**PAI Assignment 1**

## Q1:

transactionsList = [

    {'orderId': 1001, 'customerId': 'cust\_Ahmed', 'productId': 'prod\_15'},

    {'orderId': 1001, 'customerId': 'cust\_Ahmed', 'productId': 'prod\_12'},

    {'orderId': 1002, 'customerId': 'cust\_Bisma', 'productId': 'prod\_10'},

    {'orderId': 1002, 'customerId': 'cust\_Bisma', 'productId': 'prod\_12'},

    {'orderId': 1003, 'customerId': 'cust\_Faisal', 'productId': 'prod\_10'},

    {'orderId': 1004, 'customerId': 'cust\_Faisal', 'productId': 'prod\_15'},

    {'orderId': 1034, 'customerId': 'cust\_Bisma', 'productId': 'prod\_15'}

]

productCatalog = {

    'prod\_10': 'Wireless Mouse',

    'prod\_12': 'Keyboard',

    'prod\_15': 'USB-C Hub'

}

def processTransactions(transactionsList):

    dic={}

    for i in transactionsList:

        dic.setdefault(i["customerId"], set()).add(i["productId"])

    return dic

def findFrequentPairs (customerData):

    pairCount={}

    for products in customerData.values():

        productList=list(products)

        for i in range(len(productList)):

            for j in range(i+1,len(productList)):

                p1=productList[i]

                p2=productList[j]

                pair=tuple(sorted((p1,p2)))

                pairCount[pair]=pairCount.get(pair,0)+1

    return pairCount

def getRecommendations(targetProductId,frequentPairs):

    related={}

    for(p1,p2), count in frequentPairs.items():

        if targetProductId in (p1,p2):

            other=p2 if p1 == targetProductId else p1

            related[other]=count

    return sorted(related.items(),key=lambda x: x[1],reverse=True)

def generateReport(targetProductId, recommendations, catalog):

    print(f"\nRecommendations for: {catalog[targetProductId]}")

    productIds,counts= zip(\*recommendations)if recommendations else([],[])

    for idx,(pid,count) in enumerate(zip(productIds,counts),start=1):

        print(f"{idx}. {catalog[pid]}, (bought togetter {count} times)")

def main():

    customerData=processTransactions(transactionsList)

    frequentPairs=findFrequentPairs (customerData)

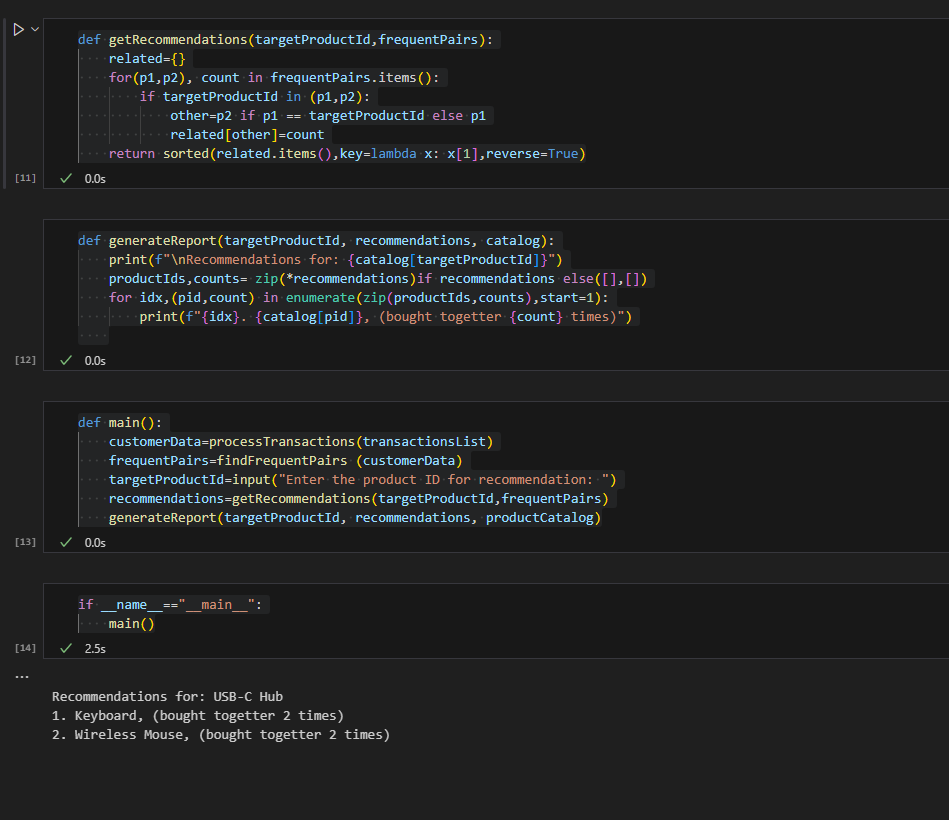
    targetProductId=input("Enter the product ID for recommendation: ")

    recommendations=getRecommendations(targetProductId,frequentPairs)

    generateReport(targetProductId, recommendations, productCatalog)

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

    main()



## Q2:

postsList = [

    {'id': 1, 'text': "I LOVE the new #GutPhone! Battery life is amazing."},

    {'id': 2, 'text': "My #GutPhone is a total disaster. The screen is already broken!"},

    {'id': 3, 'text': "Worst customer service ever from @GutPhoneSupport. Avoid!"},

    {'id': 4, 'text': "The @GutPhoneSupport team was helpful and resolved my issue. Great service!"}

]

PUNCTUATION\_CHARS = ['.', ',', '!', '?', ':', ';', "'", '"']

STOPWORDS\_SET = {'i', 'me', 'my', 'an', 'the', 'is', 'am', 'was', 'to', 'of', 'by', 'for', 'and', 'with', 'this', 'that'}

POSITIVE\_WORDS = {'love', 'amazing', 'great', 'helpful', 'resolved'}

NEGATIVE\_WORDS = {'disaster', 'broken', 'bad', 'worst', 'avoid'}

def preprocessText(text, punctuationList, stopwordsSet):

    text = text.lower()

    for ch in punctuationList:

        text = text.replace(ch, "")

    words = text.split()

    return [word for word in words if word not in stopwordsSet]

def analyzePosts(postsList, punctuation, stopwords, positive, negative):

    def scorePost(post):

        processed = preprocessText(post["text"], punctuation, stopwords)

        score = 0

        for word in processed:

            if word in positive:    score += 1

            elif word in negative:  score -= 1

        return {

            "id": post["id"],

            "text": post["text"],

            "processedText": processed,

            "score": score

        }

    return list(map(lambda p: scorePost(p), postsList))

def getFlaggedPosts(scoredPosts, sentimentThreshold=-1):

    return [post for post in scoredPosts if post["score"] <= sentimentThreshold]

def findNegativeTopics(flaggedPosts):

    topics = {}

    for post in flaggedPosts:

        for word in post["processedText"]:

            if word.startswith("#") or word.startswith("@"):

                topics[word] = topics.get(word, 0) + 1

    return topics

scoredPosts = analyzePosts(postsList, PUNCTUATION\_CHARS, STOPWORDS\_SET, POSITIVE\_WORDS, NEGATIVE\_WORDS)

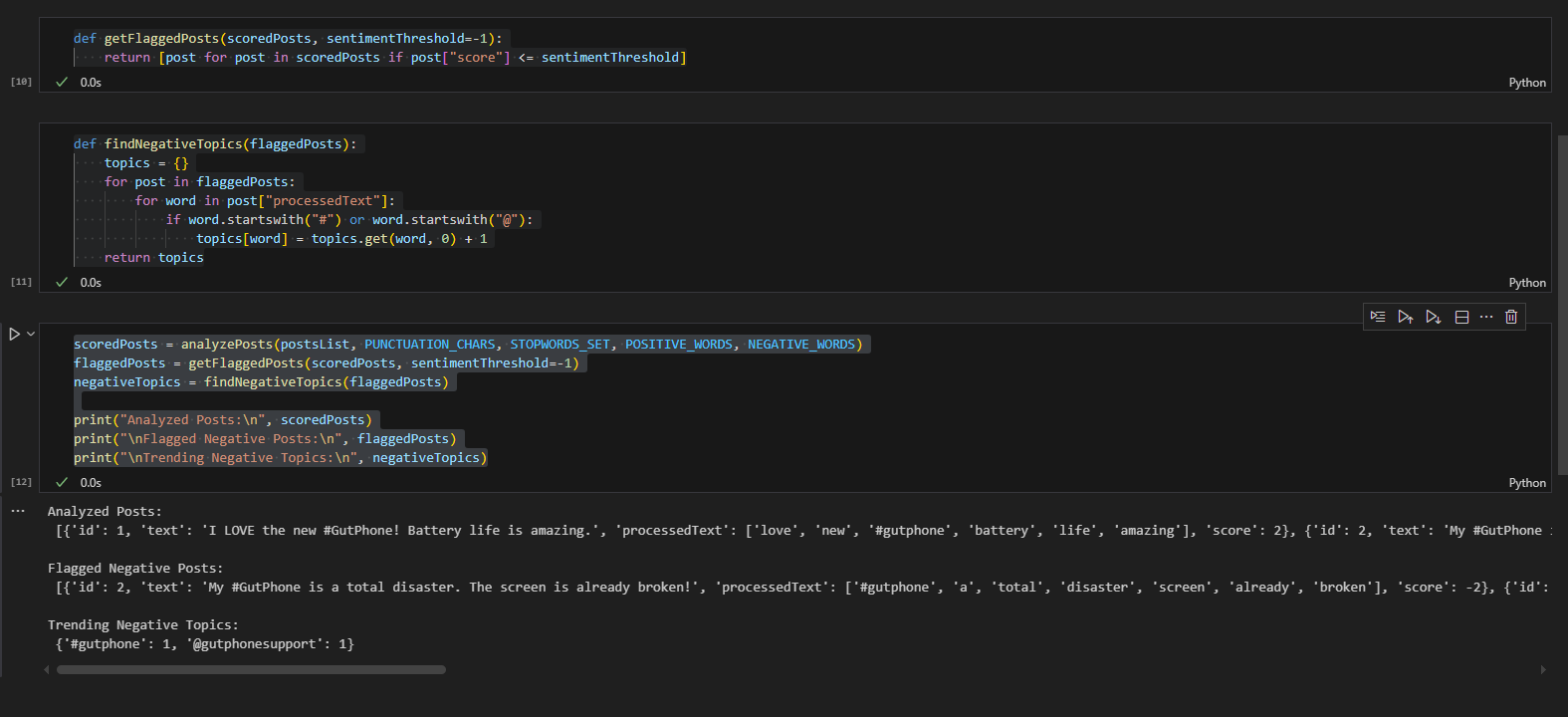
flaggedPosts = getFlaggedPosts(scoredPosts, sentimentThreshold=-1)

negativeTopics = findNegativeTopics(flaggedPosts)

print("Analyzed Posts:\n", scoredPosts)

print("\nFlagged Negative Posts:\n", flaggedPosts)

print("\nTrending Negative Topics:\n", negativeTopics)



## Q3:

class Package:

    def \_\_init\_\_(self,packageId,weight):

        self.packageId=packageId

        self.weightInKg=weight

class Drone:

    def \_\_init\_\_(self,id,maxLoad):

        self.droneId=id

        self.maxLoadInKg=maxLoad

        self.\_\_status="idle"

        self.package=None

        self.timer = 0

    def setStatus(self,newStatus):

        s=newStatus.lower()

        if s in ["idle","delivering","charging"]:

            self.\_\_status=s

        else:

            print("invalid status input")

    def getStatus(self):

        return self.\_\_status

    def assignPackage(self,packageObj):

        if self.getStatus()!="idle":

            print(f"Drone {self.droneId} not idle , can not assign the package {packageObj.packageId}")

            return False

        if packageObj.weightInKg>self.maxLoadInKg:

            print(f"Package too heavy for Drone {self.droneId}")

            return False

        self.package=packageObj

        self.setStatus("delivering")

        self.timer=2

        print(f"Drone {self.droneId} assigned to package {packageObj.packageId}")

        return True

    def update(self):

        if self.getStatus()=="delivering":

            self.timer-=1

            if self.timer==0:

                self.setStatus("charging")

                self.package=None

                self.timer=1

        elif self.getStatus()=="charging":

            self.timer-=1

            if self.timer==0:

                self.setStatus("idle")

class FleetManager:

    def \_\_init\_\_(self):

        self.drones={}

        self.pendingPackages=[]

    def addDrone(self,droneObj):

        self.drones[droneObj.droneId]=droneObj

    def addPackage(self,packageObj):

        self.pendingPackages.append(packageObj)

    def dispatchJobs(self):

        for drone in self.drones.values():

            if drone.getStatus()=="idle" and self.pendingPackages:

                pkg=self.pendingPackages.pop(0)

                success=drone.assignPackage(pkg)

                if not success:

                    self.pendingPackages.insert(0,pkg)

    def simulationTick(self):

        print("-Simulation Ticks-")

        for drone in self.drones.values():

            drone.update()

            print(f"Drone {drone.droneId} -> status: {drone.getStatus()}")

manager = FleetManager()

d1 = Drone("D1", 10)

d2 = Drone("D2", 5)

manager.addDrone(d1)

manager.addDrone(d2)

p1 = Package("P1", 4)

p2 = Package("P2", 6)

p3 = Package("P3", 8)

manager.addPackage(p1)

manager.addPackage(p2)

manager.addPackage(p3)

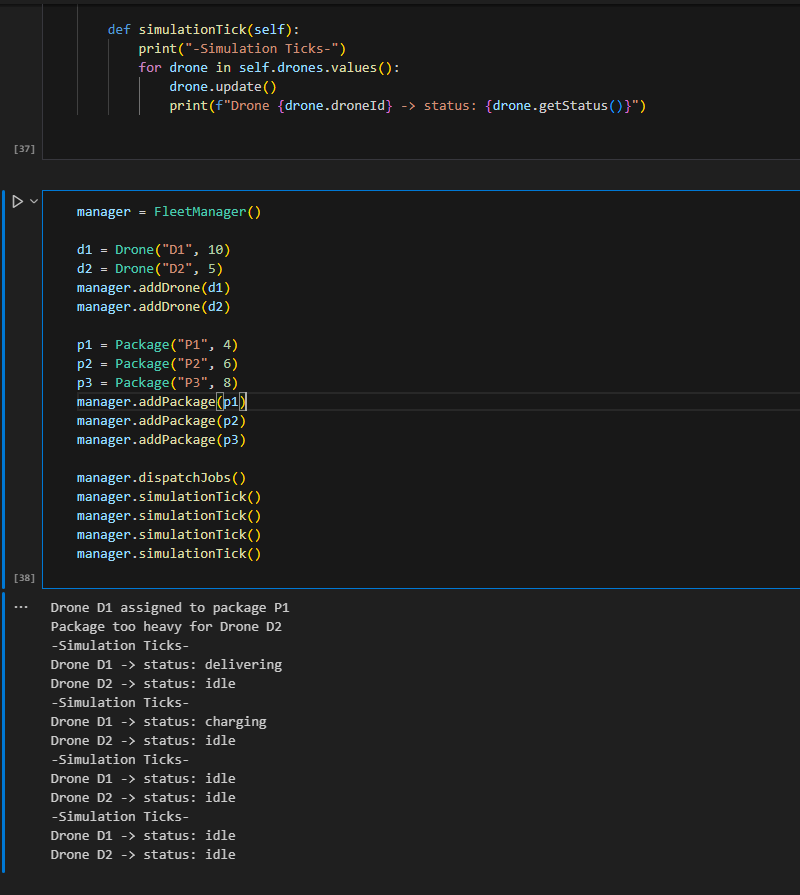
manager.dispatchJobs()

manager.simulationTick()

manager.simulationTick()

manager.simulationTick()

manager.simulationTick()



## Q4:

class Image:

    def \_\_init\_\_(self,pixlel):

        self.pixels=pixlel

    def applyTransformation(self,transformationFunction):

        newPixels=transformationFunction(self.getCopy())

        return Image(newPixels)

    def getCopy(self):

        return [row[:] for row in self.pixels]

def flipHorizontal(pixelData):

    return [row[::-1] for row in pixelData]

def adjustBrightness(pixelData,brightnessValue):

    return  [[pixel+brightnessValue for pixel in row] for row in pixelData]

def rotateNinetyDegree(pixelData):

    return  [list(row) for row in zip(\*pixelData[::-1])]

class AugmentationPipeline:

    def \_\_init\_\_(self):

        self.steps=[]

    def addSteps(self,transformFnc):

        self.steps.append(transformFnc)

    def processImage(self, originalImage):

        augmentatedImage = []

        for fn in self.steps:

            transformed = originalImage.applyTransformation(fn)

            augmentatedImage.append(transformed)

        return augmentatedImage

originalPixels=[

    [10,20,30],

    [40,50,60]

]

img=Image(originalPixels)

pipeline=AugmentationPipeline()

pipeline.addSteps(flipHorizontal)

pipeline.addSteps(lambda data: adjustBrightness(data,10))

pipeline.addSteps(rotateNinetyDegree)

result=pipeline.processImage(img)

for i,result in enumerate(result,start=1):

    print(f"Transformed Tmage {i}: ")

    for row in result.pixels:

        print(row)

