PAI ASSIGNMENT 1

Question 1:

```
# The raw log of all items in all orders
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'},
]
# The mapping of product IDs to names
productCatalog = {
'prod_10': 'Wireless Mouse',
'prod_12': 'Keyboard',
'prod_15': 'USB-C Hub',
}
def processTransactions(transactionsList):
  purchasedProductsCustomers = {}
  for details in transactionsList:
    customer = details['customerId']
    product = details['productId']
    if customer not in purchasedProductsCustomers:
      purchasedProductsCustomers[customer] = set()
  purchasedProductsCustomers[customer].add(product)
```

```
    return purchasedProductsCustomers
def findFrequentPairs(transactionLog):
    pairCounts = {}
    productPairList = []
    productPair = []
    previousCustomer = None
    for details in transactionLog:
        customerId = details['customerId']
        productId = details['productId']
        if previousCustomer is None or customerId == previousCustomer:
            productPair.append(productId)
        else:
            if len(productPair) > 1:
                productPairList.append(productPair)
            productPair = [productId]
        previousCustomer = customerId
    if(len(productPair)>1):
        productPairList.append(productPair)
    for products in productPairList:
        for i in range(len(products)):
            for j in range(i+1,len(products)):
                pair = tuple(sorted((products[i],products[j])))
                pairCounts[pair] = pairCounts.get(pair,0) + 1
    print(pairCounts)
    return pairCounts
def getRecommendations(targetProductId, frequentPairs):
```

```
    recommendations = {}
    for pair, count in frequentPairs.items():
        if targetProductId in pair:
            other = pair[0] if pair[1] == targetProductId else pair[1]
            recommendations[other] = count
    return sorted(recommendations.items(), key=lambda x: x[1], reverse=True)
def generateReport(targetProductId, recommendations, catalog):
    for index, (product, count) in enumerate(recommendations, start=1):
        print(index, catalog[product], count)
customerData = processTransactions(transactionLog)
pairs = findFrequentPairs(transactionLog)
recs = getRecommendations('prod_10', pairs)
generateReport('prod_10', recs, productCatalog)
```

Output:

```
{('prod_10', 'prod_12'): 2, ('prod_10', 'prod_15'): 1}
1 Keyboard 2
2 USB-C Hub 1
```

Question 2:

```
def preprocessText(text, punctuationList, stopwordsSet):
    text = text.lower()
    for character in punctuationList:
        if character not in ['#', '@']:
            text = text.replace(character, '')
    words = text.split()
    cleanedWords = [word for word in words if word not in stopwordsSet]
    return cleanedWords
def analyzePosts(postsList, punctuation, stopwords, positive, negative):
    scoredPosts = []
    for post in postsList:
        processed = preprocessText(post['text'], punctuation, stopwords)
        score = 0
        for word in processed:
            if word in positive:
                score += 1
            elif word in negative:
                score -= 1
        scoredPosts.append({
        'id': post['id'],
        'text': post['text'],
        'processedText': processed,
        'score': score
        })
    return scoredPosts
```

```
def getFlaggedPosts(scoredPosts,sentimentThreshold=-1):
  return [post for post in scoredPosts if post['score']<=sentimentThreshold]
def findNegativeTopics(flaggedPosts):
  topicCounts = {}
  for post in flaggedPosts:
    processedPost = post['processedText']
    for word in processedPost:
      if word.startswith('#') or word.startswith('@'):
        topicCounts[word] = topicCounts.get(word, 0) + 1
  return topicCounts
allPosts = [
  {'id': 1, 'text': 'I LOVE the new #GulPhone! Battery life is amazing.'},
  {'id': 2, 'text': 'My #GulPhone is a total disaster. The screen is already broken!'},
  {'id': 3, 'text': 'Worst customer service ever from @GulPhoneSupport. Avoid this.'},
  {'id': 4, 'text': 'The @GulPhoneSupport 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', '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'}
scoredPosts = analyzePosts(allPosts, PUNCTUATION_CHARS, STOPWORDS_SET, POSITIVE_WORDS_SET, NEGATIVE_WORDS_SET)
print("Scored Posts:")
for post in scoredPosts:
  print(post)
```

```
flaggedPosts = getFlaggedPosts(scoredPosts,sentimentThreshold=-1)
print("\nFlagged Negative Posts:")
for post in flaggedPosts:
    print(post)
negativeTopics = findNegativeTopics(flaggedPosts)
print("\nNegative Topics Frequency:")
print(negativeTopics)
```

Output:

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

Flagged Negative Posts:
{'id': 2, 'text': 'My #GulPhone is a total disaster. The screen is already broken!', 'processedText': ['#gulphone', 'total', 'disaster', 'screen', 'already', 'broken'], 'score': -2}
{'id': 3, 'text': 'Worst customer service ever from @GulPhoneSupport. Avoid this.', 'processedText': ['worst', 'customer', 'service', 'ever', 'from', '@gulphonesupport', 'avoid'], 'score': -2}

Negative Topics Frequency:
{'#gulphone': 1, '@gulphonesupport': 1}
```

Question 3:

```
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
        status = status.lower()
        self.__status = status
        self.tickCount = 0
        self.currentPackage = None
    def getStatus(self):
        return self.__status
    def setStatus(self,newStatus):
        newStatus = newStatus.lower()
        if(newStatus=='idle' or newStatus=='delivering' or newStatus=='charging'):
            self.__status = newStatus
        else:
            print("Invalid status provided!!!")
    def assignPackage(self,packageObj):
        if(self.__status=='idle' and packageObj.weightInKg<=self.maxLoadInKg):
            self.currentPackage = packageObj
            return True
        else:
```

```
        print("Package can't be assigned due to drone not available or package weight greater than drone's maximum capacity!!!")
        return False
class FleetManager:
    def __init__(self,listOfDrones,listOfPackages):
        self.listOfPackages = listOfPackages
        self.drones = {}
        for drone in listOfDrones:
            self.drones[drone.droneId] = drone
    def dispatchJobs(self):
        for package in self.listOfPackages[:]:
            for drone in self.drones.values():
                if(drone.getStatus()=='idle'):
                    if drone.assignPackage(package):
                        drone.setStatus('delivering')
                        self.listOfPackages.remove(package)
                        break
    def simulationTick(self):
        for drone in self.drones.values():
            drone.tickCount += 1
            if(drone.getStatus()=='delivering' and drone.tickCount==2):
                drone.setStatus('charging')
                drone.tickCount = 0
            elif(drone.getStatus()=='charging' and drone.tickCount==2):
                drone.setStatus('idle')
                drone.tickCount = 0
```

```
p1 = Package('PKG001',3)

p2 = Package('PKG002',6)

p3 = Package('PKG003',2)

p4 = Package('PKG004',12)

d1 = Drone('DRN001',5,'idle')

d2 = Drone('DRN002',10,'idle')

d3 = Drone('DRN003',5,'idle')

fleet = FleetManager([d1,d2,d3],[p1,p2,p3,p4])

print("--- Dispatching Jobs ---")

fleet.dispatchJobs()

d1.setStatus('flying')

for i in range(1, 7):

  print(f"\n--- Simulation Tick {i} ---")

  fleet.simulationTick()

  for drone in fleet.drones.values():

    print(f"Drone {drone.droneId}: Status = {drone.getStatus()}, TickCount =
{drone.tickCount}")
```

Output:

```
--- Dispatching Jobs ---
Invalid status provided!!!

--- Simulation Tick 1 ---
Drone DRN001: Status = delivering, TickCount = 1
Drone DRN002: Status = delivering, TickCount = 1
Drone DRN003: Status = delivering, TickCount = 1

--- Simulation Tick 2 ---
Drone DRN001: Status = charging, TickCount = 0
Drone DRN002: Status = charging, TickCount = 0
Drone DRN003: Status = charging, TickCount = 0

--- Simulation Tick 3 ---
Drone DRN001: Status = charging, TickCount = 1
Drone DRN002: Status = charging, TickCount = 1
Drone DRN003: Status = charging, TickCount = 1

--- Simulation Tick 4 ---
Drone DRN001: Status = idle, TickCount = 0
Drone DRN002: Status = idle, TickCount = 0
Drone DRN003: Status = idle, TickCount = 0

--- Simulation Tick 5 ---
Drone DRN001: Status = idle, TickCount = 1
Drone DRN002: Status = idle, TickCount = 1
Drone DRN003: Status = idle, TickCount = 1
```

```
--- Simulation Tick 6 ---
Drone DRN001: Status = idle, TickCount = 2
Drone DRN002: Status = idle, TickCount = 2
Drone DRN003: Status = idle, TickCount = 2
```

Question 4:

```
class Image:
    def __init__(self,originalPixels):
        self.originalPixels = originalPixels
    def applyTransformation(self,transformationFunc):
        self.originalPixels = transformationFunc(self.originalPixels)
    def getCopy(self):
        copyPixels = [row[:] for row in self.originalPixels]
        return copyPixels
def flipHorizontal(pixelData):
    flippedPixels = []
    for row in pixelData:
        temp = []
        for i in range(0,len(row)):
            temp.append(row[len(row)-i-1])
        flippedPixels.append(temp)
    return flippedPixels
def adjustBrightness(pixelData,brightnessValue):
    adjustedBrightness = []
    for row in pixelData:
        temp = []
        for value in row:
            value += brightnessValue
            temp.append(value)
        adjustedBrightness.append(temp)
    return adjustedBrightness
```

```
def rotateNinetyDegrees(pixelData):
    rotatedNientyDegree = []
    for i in range(0,len(pixelData[0])):
        temp = []
        for j in range(len(pixelData)-1,-1,-1):
            temp.append(pixelData[j][i])
        rotatedNientyDegree.append(temp)
    return rotatedNientyDegree
class AugmentationPipeline:
    def __init__(self):
        self.functions = []
    def addStep(self,transformFunc):
        self.functions.append(transformFunc)
    def processImage(self,originalImage):
        transformedImages = []
        for func in self.functions:
            imgCopy = Image(originalImage.getCopy())
            imgCopy.applyTransformation(func)
            transformedImages.append(imgCopy.originalPixels)
        return transformedImages
originalPixels = [
    [10, 20, 30],
    [40, 50, 60]
]
img = Image(originalPixels)
pipeline = AugmentationPipeline()
```

```
pipeline.addStep(flipHorizontal)
pipeline.addStep(lambda pixels: adjustBrightness(pixels,20))
pipeline.addStep(rotateNinetyDegrees)
results = pipeline.processImage(img)
print("Original Image:")
for row in originalPixels:
    print(row)
print("\nTransformed Images:")
for i, transformed in enumerate(results, start=1):
    print(f"\nTransformation {i}:")
    for row in transformed:
        print(row)
```

Output:

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

Transformed Images:

Transformation 1:
[30, 20, 10]
[60, 50, 40]

Transformation 2:
[30, 40, 50]
[60, 70, 80]

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