Hands-on Activity 1.1 | Optimization and Knapsack Problem

Objective(s):

This activity aims to demonstrate how to apply greedy and brute force algorithms to solve optimization problems

Intended Learning Outcomes (ILOs):

- Demonstrate how to solve knapsacks problems using greedy algorithm
- Demonstrate how to solve knapsacks problems using brute force algorithm

Resources:

- Jupyter Notebook

Procedures:

1. Create a Food class that defines the following:

- name of the food
- value of the food
- calories of the food

2. Create the following methods inside the Food class:

- A method that returns the value of the food
- A method that returns the cost of the food
- A method that calculates the density of the food (Value / Cost)
- A method that returns a string to display the name, value and calories of the food

```
class Food(object):
    def __init__(self, n, v, w):
        # Make the variables private
        self.name = n
        self.value = v
        self.calories = w
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def density(self):
        return self.getValue()/self.getCost()
    def __str__(self):
        return self.name + ': <' + str(self.value)+ ', ' + str(self.calories) + '>'
```

3. Create a buildMenu method that builds the name, value and calories of the food

```
def buildMenu(names, values, calories):
    menu = []
    for i in range(len(values)):
        menu.append(Food(names[i], values[i],calories[i]))
    return menu
```

4. Create a method greedy to return total value and cost of added food based on the desired maximum cost

```
def greedy(items, maxCost, keyFunction):
    """Assumes items a list, maxCost >= 0,         keyFunction maps elements of items to numbers"""
    itemsCopy = sorted(items, key = keyFunction,
                       reverse = True)
    result = []
    totalValue, totalCost = 0.0, 0.0
    for i in range(len(itemsCopy)):
        if (totalCost+itemsCopy[i].getCost()) <= maxCost:
            result.append(itemsCopy[i])
            totalCost += itemsCopy[i].getCost()
            totalValue += itemsCopy[i].getValue()
    return (result, totalValue)
```

5. Create a testGreedy method to test the greedy method

```
def testGreedy(items, constraint, keyFunction):
    taken, val = greedy(items, constraint, keyFunction)
    print('Total value of items taken =', val)
    for item in taken:
        print('   ', item)


def testGreedys(foods, maxUnits):
    print('Use greedy by value to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.getValue)
    print('\nUse greedy by cost to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, lambda x: 1/Food.getCost(x))
    print('\nUse greedy by density to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.density)
```

6. Create arrays of food name, values and calories
7. Call the buildMenu to create menu for food
8. Use testGreedys method to pick food according to the desired calories

```
names = ['wine', 'beer', 'pizza', 'burger', 'fries','cola', 'apple', 'donut', 'cake']
values = [89,90,95,100,90,79,50,10]
calories = [123,154,258,354,365,150,95,195]
foods = buildMenu(names, values, calories)
testGreedys(foods, 2000)
```

```
Use greedy by value to allocate 2000 calories
Total value of items taken = 603.0
    burger: <100, 354>
    pizza: <95, 258>
    beer: <90, 154>
    fries: <90, 365>
    wine: <89, 123>
    cola: <79, 150>
    apple: <50, 95>
    donut: <10, 195>

Use greedy by cost to allocate 2000 calories
Total value of items taken = 603.0
    apple: <50, 95>
    wine: <89, 123>
    cola: <79, 150>
    beer: <90, 154>
    donut: <10, 195>
    pizza: <95, 258>
    burger: <100, 354>
    fries: <90, 365>

Use greedy by density to allocate 2000 calories
Total value of items taken = 603.0
    wine: <89, 123>
    beer: <90, 154>
    cola: <79, 150>
    apple: <50, 95>
    pizza: <95, 258>
    burger: <100, 354>
    fries: <90, 365>
    donut: <10, 195>
```

```
class Food(object):
    def __init__(self, n, v, w, q):
        # Make the variables private
        self.name = n
        self.value = v
        self.calories = w
        self.weights = q
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def getWeights(self):
      return self.weights
    def density(self):
        return self.getValue()/self.getCost()
    def __str__(self):
        return self.name + ': <' + str(self.value)+ ', ' + str(self.calories) + ', '  + str(self.weights) + '>'


def buildMenu(names, values, calories, weight):
    menu = []
    for i in range(len(values)):
        menu.append(Food(names[i], values[i],calories[i],weight[i]))
    return menu


def greedy(items, maxCost, keyFunction):
    """Assumes items a list, maxCost >= 0,         keyFunction maps elements of items to numbers"""
    itemsCopy = sorted(items, key = keyFunction,
                       reverse = True)
    result = []
    totalValue, totalCost = 0.0, 0.0
    for i in range(len(itemsCopy)):
        if (totalCost+itemsCopy[i].getCost()) <= maxCost:
            result.append(itemsCopy[i])
            totalCost += itemsCopy[i].getCost()
            totalValue += itemsCopy[i].getValue()
    return (result, totalValue)


def testGreedy(items, constraint, keyFunction):
    taken, val = greedy(items, constraint, keyFunction)
    print('Total value of items taken =', val)
    for item in taken:
        print('   ', item)


def testGreedys(foods, maxUnits):
    print('Use greedy by value to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.getValue)
    print('\nUse greedy by cost to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, lambda x: 1/Food.getCost(x))
    print('\nUse greedy by density to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.density,)
    print('\nUse greedy by density to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.getWeights)



names = ['wine', 'beer', 'pizza', 'burger', 'fries','cola', 'apple', 'donut', 'cake']
values = [89,90,95,100,90,79,50,10]
calories = [123,154,258,354,365,150,95,195]
weights =[5,8,12,15,17,20,26,30]
foods = buildMenu(names, values, calories, weights)
testGreedys(foods, 2000)
```

```
Use greedy by value to allocate 2000 calories
Total value of items taken = 603.0
    burger: <100, 354, 15>
    pizza: <95, 258, 12>
    beer: <90, 154, 8>
    fries: <90, 365, 17>
    wine: <89, 123, 5>
    cola: <79, 150, 20>
    apple: <50, 95, 26>
    donut: <10, 195, 30>

Use greedy by cost to allocate 2000 calories
Total value of items taken = 603.0
    apple: <50, 95, 26>
    wine: <89, 123, 5>
    cola: <79, 150, 20>
    beer: <90, 154, 8>
    donut: <10, 195, 30>
    pizza: <95, 258, 12>
    burger: <100, 354, 15>
    fries: <90, 365, 17>

Use greedy by density to allocate 2000 calories
Total value of items taken = 603.0
    wine: <89, 123, 5>
    beer: <90, 154, 8>
    cola: <79, 150, 20>
    apple: <50, 95, 26>
    pizza: <95, 258, 12>
    burger: <100, 354, 15>
    fries: <90, 365, 17>
    donut: <10, 195, 30>

Use greedy by density to allocate 2000 calories
Total value of items taken = 603.0
    donut: <10, 195, 30>
    apple: <50, 95, 26>
    cola: <79, 150, 20>
    fries: <90, 365, 17>
    burger: <100, 354, 15>
    pizza: <95, 258, 12>
    beer: <90, 154, 8>
    wine: <89, 123, 5>
```

Task 1: Change the maxUnits to 100

```
names = ['wine', 'beer', 'pizza', 'burger', 'fries','cola', 'apple', 'donut', 'cake']
values = [89,90,95,120,90,79,50,10]
calories = [123,154,258,354,365,150,95,195]
weights =[5,8,12,15,17,20,26,30]
foods = buildMenu(names, values, calories, weights)
testGreedys(foods, 100)
```

```
Use greedy by value to allocate 100 calories
Total value of items taken = 50.0
    apple: <50, 95, 26>

Use greedy by cost to allocate 100 calories
Total value of items taken = 50.0
    apple: <50, 95, 26>

Use greedy by density to allocate 100 calories
Total value of items taken = 50.0
    apple: <50, 95, 26>

Use greedy by density to allocate 100 calories
Total value of items taken = 50.0
    apple: <50, 95, 26>
```

Task 2: Modify codes to add additional weight (criterion) to select food items.

```
# type your code here
names = ['wine', 'beer', 'pizza', 'burger', 'fries','cola', 'apple', 'donut', 'cake']
values = [89,90,95,120,90,79,50,10]
calories = [123,154,258,354,365,150,95,195]
weights =[5,8,12,15,17,20,26,30]
foods = buildMenu(names, values, calories, weights)
testGreedys(foods, 2000)
```

```
Use greedy by value to allocate 2000 calories
Total value of items taken = 623.0
    burger: <120, 354, 15>
    pizza: <95, 258, 12>
    beer: <90, 154, 8>
    fries: <90, 365, 17>
    wine: <89, 123, 5>
    cola: <79, 150, 20>
    apple: <50, 95, 26>
    donut: <10, 195, 30>

Use greedy by cost to allocate 2000 calories
Total value of items taken = 623.0
    apple: <50, 95, 26>
```

```
        wine: <89, 123, 5>
        cola: <79, 150, 20>
        beer: <90, 154, 8>
        donut: <10, 195, 30>
        pizza: <95, 258, 12>
        burger: <120, 354, 15>
        fries: <90, 365, 17>

    Use greedy by density to allocate 2000 calories
    Total value of items taken = 623.0
        wine: <89, 123, 5>
        beer: <90, 154, 8>
        cola: <79, 150, 20>
        apple: <50, 95, 26>
        pizza: <95, 258, 12>
        burger: <120, 354, 15>
        fries: <90, 365, 17>
        donut: <10, 195, 30>

    Use greedy by density to allocate 2000 calories
    Total value of items taken = 623.0
        donut: <10, 195, 30>
        apple: <50, 95, 26>
        cola: <79, 150, 20>
        fries: <90, 365, 17>
        burger: <120, 354, 15>
        pizza: <95, 258, 12>
        beer: <90, 154, 8>
        wine: <89, 123, 5>
```

Task 3: Test your modified code to test the greedy algorithm to select food items with your additional weight.

```
# type your code here
testGreedys(foods, 2000)
```

```
    Use greedy by value to allocate 2000 calories
    Total value of items taken = 623.0
        burger: <120, 354, 15>
        pizza: <95, 258, 12>
        beer: <90, 154, 8>
        fries: <90, 365, 17>
        wine: <89, 123, 5>
        cola: <79, 150, 20>
        apple: <50, 95, 26>
        donut: <10, 195, 30>

    Use greedy by cost to allocate 2000 calories
    Total value of items taken = 623.0
        apple: <50, 95, 26>
        wine: <89, 123, 5>
        cola: <79, 150, 20>
        beer: <90, 154, 8>
        donut: <10, 195, 30>
        pizza: <95, 258, 12>
        burger: <120, 354, 15>
        fries: <90, 365, 17>

    Use greedy by density to allocate 2000 calories
    Total value of items taken = 623.0
        wine: <89, 123, 5>
        beer: <90, 154, 8>
        cola: <79, 150, 20>
        apple: <50, 95, 26>
        pizza: <95, 258, 12>
        burger: <120, 354, 15>
        fries: <90, 365, 17>
        donut: <10, 195, 30>

    Use greedy by density to allocate 2000 calories
    Total value of items taken = 623.0
        donut: <10, 195, 30>
        apple: <50, 95, 26>
        cola: <79, 150, 20>
        fries: <90, 365, 17>
        burger: <120, 354, 15>
        pizza: <95, 258, 12>
        beer: <90, 154, 8>
        wine: <89, 123, 5>
```

9. Create method to use Bruteforce algorithm instead of greedy algorithm

```
def maxVal(toConsider, avail):
    """Assumes toConsider a list of items, avail a weight
       Returns a tuple of the total value of a solution to the
         0/1 knapsack problem and the items of that solution"""
    if toConsider == [] or avail == 0:
        result = (0, ())
    elif toConsider[0].getCost() > avail:
        #Explore right branch only
        result = maxVal(toConsider[1:], avail)
    else:
        nextItem = toConsider[0]
        #Explore left branch
        withVal, withToTake = maxVal(toConsider[1:],
                                     avail - nextItem.getCost())
        withVal += nextItem.getValue()
        #Explore right branch
        withoutVal, withoutToTake = maxVal(toConsider[1:], avail)
        #Choose better branch
        if withVal > withoutVal:
            result = (withVal, withToTake + (nextItem,))
        else:
            result = (withoutVal, withoutToTake)
    return result


def testMaxVal(foods, maxUnits, printItems = True):
    print('Use search tree to allocate', maxUnits,
          'calories')
    val, taken = maxVal(foods, maxUnits)
    print('Total costs of foods taken =', val)
    if printItems:
        for item in taken:
            print('   ', item)


names = ['wine', 'beer', 'pizza', 'burger', 'fries','cola', 'apple', 'donut', 'cake']
values = [89,90,95,100,90,79,50,10]
calories = [123,154,258,354,365,150,95,195]
weights =[5,8,12,15,17,20,26,30]
foods = buildMenu(names, values, calories, weights)
testMaxVal(foods, 2400)
```

```
Use search tree to allocate 2400 calories
Total costs of foods taken = 603
    donut: <10, 195, 30>
    apple: <50, 95, 26>
    cola: <79, 150, 20>
    fries: <90, 365, 17>
    burger: <100, 354, 15>
    pizza: <95, 258, 12>
    beer: <90, 154, 8>
    wine: <89, 123, 5>
```

Supplementary Activity:

- Choose a real-world problem that solves knapsacks problem
- Use the greedy and brute force algorithm to solve knapsacks problem

CHOSEN PROBLEM

I will give you some context of what is it all about, it's all about Protein Intake and cost of buying of protein foods of how much it will cost depending how much protein goal does people go to the gym want that I choose for a real world problem because for the people who goes to the gym like me who's having a problem taking and buying.

Conclusion:

For this Supplementary I applied the knapsacks problem because it talks about how many/much does it need to take a protein that you can go eat & buy base on my chosen real world problem how much they it will cost therefore, I apply all the things especially greedy and bruteforce refering in the procedure code. I also did apply those codes even though I'm having a hardtime to understand the other variables and function within these sort of code.

type your conclusion here

```
class GymFoodProteinIntake(object):
  def __init__(self, n, p, s, b):
      self.name = n
      self.price = p
      self.protein = s
      self.budget = b
  def getPrice(self):
      return self.price
  def getProtein(self):
      return self.protein
  def getBudget(self):
    return self.budget
  def primarycost(self):
      return self.getPrice()/self.getProtein()
  def __str__(self):
      return self.name + ': <' + str(self.price)+ ', ' + str(self.protein) + ', ' + str(self.budget) + '>'
def MyPacks(names, prices, calories, budget):
    pack = []
    for i in range(len(prices)):
        pack.append(GymFoodProteinIntake(names[i], prices[i],calories[i],budget[i]))
    return pack
def greedy(items, maxCost, keyFunction):
    itemsCopy = sorted(items, key = keyFunction,
                       reverse = True)
    result = []
    totalValue, totalCost = 0.0, 0.0
    for i in range(len(itemsCopy)):
        if (totalCost+itemsCopy[i].getProtein()) <= maxCost:
            result.append(itemsCopy[i])
            totalCost += itemsCopy[i].getProtein()
            totalValue += itemsCopy[i].getPrice()
    return (result, totalValue)
def testGreedy(items, constraint, keyFunction):
    taken, val = greedy(items, constraint, keyFunction)
    print('Total Price of this items to buy =', val)
    for item in taken:
        print('   ', item)
def testGreedys(foods, maxUnits):
    print('Use greedy by price to allocate the cost of 2 foods', maxUnits,              'Protein')
    testGreedy(foods, maxUnits, GymFoodProteinIntake.getPrice)
    print('\nUse greedy by cost to allocate the cost of 3 foods', maxUnits,             'Protein')
    testGreedy(foods, maxUnits, lambda x: 1/GymFoodProteinIntake.getProtein(x))
    print('\nUse greedy by budget to allocate the cost of 3 foods', maxUnits,           'Protein')
    testGreedy(foods, maxUnits, GymFoodProteinIntake.primarycost,)
    print('\nUse greedy by budget to allocate the cost of 3 foods', maxUnits,           'Protein')
    testGreedy(foods, maxUnits, GymFoodProteinIntake.getBudget)
def maxVal(toConsider, avail):
    if toConsider == [] or avail == 0:
        result = (0, ())
    elif toConsider[0].getProtein() > avail:
        result = maxVal(toConsider[1:], avail)
    else:
        nextItem = toConsider[0]
        withVal, withToTake = maxVal(toConsider[1:],
                         avail - nextItem.getProtein())
        withVal += nextItem.getPrice()
        withoutVal, withoutToTake = maxVal(toConsider[1:], avail)
        if withVal > withoutVal:
            result = (withVal, withToTake + (nextItem,))
        else:
            result = (withoutVal, withoutToTake)
    return result
def testMaxVal(foods, maxUnits, printItems = True):
        print('Use search tree to allocate', maxUnits,
              'protein within a week')
        print("Using Bruteforce to allocate price ")
        val, taken = maxVal(foods, maxUnits)
        print('Total costs of foods taken =', val)
        if printItems:
            for item in taken:
                print('   ', item)
names = ['Eggs', 'Almonds', 'Chicken breast', 'Cottage Cheese', 'Milk','Fish']
```

```
price = [89,90,95,100,90,79,]
protein = [60,63,255,312,75,150]
budget =[500,800,120,150,170,200]
foods = MyPacks(names, price, protein, budget)
testGreedys(foods, 650)
testMaxVal(foods, 550)
```

```
Use greedy by price to allocate the cost of 2 foods 650 Protein
Total Price of this items to buy = 285.0
    Cottage Cheese: <100, 312, 150>
    Chicken breast: <95, 255, 120>
    Almonds: <90, 63, 800>

Use greedy by cost to allocate the cost of 3 foods 650 Protein
Total Price of this items to buy = 443.0
    Eggs: <89, 60, 500>
    Almonds: <90, 63, 800>
    Milk: <90, 75, 170>
    Fish: <79, 150, 200>
    Chicken breast: <95, 255, 120>

Use greedy by budget to allocate the cost of 3 foods 650 Protein
Total Price of this items to buy = 443.0
    Eggs: <89, 60, 500>
    Almonds: <90, 63, 800>
    Milk: <90, 75, 170>
    Fish: <79, 150, 200>
    Chicken breast: <95, 255, 120>

Use greedy by budget to allocate the cost of 3 foods 650 Protein
Total Price of this items to buy = 443.0
    Almonds: <90, 63, 800>
    Eggs: <89, 60, 500>
    Fish: <79, 150, 200>
    Milk: <90, 75, 170>
    Chicken breast: <95, 255, 120>
Use search tree to allocate 550 protein within a week
Using Bruteforce to allocate price
Total costs of foods taken = 369
    Milk: <90, 75, 170>
    Cottage Cheese: <100, 312, 150>
    Almonds: <90, 63, 800>
    Eggs: <89, 60, 500>
```