PAI ASSIGNMENT # 1

24K-0008

Q1.

CODE:

```
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_12'},
    {'orderId': 1004, 'customerId': 'cust_Faisal', 'productId': 'prod_10'}
]

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

def ProcessTransaction(TransactionList):
    result = {}
    for t in TransactionList:
        cust = t['customerId']
        prod = t['productId']
        if cust not in result:
            result[cust] = set()
        result[cust].add(prod)
    return result

def FindFrequentPair(TransactionList):
    data = {}
    for t in TransactionList:
        cust = t['customerId']
        prod = t['productId']
        if cust not in data:
            data[cust] = set()
        data[cust].add(prod)

    pairCount = {}
    for items in data.values():
        itemsList = list(items)
        for i in range(len(itemsList)):
```

```
            for j in range(i + 1, len(itemsList)):
                a = itemsList[i]
                b = itemsList[j]
                if a > b:
                    a, b = b, a
                pair = (a, b)
                if pair in pairCount:
                    pairCount[pair] += 1
                else:
                    pairCount[pair] = 1
    return pairCount

def GetRecommendations(productId, pairs):
    rec = {}
    for pair, count in pairs.items():
        p1, p2 = pair
        if productId == p1:
            other = p2
        elif productId == p2:
            other = p1
        else:
            continue

        if other in rec:
            rec[other] += count
        else:
            rec[other] = count
    sortedRec = sorted(rec.items(), key=lambda x: x[1], reverse=True)
    return sortedRec

def GetGenerateReport(productId, recList, catalog):
    if not recList:
        print("No co-purchase data available.")
        return
    num = 1
    for pid, count in recList:
        print(num, ".", catalog.get(pid, pid), "(co-purchased", count, "times)")
        num = num + 1

result = ProcessTransaction(transactionLog)
print(result)

pairs = FindFrequentPair(transactionLog)
print(pairs)
```

```
recommend = GetRecommendations('prod_15', pairs)
print("Recommendations for prod_15:", recommend)

sample = [('prod_12', 2), ('prod_15', 2)]
GetGenerateReport('pRod_12', sample, productCatalog)
```

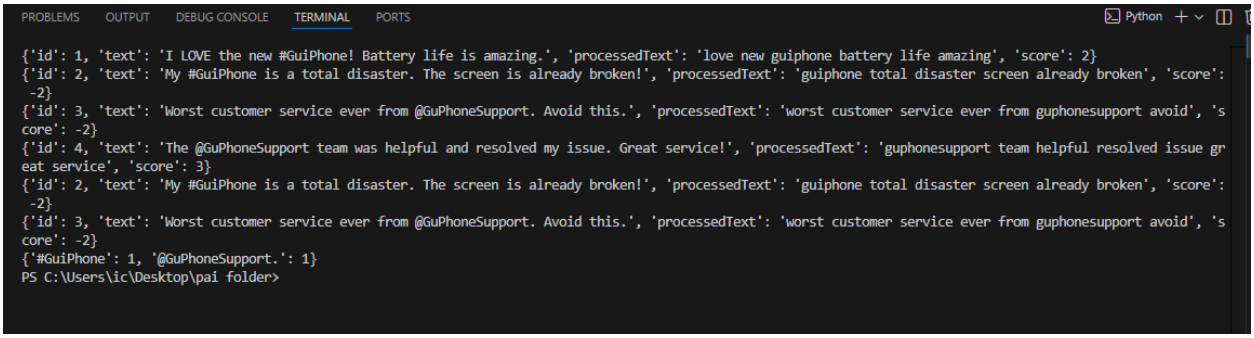
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\ic\Desktop\pai folder> & C:/Users/ic/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/ic/Desktop/pai folder/asad.py"
{'cust_Ahmed': {'prod_10', 'prod_15', 'prod_12'}, 'cust_Bisma': {'prod_10', 'prod_15'}, 'cust_Faisal': {'prod_10', 'prod_12'}}
{('prod_10', 'prod_15'): 2, ('prod_10', 'prod_12'): 2, ('prod_12', 'prod_15'): 1}
Recommendations for prod_15: [('prod_10', 2), ('prod_12', 1)]
1 . Keyboard (co-purchased 2 times)
2 . USB-C Hub (co-purchased 2 times)
PS C:\Users\ic\Desktop\pai folder>
```

Q2.

```
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 @GuPhoneSupport. Avoid
this.'},
    {'id': 4, 'text': 'The @GuPhoneSupport team was helpful and resolved my
issue. Great service!'}
]

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

def PreProcessText(text, punctuationList, stopWordsSet):
    clean_text = ""
    final_text = ""
    text = text.lower()
    for ch in text:
        if ch not in punctuationList:
            clean_text += ch
    words = clean_text.split()
    for w in words:
        if w not in stopWordsSet:
            final_text += w + " "
```

```
    return final_text

def AnalyzePosts(postsList, punctuation, stopwords, positive, negative):
    newList = []
    for post in postsList:
        text = PreProcessText(post['text'], punctuation, stopwords)
        score = 0
        for word in text.split():
            if word in positive:
                score += 1
            elif word in negative:
                score -= 1
        newList.append({
            'id': post['id'],
            'text': post['text'],
            'processedText': text.strip(),
            'score': score
        })
    return newList

def GetFlaggedPosts(posts, sentimentThreshold=-1):
    flagged = []
    for post in posts:
        if post['score'] <= sentimentThreshold:
            flagged.append(post)
    return flagged

def FindNegativePosts(flaggedPosts):
    result = {}
    for post in flaggedPosts:
        text = post['text']
        words = text.split()
        for word in words:
            if "@" in word or "#" in word:
                if word not in result:
                    result[word] = 0
                result[word] += 1
    return result

text = "PAI  is kesy !@ (prhen?"
res = PreProcessText(text, PUNCTUATION_CHARS, STOPWORDS_SET)
print(res)

posts = AnalyzePosts(allPosts, PUNCTUATION_CHARS, STOPWORDS_SET,
POSITIVE_WORDS_SET, NEGATIVE_WORDS_SET)
```

```
for p in posts:
    print(p)

flagged = GetFlaggedPosts(posts)
for f in flagged:
    print(f)


negatives = FindNegativePosts(flagged)
print(negatives)
```



Q3.

check_status = ['idle', 'delivering', 'charging']

class Package:

    def __init__(self, packageId, weightInKg):

        self.packageId = packageId

        self.weightInKg = weightInKg

class Drone:

    def __init__(self, droneId, maxLoadInKg, status):

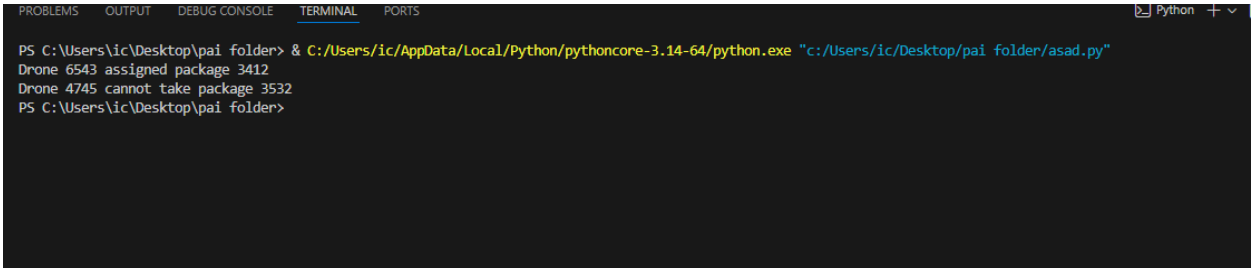
        self.droneId = droneId

        self.maxLoadInKg = maxLoadInKg

```
        self.__status = status
        self.assigned_package = None
        self.delivery_timer = 0

    def get_status(self):
        return self.__status

    def set_status(self, newStatus):
        if newStatus.lower() not in check_status:
            print("Invalid status")
            return
        self.__status = newStatus.lower()

    def assign_package(self, packageObj):
        if self.__status == 'idle' and packageObj.weightInKg <= self.maxLoadInKg:
            self.assigned_package = packageObj
            self.set_status('delivering')
            self.delivery_timer = 2
            print("Drone", self.droneId, "assigned package", packageObj.packageId)
            return True
        else:
            print("Drone", self.droneId, "cannot take package", packageObj.packageId)
            return False


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

```
        self.current_pkg_index = 0

    def add_drone(self, droneId, maxLoadInKg, status):
        index = len(self.drones)
        self.drones[index] = Drone(droneId, maxLoadInKg, status)

    def add_package(self, packageObj):
        self.pending_packages.append(packageObj)

    def dispatchJobs(self):
        for drone in self.drones.values():
            if drone.get_status() == 'idle' and self.current_pkg_index < len(self.pending_packages):
                pkg = self.pending_packages[self.current_pkg_index]
                if drone.assign_package(pkg):
                    self.current_pkg_index += 1

    def simulationTick(self):
        for drone in self.drones.values():
            if drone.get_status() == 'delivering':
                drone.delivery_timer -= 1
                if drone.delivery_timer <= 0:
                    drone.assigned_package = None
                    drone.set_status('charging')
            elif drone.get_status() == 'charging':
                drone.set_status('idle')



packages = [
    Package(3412, 10),
```

```
    Package(3532, 6.4),
    Package(4523, 2.5),
    Package(3112, 7.3),
    Package(6456, 8)
]

fm = FleetManager()
fm.add_drone(6543, 10, 'idle')
fm.add_drone(9041, 5, 'charging')
fm.add_drone(4523, 8, 'delivering')
fm.add_drone(4745, 6, 'idle')

for pkg in packages:
    fm.add_package(pkg)

fm.dispatchJobs()
fm.simulationTick()
```



Q4

```
originalPixels = [[10, 20, 30], [40, 50, 60]]

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

```
    def applyTransformation(self, func):
        self.pixels = func(self.pixels)

    def getCopy(self):
        return [row[:] for row in self.pixels]


def flipHorizontal(pixels):
    return [row[::-1] for row in pixels]


def adjustBrightness(pixels, value):
    for i in range(len(pixels)):
        for j in range(len(pixels[0])):
            pixels[i][j] += value
    return pixels


def rotateNinetyDegrees(pixels):
    rows = len(pixels)
    cols = len(pixels[0])
    rotated = []
    for i in range(cols):
        new_row = []
        for j in range(rows):
            new_row.append(pixels[rows - j - 1][i])
        rotated.append(new_row)
    return rotated
```

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

    def addStep(self, func):
        self.steps.append(func)

    def processImage(self, image):
        result = []
        for step in self.steps:
            temp = Image(image.getCopy())
            temp.applyTransformation(step)
            result.append(temp.pixels)
        return result


img = Image(originalPixels)
pipeline = AugmentationPipeline()

pipeline.addStep(flipHorizontal)
pipeline.addStep(lambda p: adjustBrightness(p, 10))
pipeline.addStep(rotateNinetyDegrees)

augmentedImages = pipeline.processImage(img)

for i in range(len(augmentedImages)):
    print("Transformation", i + 1)
```

```
for row in augmentedImages[i]:
    print(row)
print()
```