24K0025 – Syed Hanzala Ali – PAI Assignment 01

Q1

```
transactionLog = [
    {'orderId': 1001, 'customerId': 'cust_Ahmed', 'productId': 'prod_10'},
    {'orderId': 1001, 'customerId': 'cust_Ahmed', 'productId': 'prod_12'},
    {'orderId': 1002, 'customerId': 'cust_Bisma', 'productId': 'prod_10'},
    {'orderId': 1002, 'customerId': 'cust_Bisma', 'productId': 'prod_15'},
    {'orderId': 1003, 'customerId': 'cust_Ahmed', 'productId': 'prod_15'},
    {'orderId': 1004, 'customerId': 'cust_Faisal', 'productId': 'prod_10'},
    {'orderId': 1004, 'customerId': 'cust_Faisal', 'productId': 'prod_12'},
]

productCatalog = {
    'prod_10': 'Wireless Mouse',
    'prod_12': 'Keyboard',
    'prod_15': 'USB-C Hub',
}

def processTransaction(transactionList):
    customerData = {}
    for record in transactionList:
        customer = record['customerId']
        product = record['productId']
        if customer not in customerData:
            customerData[customer] = set()
        customerData[customer].add(product)
    return customerData


def findFrequentPairs(customerData):
    pairCount = {}

    for products in customerData.values():
        productList = sorted(list(products))

        for i in range(len(productList)):
            for j in range(i+1 ,len(productList)):
                pair = (productList[i] , productList[j])
                if pair not in pairCount:
                    pairCount[pair] = 0
                pairCount[pair] += 1
```

```
    return pairCount

def getRecommendation(targetProductId , frequentPairs):
    recommendations = {}
    for (p1,p2) , count in frequentPairs.items():
        if targetProductId in (p1,p2):
            other = p2 if p1 == targetProductId else p1
            recommendations[other] = count
    return sorted(recommendations.items() , key= lambda x : x[1] , reverse=True)

def generateReport(targetProductId , recommendations , catalog):
    print(f"\nRecommendations for {catalog[targetProductId]} :\n")

    pID = [prod for prod , count in recommendations]
    counts = [count for prod , count in recommendations]

    combine = zip(pID , counts)

    for i , (prod , count) in enumerate(combine , start=1):
        productName = catalog[prod]
        print(f"{i}. {productName} , bought together {count} times")


def main():
    customerData = processTransaction(transactionLog)
    frequentPairs = findFrequentPairs(customerData)
    recs = getRecommendation('prod_12' , frequentPairs)
    generateReport('prod_12' , recs , productCatalog)

main()
```

Output Q1

```
Recommendations for Keyboard :

1. Wireless Mouse , bought together 2 times
2. USB-C Hub , bought together 1 times
```

Q2

```
PUNCTUATION_CHARS = '!\"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
STOPWORDS_SET =
{'i','me','my','a','an','the','is','am','was','and','but','if','or','to','of','at
','by','for','with','this','that'}
POSITIVE_WORDS_SET = {'love','amazing','great','helpful','resolved'}
NEGATIVE_WORDS_SET = {'disaster','broken','worst','avoid','bad'}

allPosts = [
    {'id': 1, 'text': 'I LOVE the new #GuIPhone! Battery life is amazing.'},
    {'id': 2, 'text': 'My #GuIPhone is a total disaster. The screen is already
broken!'},
    {'id': 3, 'text': 'Worst customer service ever from @GuIPhoneSupport. Avoid
this.'},
    {'id': 4, 'text': 'The @GuIPhoneSupport team was helpful and resolved my
issue. Great service!'}
]

def preprocessText(text, punctuationList, stopwordsSet):
    text = text.lower()
    for ch in punctuationList:
        text = text.replace(ch, '')
    words = text.split()
    clean_words = [word for word in words if word not in stopwordsSet]
    return clean_words

def analyzePosts(postsList, punctuation, stopwords, positive, negative):
    def scoreText(words):
        score = 0
        for w in words:
            if w in positive:
                score += 1
            elif w in negative:
                score -= 1
        return score
    return list(map(lambda post: {
        'id': post['id'],
        'text': post['text'],
        'processedText': preprocessText(post['text'], punctuation, stopwords),
        'score': scoreText(preprocessText(post['text'], punctuation, stopwords))
    }, postsList))

def getFlaggedPosts(posts):
    flagged = list(filter(lambda p: p['score'] < 0, posts))
```

```
    return flagged

def findNegativeTopics(posts):
    hashtags = {}
    mentions = {}
    for post in posts:
        words = post['text'].split()
        for word in words:
            if word.startswith('#'):
                hashtags[word] = hashtags.get(word, 0) + 1
            elif word.startswith('@'):
                mentions[word] = mentions.get(word, 0) + 1
    return {'hashtags': hashtags, 'mentions': mentions}

analyzed = analyzePosts(allPosts, PUNCTUATION_CHARS, STOPWORDS_SET,
POSITIVE_WORDS_SET, NEGATIVE_WORDS_SET)
flagged = getFlaggedPosts(analyzed)
topics = findNegativeTopics(flagged)

print("Analyzed Posts:")
for post in analyzed:
    print(post)

print("\nFlagged (Negative) Posts:")
for post in flagged:
    print(post)

print("\nNegative Topics Found:")
print(topics)
```

Output Q2

```
Analyzed Posts:
{'id': 1, 'text': 'I LOVE the new #GuIPhone! Battery life is amazing.', 'processedText': ['love', 'new', 'guiphone', 'battery',
'life', 'amazing'], 'score': 2}
{'id': 2, 'text': 'My #GuIPhone is a total disaster. The screen is already broken!', 'processedText': ['guiphone', 'total',
'disaster', 'screen', 'already', 'broken'], 'score': -2}
{'id': 3, 'text': 'Worst customer service ever from @GuIPhoneSupport. Avoid this.', 'processedText': ['worst', 'customer',
'service', 'ever', 'from', 'guiphonesupport', 'avoid'], 'score': -2}
{'id': 4, 'text': 'The @GuIPhoneSupport team was helpful and resolved my issue. Great service!', 'processedText':
['guiphonesupport', 'team', 'helpful', 'resolved', 'issue', 'great', 'service'], 'score': 3}

Flagged (Negative) Posts:
{'id': 2, 'text': 'My #GuIPhone is a total disaster. The screen is already broken!', 'processedText': ['guiphone', 'total',
'disaster', 'screen', 'already', 'broken'], 'score': -2}
{'id': 3, 'text': 'Worst customer service ever from @GuIPhoneSupport. Avoid this.', 'processedText': ['worst', 'customer',
'service', 'ever', 'from', 'guiphonesupport', 'avoid'], 'score': -2}

Negative Topics Found:
{'hashtags': {'#GuIPhone': 1}, 'mentions': {'@GuIPhoneSupport.': 1}}
```

Q3

```
class Package:
    def __init__(self, packageId, weightInKg):
        self.packageId = packageId
        self.weightInKg = weightInKg

    def __str__(self):
        return f"Package({self.packageId}, {self.weightInKg}kg)"


class Drone:
    def __init__(self, droneId, maxLoadInKg):
        self.droneId = droneId
        self.maxLoadInKg = maxLoadInKg
        self._status = 'idle'
        self.currentPackage = None
        self.timer = 0

    def getStatus(self):
        return self._status

    def setStatus(self, newStatus):
        allowed = ['idle', 'delivering', 'charging']
        if newStatus in allowed:
            self._status = newStatus
        else:
            print(f"Invalid status '{newStatus}' for drone {self.droneId}")

    def assignPackage(self, packageObj):
        if self._status == 'idle' and packageObj.weightInKg <= self.maxLoadInKg:
            self.currentPackage = packageObj
            self.setStatus('delivering')
            self.timer = 2
            print(f"Drone {self.droneId} started delivering
{packageObj.packageId}")
            return True
        else:
            print(f"Drone {self.droneId} cannot take {packageObj.packageId}")
            return False

    def updateState(self):
        if self._status == 'delivering':
            self.timer -= 1
            if self.timer <= 0:
```

```
                print(f"Drone {self.droneId} finished delivery of
{self.currentPackage.packageId}")
                self.currentPackage = None
                self.setStatus('charging')
                self.timer = 2
        elif self._status == 'charging':
            self.timer -= 1
            if self.timer <= 0:
                print(f"Drone {self.droneId} finished charging and is now idle")
                self.setStatus('idle')


class FleetManager:
    def __init__(self):
        self.drones = {}
        self.pendingPackages = []

    def addDrone(self, drone):
        self.drones[drone.droneId] = drone

    def addPackage(self, package):
        self.pendingPackages.append(package)

    def dispatchJobs(self):
        for drone in self.drones.values():
            if drone.getStatus() == 'idle' and self.pendingPackages:
                package = self.pendingPackages.pop(0)
                success = drone.assignPackage(package)
                if not success:
                    self.pendingPackages.insert(0, package)

    def simulationTick(self):
        print("\n=== Simulation Tick ===")
        for drone in self.drones.values():
            drone.updateState()
        self.dispatchJobs()


if __name__ == "__main__":
    manager = FleetManager()
    manager.addDrone(Drone("D1", 5))
    manager.addDrone(Drone("D2", 10))
    manager.addPackage(Package("P1", 3))
    manager.addPackage(Package("P2", 8))
    manager.addPackage(Package("P3", 4))
```

```
  for tick in range(6):
      print(f"\n--- Tick {tick+1} ---")
      manager.simulationTick()
```

Output Q3

```
--- Tick 1 ---

=== Simulation Tick ===
Drone D1 started delivering P1
Drone D2 started delivering P2

--- Tick 2 ---

=== Simulation Tick ===

--- Tick 3 ---

=== Simulation Tick ===
Drone D1 finished delivery of P1
Drone D2 finished delivery of P2

--- Tick 4 ---

=== Simulation Tick ===

--- Tick 5 ---

=== Simulation Tick ===
Drone D1 finished charging and is now idle
Drone D2 finished charging and is now idle
Drone D1 started delivering P3

--- Tick 6 ---

=== Simulation Tick ===
```

Q4

```
class Image:
    def __init__(self , pixelData):
        self.pixels = pixelData

    def getCopy(self):
        copyPixels = []
        for row in self.pixels:
            newRow = []
            for value in row:
                newRow.append(value)
            copyPixels.append(newRow)
        return copyPixels

    def applyTransformation(self, transformationFunc):
        self.pixels = transformationFunc(self.pixels)

def flipHorizontal(pixelData):
    newPixels = []
    for row in pixelData:
        newRow = []
        for value in reversed(row):
            newRow.append(value)
        newPixels.append(newRow)
    return newPixels

def adjustBrightness(pixelData , brightnessValue):
    newPixels = []
    for row in pixelData:
        newRow = []
        for value in row:
            newRow.append(value + brightnessValue)
        newPixels.append(newRow)
    return newPixels

def rotateNinetyDegrees(pixelData):
    rows = len(pixelData)
    cols = len(pixelData[0])
    newPixels = []
    for i in range(cols):
        newRow = []
        for j in range(rows):
            newRow.append(pixelData[rows-j-1][i])
        newPixels.append(newRow)
```

```
    return newPixels

class AugmentationPipeline:
    def __init__(self):
        self.steps = []

    def addstep(self , transformFunc):
        self.steps.append(transformFunc)

    def processImage(self , originalImage):
        augmentedImages = []
        for func in self.steps:
            newImage = Image(originalImage.getCopy())
            newImage.applyTransformation(func)
            augmentedImages.append(newImage)
        return augmentedImages

if __name__ == "__main__":
    originalPixels = [
        [10,20,30],
        [40,50,60]
    ]

    img = Image(originalPixels)
    pipeline = AugmentationPipeline()
    pipeline.addstep(flipHorizontal)
    pipeline.addstep(lambda pixels : adjustBrightness(pixels,10))
    pipeline.addstep(rotateNinetyDegrees)

    results = pipeline.processImage(img)

    print("Original:")
    for row in img.pixels:
        print(row)

    print("\nAugmented Images:")
    for idx, image in enumerate(results, start=1):
        print(f"Image {idx}:")
        for row in image.pixels:
            print(row)
        print()
```

Output Q4

```
Original:
[10, 20, 30]
[40, 50, 60]

Augmented Images:
Image 1:
[30, 20, 10]
[60, 50, 40]

Image 2:
[20, 30, 40]
[50, 60, 70]

Image 3:
[40, 10]
[50, 20]
[60, 30]
```