PAI ASSIGNMENT 1

Task#1:

```
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):
    custPurchase = {}
    for t in transactionsList:
        cust = t['customerId']
        prod = t['productId']
        if cust not in custPurchase:
            custPurchase[cust] = set()
```

```python
        custPurchase[cust].add(prod)
    return custPurchase


def findFrequentPairs(custPurchase):
    pairCounter = {}
    for p in custPurchase.values():
        p = list(p)
        for i in range(len(p)):
            for j in range(i + 1, len(p)):
                p1, p2 = p[i], p[j]
                if p1 > p2:
                    p1, p2 = p2, p1
                pair = (p1, p2)
                if pair not in pairCounter:
                    pairCounter[pair] = 0
                pairCounter[pair] += 1
    return pairCounter


def getRecommendations(targetProductId, frequentPairs):
    r = []
    for pair, count in frequentPairs.items():
        p1, p2 = pair
        if targetProductId == p1 or targetProductId == p2:
            other = p2 if targetProductId == p1 else p1
            r.append((other, count))

    for i in range(len(r)):
        for j in range(i + 1, len(r)):
            if r[j][1] > r[i][1]:
                r[i], r[j] = r[j], r[i]

    return r
```

```python
def generateReport(targetProductId, recommendations,
catalog):
    report = "Recommendations for '" +
catalog[targetProductId] + "':\n"
    index = 1
    for prodId, count in recommendations:
        report += str(index) + ", " + catalog[prodId] + " – "
+ str(count) + "\n"
        index += 1
    return report

custPurchase = processTransactions(transactionLog)
frequentPairs = findFrequentPairs(custPurchase)
recommendations = getRecommendations('prod_10',
frequentPairs)
print(generateReport('prod_10', recommendations,
productCatalog))
```

Output:

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

Tast#2:

```python
def prepocessText(data, markList, stopDictionary):
  data = data.lower()
  for ch in markList:
    if ch not in ["#", "@"]:
      data = data.replace(ch, "")
  tokens = data.split()
  tokens = [token for token in tokens if token not in stopDictionary]
  return tokens

def analyzePosts(postList, marks, stopWords, positiveWords,
negativeWords):
  evaluated = []
  for post in postList:
    filtered = prepocessText(post['text'], marks, stopWords)
    value = 0
    for word in filtered:
      if word in positiveWords:
        value += 1
      elif word in negativeWords:
        value -= 1
    evaluated.append({'postId':post['id'], 'content': post['text'],
'cleanTokens': filtered, 'sentiment': value})
  return evaluated

def extractLowScore(reviewedPosts, minSentiment = -1):
  return [p for p in reviewedPosts if p['sentiment'] <= minSentiment]

def findBadWords(lowPosts):
  topicDict = {}
  for post in lowPosts:
    for word in post['cleanTokens']:
      if word.startswith("#") or word.startswith("@"):
          topicDict[word] = topicDict.get(word, 0) + 1
  return topicDict

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'}
```

```
reviews = analyzePosts(allPosts, PUNCTUATION_CHARS, STOPWORDS_SET,
POSITIVE_WORDS_SET, NEGATIVE_WORDS_SET)
print("Scored Posts:")
for r in reviews:
  print(r)

lowRated = extractLowScore(reviews)
print("\nLow Rated Posts:")
for r in lowRated:
  print(r)

topics = findBadWords(lowRated)
print("\nNegative Topics:")
print(topics)
```

Output:

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

Low Rated Posts:
{'postId': 2, 'content': 'My #GulPhone is a total disaster. The screen is
already broken!', 'cleanTokens': ['#gulphone', 'total', 'disaster',
'screen', 'already', 'broken'], 'sentiment': -2}
{'postId': 3, 'content': 'Worst customer service ever from
@GulPhoneSupport. Avoid this.', 'cleanTokens': ['worst', 'customer',
'service', 'ever', 'from', '@gulphonesupport', 'avoid'], 'sentiment': -2}

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

Task#3:

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

class Drone:
  def __init__(self,unitID, limitKG, condition):
    self.unitID = unitID
    self.limitKG = limitKG
    self.condition = condition.lower()
    self.__conditions = condition
    self.count = 0
    self.activePackage = None
  def getConditons(self):
    return self.__conditions
  def setCondition(self, newCon):
    conditions = newCon.lower()
    if newCon == "Charging" or newCon =="idle" or newCon == "loading" or
newCon == "delivering" or newCon == "returning":
      self.__conditions = newCon
    else:
      print("invalid condition")
  def attachPackage(self, obj):
    if self.__conditions == "idle" and obj.weight <= self.limitKG:
      self.activePackage = obj
      self.count += obj.weight
      return True
    else:
      return False

  def detachPackage(self):
    self.activePackage = None
    self.count = 0
    return True

class FleetManager:
   def __init__(self, droneList, packageList):
      self.droneList = droneList
      self.units = {}
      for drone in droneList:
        self.units[drone.unitID] = drone
   def sendDeliveries(self):
      for pkg in self.droneList[:]:
        for unit in self.units.values():
          if unit.getConditons == "idle":
            if unit.attachPackage(pkg):
              unit.setCondition("loading")
              self.droneList.remove(pkg)
              break

   def runCount(self):
    for unit in self.units.values():
      unit.count += 1
      if unit.getConditons == "loading":
        unit.setCondition("Charging")
        unit.count = 0
      elif unit.getConditons == "Charging":
        unit.setCondition("idle")
        unit.count = 0

P1 = Package('P1', 3)
P2 = Package('P2', 2)
P3 = Package('P3', 1)

U1 = Drone('U1', 10, 'idle')
U2 = Drone('U2', 10, 'idle')

C1 = FleetManager([U1, U2], [P1, P2, P3])
C1.sendDeliveries()

U1.setCondition("delivering")

for step in range(1, 6):
  C1.runCount()
  for u in C1.units.values():
    print(f"Unit {u.unitID}, COndition: {u.getConditons()}, Count:
{u.count}")
```

Output:

```
Unit U1, COndition: delivering, Count: 1
Unit U2, COndition: idle, Count: 1
Unit U1, COndition: delivering, Count: 2
Unit U2, COndition: idle, Count: 2
Unit U1, COndition: delivering, Count: 3
Unit U2, COndition: idle, Count: 3
Unit U1, COndition: delivering, Count: 4
Unit U2, COndition: idle, Count: 4
Unit U1, COndition: delivering, Count: 5
Unit U2, COndition: idle, Count: 5
```

Task#4:

```python
class Image:
  def __init__(self, pixels):
    self.pixels = pixels
  def applyChanges(self, transformationFunctin):
    self.pixels = transformationFunctin(self.pixels)
  def copy(self):
    newPixels = [row[:] for row in self.pixels]
    return Image(newPixels)

def flipHorizontal(pixels):
  horizontalFlipped = []
  for row in pixels:
    temp = []
    for i in range(0, len(row)):
      temp.append(row[len(row)-1-i])
    horizontalFlipped.append(temp)
  return horizontalFlipped

def changeLighting(pixel, value):
  newLight = []
  for row in pixel:
    temp = []
    for i in row:
      i += value
      temp.append(i)
    newLight.append(temp)
  return newLight

def turnClockwise(pixel):
  turned = []
  for i in range(len(pixel[0])):
    temp = []
    for j in range(len(pixel)-1, -1, -1):
      temp.append(pixel[j][i])
    turned.append(temp)
  return turned

class TransformationPipeline:
  def __init__(self):
    self.steps = []
  def addAction(self, action):
    self.steps.append(action)
  def apply(self, originalImage):
    results = []
    for funct in self.steps:
      copyImage = originalImage.copy()
      copyImage.applyChanges(funct)
      results.append(copyImage.pixels)
    return results

pixels = [[10,20,30],[40,50,60]]
Img = Image(pixels)
pipeline = TransformationPipeline()
pipeline.addAction(flipHorizontal)
pipeline.addAction(lambda x: changeLighting(x, 10))
pipeline.addAction(turnClockwise)
results = pipeline.apply(Img)

results = pipeline.apply(Img)

print("Original Image:")
for row in pixels:
    print(row)

print("\nChanged Images:")
for i, out in enumerate(results):
    print(f"\nStep {i + 1}:")
    for row in out:
        print(row)
```

Output:

```
Original Image:
```

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

Changed Images:

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

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

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