Report

- How to run the code in Step II and Step III
  - Step II

    This is an example of executing step2.py. The name of the output file of this example would be:
    - "step2_task1_datasetA_0.006_result1.txt"
    - "step2_task1_datasetA_0.006_result2.txt"
    - "step2_task2_datasetA_0.006_result1.txt"

    –f: The dataset path

    –s: The minimum support (the default minimum support: 0.01)

    –p: The output file path (the default path is "./")

```
python3 Step2.py -f datasetA.data -s 0.003 -p ./result_files
```

  - Step III

    This is an example of executing step3.py. The name of the output file of this example would be:
    - "step3_task1_datasetA_0.006_result1.txt"
    - "step3_task1_datasetA_0.006_result2.txt"

    –f: The dataset path (the default dataset is "datasetA.data")

    –s: The minimum support (the default minimum support: 0.01)

    –c: The minimum confidence (the default minimum confidence: 0.5)

    –p: The output file path (the default output file path is "./")

```
python3 Step3.py -f datasetA.data -s 0.006 -p ./
```

- Step II
  - Report on the mining algorithms/codes:
    - The modification I made for Task 1 and Task 2
      1. 在Task 1中，我使用平行化的方法搜集support大於等於minimum support的itemsets，如程式碼的mp.pool，使用多核心平行處理，計算各個transaction的出現次數，以計算各個itemset的support，並使用check_item_support這個函數計算每個itemset在transaction list中出現的次數，以計算support，並把support大於minimum support的itemset加入到_itemSet，並在最後回傳support大於minimum support的itemsets。

```python
def returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet):
    """calculates the support for items in the itemSet and returns a subset
    of the itemSet each of whose elements satisfies the minimum support"""
    _itemSet = set()
    localSet = defaultdict(int)

    # 平行計算min support
    with mp.Pool(processes=mp.cpu_count()) as pool:
        results = pool.starmap(check_item_support, [(item, transactionList) for item in itemSet])

    for item, count in results:
        freqSet[item] += count
        support = float(count) / len(transactionList)
        # 把support >= minSupport 的 itemset加入_itemSet，並在最後回傳_itemSet
        if support >= minSupport:
            _itemSet.add(item)

    return _itemSet

def check_item_support(item, transactionList):
    count = sum(1 for transaction in transactionList if item.issubset(transaction))
    return item, count
```

2. 在Task 2中，我新增了isClosed()和getFrequentClosedItemsets()這兩個函數，在isClosed()這個函數中，我在frequent itemset中確認每個itemset若沒有其superset跟他有一樣的support，就為frequent closed itemset。而最後在getFrequentClosedItemsets()中搜集所有frequent closed itemset並回傳。

```python
def isClosed(freqSet, item, support):
    """Check if an itemset is closed"""
    for other_item in freqSet:
        if item != other_item and item.issubset(other_item) and freqSet[other_item] == support:
            return False
    return True

def getFrequentClosedItemsets(freqSet, largeSet, transactionList):
    """Filter for frequent closed itemsets"""
    closed_itemsets = []

    for key, value in largeSet.items():
        for item in value:
            support = float(freqSet[item]) / len(transactionList)
            if isClosed(freqSet, item, freqSet[item]):
                closed_itemsets.append((tuple(item), support))

    return closed_itemsets
```

3. 在原本的程式碼中，我修改了printResults()，以符合作業要求，並且因應command line的指令，利用傳入的參數修改輸出檔案的檔名，以辨別不同資料集、支持度的輸出檔案。另外，printClosedItemsets()則是獨立出來輸出task 2的frequent closed itemset的函數。

```python
def printResults(items, candidate_count, dataset_name, output_path, task_num, support_str):
    idx = 1
    total_num_itemset = len(items)
    result1_filename = os.path.join(output_path, f"step2_task{task_num}_{dataset_name}_{support_str}_result1.txt")
    result2_filename = os.path.join(output_path, f"step2_task{task_num}_{dataset_name}_{support_str}_result2.txt")

    with open(result1_filename, 'w') as f1:
        with open(result2_filename, 'w') as f2:
            f2.write(str(total_num_itemset) + "\n")
            for item, support in sorted(items, key=lambda x: x[1], reverse=True):
                item_str = "{" + ",".join(item) + "}"
                support = support * 100
                f1.write(f"{support:.1f}\t{item_str}\n")
            for candi in candidate_count:
                f2.write(f"{idx}\t{candidate_count[idx-1][0]}\t{candidate_count[idx-1][1]}\n")
                idx += 1
```

```python
def printClosedItemsets(closed_items, dataset_name, output_path, support_str):
    result_filename = os.path.join(output_path, f"step2_task2_{dataset_name}_{support_str}_result1.txt")
    total_num_closed_itemsets = len(closed_items)
    with open(result_filename, 'w') as f:
        f.write(str(total_num_closed_itemsets) + "\n")
        for item, support in sorted(closed_items, key=lambda x: x[1], reverse=True):
            item_str = "{" + ",".join(item) + "}"
            support = support * 100
            f.write(f"{support:.1f}\t{item_str}\n")
```

- ○ The restrictions
  這個演算法測試過助教提供的Toy dataset（資料集大小為一百萬筆transaction），其執行時間需要用到24~48小時之間的時間，相較於step3所使用到的FP-Growth只需要四個小時多的時間，因此，資料集的數量限制了此Apriori演算法的執行時間。
- ○ Problems encountered in mining
  在這個演算法中遇到最大的問題就是執行時間，在跑Dataset C (50萬筆資料)的情況下，要花費超過五個小時的時間，後來嘗試平行化計算itemset的support，發現時間大幅減少，最後跑Dataset C只需要兩個小時多的時間。然而，遇到100萬筆資料時，花費時間還是太長，要花一至兩天的時間才能完成。
- ○ Any observations/discoveries
  在實作的過程中，發現尋找frequent itemset的過程可以平行化處理，在尋找哪個transaction出現次數大於等於minimum的過程省下大量的時間。另外，也觀察到沒有使用平行化處理時，使用越小的minimum support，花費時間會越長，但使用平行化執行程式後，則沒有這樣的規律，mining時間與minimum support的大小沒有太大的關聯。

- Paste the screenshot of the computation time
  - Dataset A with minimum support 0.003
```
Task1 computation time: 9.94 seconds.
Task2 computation time: 74.85 seconds.
```
  - Dataset A with minimum support 0.006
```
Task1 computation time: 1.41 seconds.
Task2 computation time: 5.82 seconds.
```
  - Dataset A with minimum support 0.009
```
Task1 computation time: 0.90 seconds.
Task2 computation time: 2.18 seconds.
```
  - Dataset B with minimum support 0.002
```
Task1 computation time: 492.82 seconds.
Task2 computation time: 571.76 seconds.
```
  - Dataset B with minimum support 0.004
```
Task1 computation time: 307.48 seconds.
Task2 computation time: 320.13 seconds.
```
  - Dataset B with minimum support 0.006
```
Task1 computation time: 271.56 seconds.
Task2 computation time: 276.88 seconds.
```
  - Dataset C with minimum support 0.005
```
Task1 computation time: 7020.31 seconds.
Task2 computation time: 7047.03 seconds.
```
  - Dataset C with minimum support 0.010
```
Task1 computation time: 8466.41 seconds.
Task2 computation time: 8472.84 seconds.
```

- Dataset C with minimum support 0.015

```
Task1 computation time: 7309.81 seconds.
Task2 computation time: 7313.81 seconds.
```

- For Task 2, you also need to show the ratio of computation time compared to that of Task 1 in your report.
  - Dataset A with minimum support 0.003

    ```
    Time ratio (Task2/Task1): 7.53%
    ```

  - Dataset A with minimum support 0.006

    ```
    Time ratio (Task2/Task1): 4.14%
    ```

  - Dataset A with minimum support 0.009

    ```
    Time ratio (Task2/Task1): 2.41%
    ```

  - Dataset B with minimum support 0.002

    ```
    Time ratio (Task2/Task1): 1.16%
    ```

  - Dataset B with minimum support 0.004

    ```
    Time ratio (Task2/Task1): 1.04%
    ```

  - Dataset B with minimum support 0.006

    ```
    Time ratio (Task2/Task1): 1.02%
    ```

  - Dataset C with minimum support 0.005

    ```
    Time ratio (Task2/Task1): 1.00%
    ```

  - Dataset C with minimum support 0.010

    ```
    Time ratio (Task2/Task1): 1.00%
    ```

  - Dataset C with minimum support 0.015

    ```
    Time ratio (Task2/Task1): 1.00%
    ```

- Paste the screenshot of your code modification for Task 1 and Task 2 with comments and explain it.
  - 在Task 1中，我使用平行化的方式找到support大於等於minimunm support的itemsets。如下圖所示，我使用multiprocessing.Pool，而其中的process即為系統的CPU核心數，接著，我使用pool.starmap()將每個item與transaction配對，平行呼叫check_item_support()這個函數，以同時計算多個item的support。而最後的results則是包含多個item與count配對的列表，其中count為該item出現的次數，用來計算support。接著，遍尋results內的元素，freqSet[item] += count 代表將count累加到freqSet對應的item中，接著除以transaction list的長度以計算support，若support大於等於我們所指定的minimum，則加到_itemSet中，最後回傳由frequent items（support >= minimum support的itemset）所構成的_itemSet。另外，我也使用在command line輸入的指令參數，輸出相對應的文件檔名（dataset, minimum support, task 1 or 2, result 1 or 2）、以及作業所要求的格式。

```python
def returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet):
    """calculates the support for items in the itemSet and returns a subset
    of the itemSet each of whose elements satisfies the minimum support"""
    _itemSet = set()
    localSet = defaultdict(int)

    # 平行計算min support
    with mp.Pool(processes=mp.cpu_count()) as pool:
        results = pool.starmap(check_item_support, [(item, transactionList) for item in itemSet])

    for item, count in results:
        freqSet[item] += count
        support = float(count) / len(transactionList)
        # 把support >= minSupport 的 itemset加入__itemSet，並在最後回傳_itemSet
        if support >= minSupport:
            _itemSet.add(item)

    return _itemSet

def check_item_support(item, transactionList):
    count = sum(1 for transaction in transactionList if item.issubset(transaction))
    return item, count
```

```python
def printResults(items, candidate_count, dataset_name, output_path, task_num, support_str):
    idx = 1
    total_num_itemset = len(items)
    result1_filename = os.path.join(output_path, f"step2_task{task_num}_{dataset_name}_{support_str}_result1.txt")
    result2_filename = os.path.join(output_path, f"step2_task{task_num}_{dataset_name}_{support_str}_result2.txt")

    with open(result1_filename, 'w') as f1:
        with open(result2_filename, 'w') as f2:
            f2.write(str(total_num_itemset) + "\n")
            for item, support in sorted(items, key=lambda x: x[1], reverse=True):
                item_str = "{" + ",".join(item) + "}"
                support = support * 100
                f1.write(f"{support:.1f}\t{item_str}\n")
            for candi in candidate_count:
                f2.write(f"{idx}\t{candidate_count[idx-1][0]}\t{candidate_count[idx-1][1]}\n")
                idx += 1
```

- 在Task 2中，我使用getFrequentClosedItemsets()來找尋frequent closed itemsets。由於前面已經計算出frequent itemsets，我們直接利用這個frequent itemsets來mine frequent closed itemsets（而Task 2 的執行時間依然包含Task 1 mining frequent itemsets 的時間），其方式如下程式碼所示，我們遍尋largeSet，並再次計算各個item的support，接著使用isClosed()函數來檢視該item是否為frequent closed item，isClosed()函數會檢查freqSet的其他frequent item (other_item) 判斷是否與其相同、是否為other_item的子集、是否與other_item有一樣的support，若上述三個條件都成立，則該item就不為frequent closed itemset，即回傳False，反之，則回傳True，表示該item為frequent closed item。另外，輸出文件則一樣因應command line輸出不同的文件檔名、格式。

```python
def isClosed(freqSet, item, support):
    """Check if an itemset is closed"""
    for other_item in freqSet:
        if item != other_item and item.issubset(other_item) and freqSet[other_item] == support:
            return False
    return True

def getFrequentClosedItemsets(freqSet, largeSet, transactionList):
    """Filter for frequent closed itemsets"""
    closed_itemsets = []

    for key, value in largeSet.items():
        for item in value:
            support = float(freqSet[item]) / len(transactionList)
            if isClosed(freqSet, item, freqSet[item]):
                closed_itemsets.append((tuple(item), support))

    return closed_itemsets
```

```python
def printClosedItemsets(closed_items, dataset_name, output_path, support_str):
    result_filename = os.path.join(output_path, f"step2_task2_{dataset_name}_{support_str}_result1.txt")
    total_num_closed_itemsets = len(closed_items)
    with open(result_filename, 'w') as f:
        f.write(str(total_num_closed_itemsets) + "\n")
        for item, support in sorted(closed_items, key=lambda x: x[1], reverse=True):
            item_str = "{" + ",".join(item) + "}"
            support = support * 100
            f.write(f"{support:.1f}\t{item_str}\n")
```

- Step III
  - Descriptions of your mining algorithm
    - Relevant references
      https://github.com/chonyy/fpgrowth_py/blob/master/fpgrowth_py/fpgrowth.py
    - Program flow

    在Step 3中，我使用FP-Growth Algorithm來mine frequent itemsets。在主程式中，我使用了四個參數，分別是inputFile, minSupport, minConfidence, outputPath，因應inputFile的格式，若是屬於.data檔案，則先轉成.csv檔案再讀取transaction。接著使用fpgrowthFromFile()函數讀取transaction進行FP-Growth演算法，一開始先計算support，跟Apriori不同的是，這個演算法的minSupport的算法是transaction list數量乘以我們一開始所輸入的minSupport。之後利用constructTree()函數建立FP tree和header table，FP tree中的每個節點包含幾個要素，itemName代表該transaction的名稱、count代表該item在目前的樹中的出現次數、parent則指向該節點的父節點、children指向子節點，而每一筆的transaction按frequency高低依序插入樹中，若遇到樹中已存在該節點，則增加其count，否則創建新的節點並連接到樹中；header table中的元素為大於或等於minimum support的items，每個item會指向FP tree中該item的第一個節點，並以chain的形式串連所有出現在該item的節點，以便快速定位每個frequent itemset在FP tree中的位置，header table會依據frequency的高低來更新FP tree，若FP tree存在（非空），則執行mineTree()函數來遞迴mining FP tree，依照header table 中的item 排序，逐個提取frequent items，若新建的FP tree不為空，則繼續遞迴mining該樹，擴展frequent itemsets，另外，若有新的transaction插入到FP tree中，header table也會因此更新，指向FP tree中新插入的節點，並形成chain以便查詢。最後的associationRule()函數則是用來計算每個association rule的confidence，只保留confidence大於minimum confidence 的rule。最後printResults()函數如Apriori演算法中的一樣，輸出frequent items及其support，並且記錄每個階段的candidate counts。
  - Differences/Improvements in your algorithm
    - 不同於step 2使用的Apriori演算法，在step 3中我使用FP-Growth演算法，即便Apriori中使用平行化的運算大幅改善運算時間，不使用平行運算的FP-Growth在絕大部分的表現（執行時間）還是比Apriori演算法快速，其原因為FP tree能夠直接幫助mining frequent itemsets，不需要生成candidate itemsets，不像Apriori演算法會生成大量的candidate itemsets，在每一輪新增item element數時都需要重新遍尋items，且隨著item數量的增加會大幅增加candidate itemsets的數量，導致計算量大幅提升。另外，針對一些出現次數較少的items，FP-Growth能夠保留item之間的重要關聯性，使得在處理frequent itemsets時更有優勢、更快速。

- 在step3.py中，我更改原始程式碼，使用set()來儲存frequent itemsets，這樣做的目的是為了防止最後輸出frequent itemsets時會跑出重複的items，另外，也新增了printResults()來輸出task 1所需的文件，如同Apriori演算法中的輸出格式一樣。

```python
def printResults(freqItems, candidate_counts,dataset_name, outputPath, task_num, support_str, transactionList):
    result1_filename = os.path.join(outputPath, f"step3_task{task_num}_{dataset_name}_{support_str}_result1.txt")
    result2_filename = os.path.join(outputPath, f"step3_task{task_num}_{dataset_name}_{support_str}_result2.txt")

    with open(result1_filename, 'w') as f1:
        total_transactions = len(transactionList)
        itemSetSupportList = []
        for itemSet in freqItems:
            support = getSupport(itemSet, transactionList) / total_transactions
            if support >= float(support_str):
                itemSetSupportList.append((itemSet, support))
        sorted_itemsets = sorted(itemSetSupportList, key=lambda x: x[1], reverse=True)

        for itemSet, support in sorted_itemsets:
            support = support * 100
            item_str = "{" + ",".join(itemSet) + "}"
            f1.write(f"{support:.1f}\t{item_str}\n")

    with open(result2_filename, 'w') as f2:
        f2.write(f"{len(freqItems)}\n")
        for idx, (before, after) in enumerate(candidate_counts, 1):
            f2.write(f"{idx}\t{before}\t{after}\n")
```

- Computation time
    - Compare the computation time of **Task 1** with Step II
        - Dataset A with minimum support 0.003: −126.56%
        - Dataset A with minimum support 0.006: 29.08%
        - Dataset A with minimum support 0.009: 61.10%
        - Dataset B with minimum support 0.002: −51.74%
        - Dataset B with minimum support 0.004: 0.46%
        - Dataset B with minimum support 0.006: 16.98%
        - Dataset C with minimum support 0.005: 28.35%
        - Dataset C with minimum support 0.010: 45.11%
        - Dataset C with minimum support 0.015: 40.75%
    - Paste the screenshot of the computation time.
        - Dataset A with minimum support 0.003

          ```
          Task 1 computation time: 22.52 seconds.
          ```
        - Dataset A with minimum support 0.006

          ```
          Task 1 computation time: 1.00 seconds.
          ```
        - Dataset A with minimum support 0.009

          ```
          Task 1 computation time: 0.35 seconds.
          ```
        - Dataset B with minimum support 0.002

          ```
          Task 1 computation time: 747.82 seconds.
          ```
        - Dataset B with minimum support 0.004

          ```
          Task 1 computation time: 306.08 seconds.
          ```
        - Dataset B with minimum support 0.006

          ```
          Task 1 computation time: 225.44 seconds.
          ```
        - Dataset C with minimum support 0.005

          ```
          Task 1 computation time: 5029.85 seconds.
          ```

- Dataset C with minimum support 0.010

  ```
  Task 1 computation time: 4647.32 seconds.
  ```

- Dataset C with minimum support 0.015

  ```
  Task 1 computation time: 4331.06 seconds.
  ```

- Discuss the **scalability** of your algorithm in terms of the dataset size (i.e., the rate of change on computing time under different data sizes, the largest dataset size the algorithm can handle, etc).
  - 由上面的執行時間可以看到，資料數由1000筆至100000筆資料時，執行時間變成原來的約100倍，而由100000筆資料至500000筆資料時，執行時間變成原本的約11倍。而又500000筆資料至1000000筆資料時，執行時間則變成原來的2倍左右。