Shahzaib Zaheer, Assignment 1
BAI-3A, 24K-0040

Note: all these task are done in jupyter notebook

Task 1 Code:

```
import numpy, json
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 processTransactions(transactionslist):
    data = {}
    for t in transactionslist:
        customer_id = t['customerId']
        product_id = t['productId']
        if customer_id not in data:
            data[customer_id] = set()
        data[customer_id].add(product_id)
    return data
print("Processing transactions")
data = processTransactions(transactionLog)
print("Customer data:", data)
def findFrequentPairs(custData):
    freqs = {}
    for CustID, product in custData.items():
        allproducts = list(product)
        allproducts.sort()

        for i in range(len(allproducts)):
            for j in range(1, len(allproducts)):

                sets = (allproducts[i], allproducts[j])

                if sets in freqs:
                    freqs[sets] += 1
```

Shahzaib Zaheer, Assignment 1
BAI-3A, 24K-0040

```
            else:
                freqs[sets] = 1
    return freqs

print("Finding frequent pairs")
pairs = findFrequentPairs(data)
print("Frequent pairs:", pairs)
def getRecommendation(targetId, freq):
    recommends = []
    for p in freq:
        count = freq[p]
        if targetId == p[0]:
            other = p[1]
            recommends.append((other, count))
        elif targetId == p[1]:
            other = p[0]
            recommends.append((other, count))

    recommends.sort()
    recommends.reverse()
    return recommends
def generateReport(target_id, recommendations, catalog):
    print("Report")

    target_name = catalog.get(target_id, "Unknown Product")
    print(f"Target Product: {target_id} ({target_name})")

    if not recommendations:
        print("No recommendations found.")
        return
    size = len(recommendations)
    print(f"Number of recommendations: {size}\n")
    print("Ranked Recommendations:\n")

    for i, (productId, count) in enumerate(recommendations, 1):
        name = catalog.get(productId, "Unknown Product")
        print(f"{i}. {productId} ({name}) - Co-purchased {count} time(s)")

testing = ['prod_10', 'prod_12', 'prod_15']

for product in testing:
    print("")
    recommendations = getRecommendation(product, pairs)
```

Shahzaib Zaheer, Assignment 1
BAI-3A, 24K-0040

generateReport(product, recommendations, productCatalog)

Output:

```
Report
Target Product: prod_10 (Wireless Mouse)
Number of recommendations: 2

Ranked Recommendations:

1. prod_15 (USB-C Hub) - Co-purchased 2 time(s)
2. prod_12 (Keyboard) - Co-purchased 2 time(s)

Report
Target Product: prod_12 (Keyboard)
Number of recommendations: 4

Ranked Recommendations:

1. prod_15 (USB-C Hub) - Co-purchased 1 time(s)
2. prod_15 (USB-C Hub) - Co-purchased 1 time(s)
3. prod_12 (Keyboard) - Co-purchased 2 time(s)
4. prod_10 (Wireless Mouse) - Co-purchased 2 time(s)

Report
Target Product: prod_15 (USB-C Hub)
Number of recommendations: 4

Ranked Recommendations:

1. prod_15 (USB-C Hub) - Co-purchased 2 time(s)
2. prod_12 (Keyboard) - Co-purchased 1 time(s)
3. prod_12 (Keyboard) - Co-purchased 1 time(s)
4. prod_10 (Wireless Mouse) - Co-purchased 2 time(s)
```

Task2 code:

```
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', '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'}
```

Shahzaib Zaheer, Assignment 1
BAI-3A, 24K-0040

```
def preProcess(punc, stops, txt):
    txt = txt.lower()

    for ch in punc:
        txt = txt.replace(ch, ' ')

    words = txt.split()

    clean = []
    for w in words:
        if w not in stops:
            clean.append(w)

    return clean

print("Testing cleanText function:")
txt = "I LOVE the new #GuiPhone! Battery life is amazing."
result = preProcess(PUNCTUATION_CHARS, STOPWORDS_SET, txt)
print("Original:", txt)
print("Cleaned:", result)

def analyzePosts(posts, punc, stops, pos, neg):
    def process_post(p):
        words = preProcess(punc, stops, p['text'])

        score = 0
        for w in words:
            if w in pos:
                score += 1
            elif w in neg:
                score -= 1

        return {
            'id': p['id'],
            'text': p['text'],
            'processedText': words,
            'score': score
        }
```

```
    result = list(map(lambda p: process_post(p), posts))
    return result

print("\nAnalyzing all posts:")
scored_posts = analyzePosts(allPosts, PUNCTUATION_CHARS, STOPWORDS_SET,
POSITIVE_WORDS_SET, NEGATIVE_WORDS_SET)

for post in scored_posts:
    print(f"ID: {post['id']}, Score: {post['score']}, Text: {post['text']}")
    print(f"Processed: {post['processedText']}\n")
def getFlaggedPosts(scoredPosts, sentimentThreshold=-1):
    return [post for post in scoredPosts if post['score'] <= sentimentThreshold]

#testing for above func
print("Getting flagged posts:")
flagged_posts = getFlaggedPosts(scored_posts, -1)

print(f"Found {len(flagged_posts)} negative posts:")
for post in flagged_posts:
    print(f"ID: {post['id']}, Score: {post['score']}, Text: {post['text']}")
def findNegativeTopics(flaggedPosts):
    topic_counts = {}

    for post in flaggedPosts:
        words = post['text'].split()

        for word in words:
            if word.startswith('#') or word.startswith('@'):
                if word in topic_counts:
                    topic_counts[word] += 1
                else:
                    topic_counts[word] = 1

    return topic_counts
#testing
print("Finding negative words:")
negTopics = findNegativeTopics(flagged_posts)

print("Negative words and their frequencies:")
```

Shahzaib Zaheer, Assignment 1
BAI-3A, 24K-0040

```
for topic, count in negTopics.items():
    print(f"{topic}: {count} times")



print("1. Preprocessing test:")
text = "Hello! This is a test #message with @mentions and punctuation!"
cleaned = preProcess(PUNCTUATION_CHARS, STOPWORDS_SET, text)  # Fixed
parameter order
print(f"Original: {text}")
print(f"Cleaned: {cleaned}")

print("\n2. Analyzing all posts:")
allscores = analyzePosts(allPosts, PUNCTUATION_CHARS, STOPWORDS_SET,
POSITIVE_WORDS_SET, NEGATIVE_WORDS_SET)

for post in allscores:
    print(f"Post {post['id']}: Score = {post['score']}")

print("\n3. Flagging negative posts:")
negativePost = getFlaggedPosts(allscores, -1)
print(f"Found {len(negativePost)} negative posts:")

for post in negativePost:
    print(f"Post {post['id']}: '{post['text']}' (Score: {post['score']})")

print("\n4. Negative topics analysis:")
topics = findNegativeTopics(negativePost)
if topics:
    print("Negative topics found:")
    for topic, count in topics.items():
        print(f"{topic}: appears {count} times")
else:
    print("No negative topics found.")
```

Shahzaib Zaheer, Assignment 1
BAI-3A, 24K-0040

Output:

```
1. Preprocessing test:
Original: Hello! This is a test #message with @mentions and punctuation!
Cleaned: ['hello', 'test', 'message', 'mentions', 'punctuation']

2. Analyzing all posts:
Post 1: Score = 2
Post 2: Score = -2
Post 3: Score = -2
Post 4: Score = 3

3. Flagging negative posts:
Found 2 negative posts:
Post 2: 'My #GuiPhone is a total disaster. The screen is already broken!' (Score: -2)
Post 3: 'Worst customer service ever from @GuPhoneSupport. Avoid this.' (Score: -2)

4. Negative topics analysis:
Negative topics found:
#GuiPhone: appeared 1 times
@GuPhoneSupport.: appeared 1 times
```

Task 3 Code:

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


    def getStatus(self):
        return self._status

    def setStatus(self, newStatus):
        valid = ['idle', 'delivering', 'charging']
        if newStatus in valid:
            self._status = newStatus
```

Shahzaib Zaheer, Assignment 1
BAI-3A, 24K-0040

```
        else:
            print(f"Error: Status must be one of {valid}")

    def assignPackage(self, packageObj):
        if self._status != 'idle':
            print(f"Drone {self.droneId} is not idle")
            return False

        if packageObj.weightInKg > self.maxLoadInKg:
            print(f"Package too heavy for drone {self.droneId}")
            return False

        self.currentPackage = packageObj
        self.setStatus('delivering')
        self.timer = 2
        print(f"Package {packageObj.packageId} assigned to drone {self.droneId}")
        return True

    def update(self):
        if self._status == 'delivering':
            self.timer -= 1
            if self.timer <= 0:
                print(f"Drone {self.droneId} completed delivery")
                self.currentPackage = None
                self.setStatus('charging')

        elif self._status == 'charging':
            self.setStatus('idle')
            print(f"Drone {self.droneId} is now idle")
print("Testing Package and Drone classes:")

p1 = Package("PKG001", 2.5)
p2 = Package("PKG002", 1.8)
p3 = Package("PKG003", 3.0)

print(f"Created package: {p1.packageId}, weight: {p1.weightInKg}kg")

d1 = Drone("DRONE1", 3.0)
d2 = Drone("DRONE2", 2.0)
```

Shahzaib Zaheer, Assignment 1
BAI-3A, 24K-0040

```
print(f"Created drone: {d1.droneId}, max load: {d1.maxLoadInKg}kg")

print(f"Drone status: {d1.getStatus()}")
d1.setStatus('delivering')
print(f"updated status: {d1.getStatus()}")

d1.setStatus('idle')
print(f"updated status: {d1.getStatus()}")
print("Testing:")

d1.setStatus('idle')
result = d1.assignPackage(p1)
print(f"Assignment result: {result}")

d1.setStatus('delivering')
result = d1.assignPackage(p2)
print(f"when busy: {result}")

d3 = Drone("DRONE3", 1)
d3.setStatus('idle')
result = d3.assignPackage(p1)
print(f"Assignig heavy packages: {result}")
class FleetManager:
    def __init__(self):
        self.pendingPackages = []
        self.drones = {}

    def addDrone(self, drone):
        self.drones[drone.droneId] = drone
        print(f"Added drone {drone.droneId}")

    def addPackage(self, package):
        self.pendingPackages.append(package)
        print(f"Added package {package.packageId}")

    def dispatchJobs(self):
        print("Dispatching jobs")
        assigned = 0
```

Shahzaib Zaheer, Assignment 1
BAI-3A, 24K-0040

```
        for package in self.pendingPackages[:]:
            for drone in self.drones.values():
                if drone.getStatus() == 'idle':
                    if drone.assignPackage(package):
                        self.pendingPackages.remove(package)
                        assigned += 1
                        break

        print(f"Assigned {assigned} packages")

    def simulationTick(self):
        print("Simulation tick")
        for drone in self.drones.values():
            drone.update()

print("Thsi is main func")

manager = FleetManager()

manager.addDrone(Drone("D1", 2.5))
manager.addDrone(Drone("D2", 3.0))

manager.addPackage(Package("P1", 2.0))
manager.addPackage(Package("P2", 1.5))
manager.addPackage(Package("P3", 2.8))

manager.dispatchJobs()

for i in range(3):
    manager.simulationTick()
    if i < 2:
        manager.dispatchJobs()
```

Output:

Shahzaib Zaheer, Assignment 1
BAI-3A, 24K-0040

```
Thsi is main func
Added drone D1
Added drone D2
Added package P1
Added package P2
Added package P3
Dispatching jobs
Package P1 assigned to drone D1
Package P2 assigned to drone D2
Assigned 2 packages
Simulation tick
Dispatching jobs
Assigned 0 packages
Simulation tick
Drone D1 completed delivery
Drone D2 completed delivery
Dispatching jobs
Assigned 0 packages
Simulation tick
Drone D1 is now idle
Drone D2 is now idle
```

Task 4 Code:

```
originalPixels = [
    [10, 20, 30],
    [40, 50, 60]
]
class Image:
    def __init__(self, pixels):
        self.pixels = pixels

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

    def getCopy(self):
        copy = []
        for row in self.pixels:
```

```
            newRow = []
            for val in row:
                newRow.append(val)
            copy.append(newRow)
        return Image(copy)
def flipHorizontal(data):
    newdata = []
    for row in data:
        newrow = row[::-1]
        newdata.append(newrow)
    return newdata

def adjustBrightness(data, brightval):
    newdata = []
    for row in data:
        newrow = []
        for val in row:
            newval = val + brightval
            if newval < 0:
                newval = 0
            newrow.append(newval)
        newdata.append(newrow)
    return newdata

def rotateNinetyDegrees(data):
    rows = len(data)
    cols = len(data[0])
    newdata = []
    for j in range(cols):
        newrow = []
        for i in range(rows-1, -1, -1):
            newrow.append(data[i][j])
        newdata.append(newrow)
    return newdata
class AugmentationPipeline:
    def __init__(self):
        self.steps = []

    def addStep(self, func):
```

```
        self.steps.append(func)

    def processImage(self, img):
        results = []
        for step in self.steps:
            copy = img.getCopy()
            copy.applyTransformation(step)
            results.append(copy)
        return results
img = Image(originalPixels)

pipe = AugmentationPipeline()
pipe.addStep(flipHorizontal)
pipe.addStep(lambda data: adjustBrightness(data, 10))
pipe.addStep(rotateNinetyDegrees)

results = pipe.processImage(img)

print("final output:")
for i, result in enumerate(results):
    print(f"Image {i+1}:")
    for row in result.pixels:
        print(row)
```

Output:

```
final output:
Image 1:
[30, 20, 10]
[60, 50, 40]
Image 2:
[20, 30, 40]
[50, 60, 70]
Image 3:
[40, 10]
[50, 20]
[60, 30]
```