookstore.get\_all\_invoices()) == 4

Output:

****

**Question 2**

# Importing Regex to use later to strip out punctuation

import re

import os

import sys

class SentimentLexicon:

    def \_\_init\_\_(self):

        # Initialising the Dictionary that will store all the words from the input values with their respective positive/negative values

        self.dictionary = {}

    def create\_lexicon(self, positive\_words\_file, negative\_words\_file):

# os.path.join(sys.path[0]) takes the path of the directory the python file is running in and joins it with the file name.

      # This means the input needs to be in the same folder as the code.

        with open((os.path.join(sys.path[0], positive\_words\_file)), 'r') as positive\_words:

            for line in positive\_words: # looping through each line in the input file

                if ";" not in line: # the first lines of the input file are commented and contain semi colons. These need to be omitted.

                    word = line.strip() # stripping away /n characters

                    self.dictionary[word] = 1 # adding positive words to the dictionary with the value of positive 1

        with open((os.path.join(sys.path[0], negative\_words\_file)), 'r') as negative\_words:

            for line in negative\_words: # looping through each line in the input file

                if ";" not in line:  # the first lines of the input file are commented and contain semi colons. These need to be omitted.

                    word = line.strip() # stripping away /n characters

                    self.dictionary[word] = -1 # adding positive words to the dictionary with the value of negative 1

class Classifier:

    def \_\_init\_\_(self, sentiment\_lexicon):

        self.sentiment\_lexicon: SentimentLexicon = sentiment\_lexicon # giving the classifier access to the already instantiated SentimentLexicon - This could be done via inheritence

        self.classified\_sentences: list[dict] = [] # will store the results of all the processed texts

# a short list of negation words which will allow us to invert the score of positive of negative words

        self.negation\_words = ["not", "never", "nothing", "nowhere", "noone", "none",

                               "havent", "hasnt", "hadnt", "cant", "couldnt", "shouldnt",

                               "wont", "wouldnt", "dont", "doesnt", "didnt", "isnt", "arent", "aint"]

    def classify(self, text: str):

        score = 0

        negation = False # the word "bad" would be scored as -1 but if it is "not bad" it should be scored a 1 since it is inverted with the use of not

        for word in text.split():

            word = re.sub(r'[^\w\s]','', word) # stripping punctuation from the word using regex

            if word in self.negation\_words:

                negation = True if not negation else False # if negation is False make it True. If it is True make it False. This accounts for double negatives like "not not bad"

            if word in self.sentiment\_lexicon.dictionary:

                if negation: # if negation is true the next adjective in the list of dictionary words from the lexicon class will have its score inverted. 1 => -1 or -1 => 1

                    score -= self.sentiment\_lexicon.dictionary[word]

                    negation = False # after the negative has been used negation is then reset to False

                else:

                    score += self.sentiment\_lexicon.dictionary[word]

        if score > 0: # if the sentence score > 0 sentiment is 1

            sentiment = 1

        elif score < 0: # if the sentence score < 0 sentiment is 0

            sentiment = -1

        else: # if the sentence score is not greater than or less than 0 it must be 0 - the sentiment is nothing then.

            sentiment = 0

        result = {"text": text, "sentiment": sentiment}

        self.classified\_sentences.append(result) # appending the current result to the history of classified\_sentences

    def classify\_multiple\_words(self, list\_of\_text): # helper function to loop through and classify multiple words/sentences.

        for text in list\_of\_text:

            self.classify(text)

    def show\_classified\_sentences(self): # helper function to display all previously classified sentences.

        for sentence in self.classified\_sentences:

            print(sentence)

lexicon = SentimentLexicon() # instantiating the lexicon

lexicon.create\_lexicon("positive-words.txt", "negative-words.txt") # calling the function that will populate the lexicon dictionary

classifier = Classifier(lexicon) # instatiating the classifier and passing in the lexicon

example\_text = [ # example sentences

    "I love Python",

    "Python is the language I love!",

    "The iPhone is clearly not the most terrible and worst phone ever. It is the best."

]

classifier.classify\_multiple\_words(example\_text) # calling the helper function that loops through the sentences above and classifies them

classifier.show\_classified\_sentences() # calling the helper function that loops through the stored sentence,sentiment pairs and prints them

**Test Cases:**

* Scenario 1**:**

example\_text = [ # example sentences

    "I love Python",

    "Python is the language I love!",

    "The iPhone is clearly not the most terrible and worst phone ever. It is the best."

]

Output:

**Text

Description automatically generated**

* Scenario 2:

example\_text = [ # example sentences

    "I hate every programming language including Python!",

    "I really don't hate this but it's not good...",

    "Anyone who likes coding is good but not great because it's difficult."

]

Output:

**A screenshot of a computer

Description automatically generated with medium confidence**

**Question 3**

**Text

Description automatically generated**

Output:

Graphical user interface, text

Description automatically generated

**Question 4**

**Text

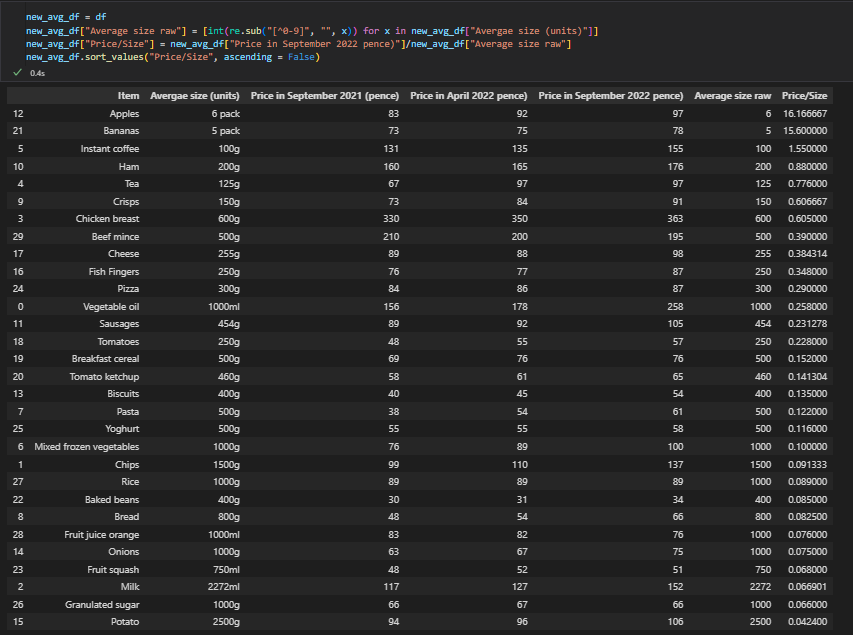
Description automatically generated**

Data Exploration:

A screenshot of a computer

Description automatically generated with medium confidence

We can see from a simple describe function of the dataset that the minimum price of a single item in the dataset has stayed almost the same (only increasing by £4) whereas the maximum price of a single item in the dataset has increased by £33. We can also see that there is a smaller jump with the median price April 2022 => September 2022 (only increasing by £3) compared to the previous jump (£9) however the increased by £10. This means that generally items under the median price of £88 where the items that increased the most in value.



We’re able to created an extra column which have the average size with the units stripped away. We can use this to work out the price/unit (milliliter or gram). I did this for the September 2022 prices but this can be done over all three time periods or over any of the individual ones. We can see that the most expensive item via this metric are fruits. This makes sense as 97p for only 6 apples is quite a lot. This is only because the unit used for the fruit is “pack” instead of “grams”. We can discount these as anomolies as we know that 6 apples or 5 bananas would be quite heavy. If we used grams instead the values would be more logical. We can then see that the most expensive item is instant coffee. This makes a lot of sense as coffee is generally quite expensive and you don’t get a lot. The inverse is that potatos are the cheapest per gram. This is also expected as potatos are incredibly cheap and you get lots of them. Through this analysis you will notice the bottom 10 items (the 10 cheapest per size) as often staple items in all cookery (Potatos => Chips). This is the reason why these items are such staples of cookery as everyone can afford them.

Comparing these values over the years would not be useful currently as the quauntity value is the same and would just give the same values as the next graph with just a factor of the quantity taken out.

Chart

Description automatically generated

Through plotting the data on a bar chart with price on the y axis and item on the x axis with 3 split bars for each time of data we can see that all the food items have increased in price from September 2021 => September 2022 with the exception of Fruit Orange Juice and Granulated Sugar and Beef Mince.

Chart, bar chart

Description automatically generated

The summed prices of all the items has obviously also increased over time. Here we can see the difference in price between all the summed items. We can then actually work out price increase. Since the gradient of the line between April 2022 => September 2022 is steeper than the Increase from September 2021 => April 2022 we know that the prices have increased more. The exact values of the increase are

|  |  |  |  |
| --- | --- | --- | --- |
| Dates | Price Points | Raw Diff | % Change |
| September 2021 => April 2022 | 2742 => 2929 | +187 | +6.8% |
| April 2022 => September 2022 | 2929 => 3210 | +281 | +9.6% |

Text

Description automatically generated

By performing a groupby on the Average size and then grouping by the average size we can see how many items fall into each band of size. This would be equal to a SQL query: SELECT average\_raw\_size, COUNT(\*) FROM dataframe GROUP BY average\_raw\_size. This is done to see the distribution of weights. From this we can see that the most popular weight is 1000g (1Kg) but as expected there is a long tail. We could equally try to fit the raw\_size into bands of 200. We could use an analysis like that to work out how much you could fit in a shopping bag for example. Given that you can only carry so much.

A screenshot of a computer

Description automatically generated with medium confidence

By comparing the means and maxes of items between September 2021 => September 2022 I was able to assert that the average increase per item was 17.07% but one item had a single increase of 65% in just one year!

Tkinter Code:

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import tkinter as tk

import pandas as pd

import os

import sys

import matplotlib.pyplot as plt

from pandas import DataFrame

def graph\_data(dataframe: DataFrame):

    fig1 = plt.figure()

    fig2 = plt.figure()

    ax1 = fig1.add\_subplot(111)

    ax2 = fig2.add\_subplot(111)

    summed = dataframe.sum(numeric\_only=True)

    summed.plot(figsize=(15,10), kind='bar', ax=ax2)

    summed.plot(figsize=(15,10), kind='line', color='red', linewidth='5', ax=ax2)

    dataframe.plot(x="Item", kind='bar', figsize=(10,5), ax=ax1)

    #Add x and y axis labels and a title

    plt.xlabel('Item')

    plt.ylabel('Price')

    plt.title('Price of Each Item in given times')

    plt.text(0,1.2,'This text will be on top of the graph', transform=ax1.transAxes)

    fig1.set\_tight\_layout(True)

    fig2.set\_tight\_layout(True)

    ax1.legend(loc='upper left', bbox\_to\_anchor=(1,1), fontsize=6)

    canvas1 = FigureCanvasTkAgg(fig1, master=root)

    canvas1.get\_tk\_widget().pack()

    canvas1.draw()

    canvas2 = FigureCanvasTkAgg(fig2, master=root)

    canvas2.get\_tk\_widget().pack()

    canvas2.draw()

def on\_closing():

    root.destroy()

def main():

    dataframe: DataFrame = pd.read\_excel((os.path.join(sys.path[0],'food-data.xlsx')))

    button = tk.Button(root, text="Show Graph", bg="green", fg="white", font=("Helvetica", 16), command=lambda: graph\_data(dataframe))

    button.pack()

    root.protocol("WM\_DELETE\_WINDOW", on\_closing)

    root.mainloop()

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

    root = tk.Tk()

    main()

Screenshots:

Graphical user interface

Description automatically generated

A picture containing chart

Description automatically generated