Midterm project report

Tzuyu Wang, [tw329@njit.edu](mailto:tw329@njit.edu)

Repository link:

<https://github.com/tw329/CS634_midterm_project.git>

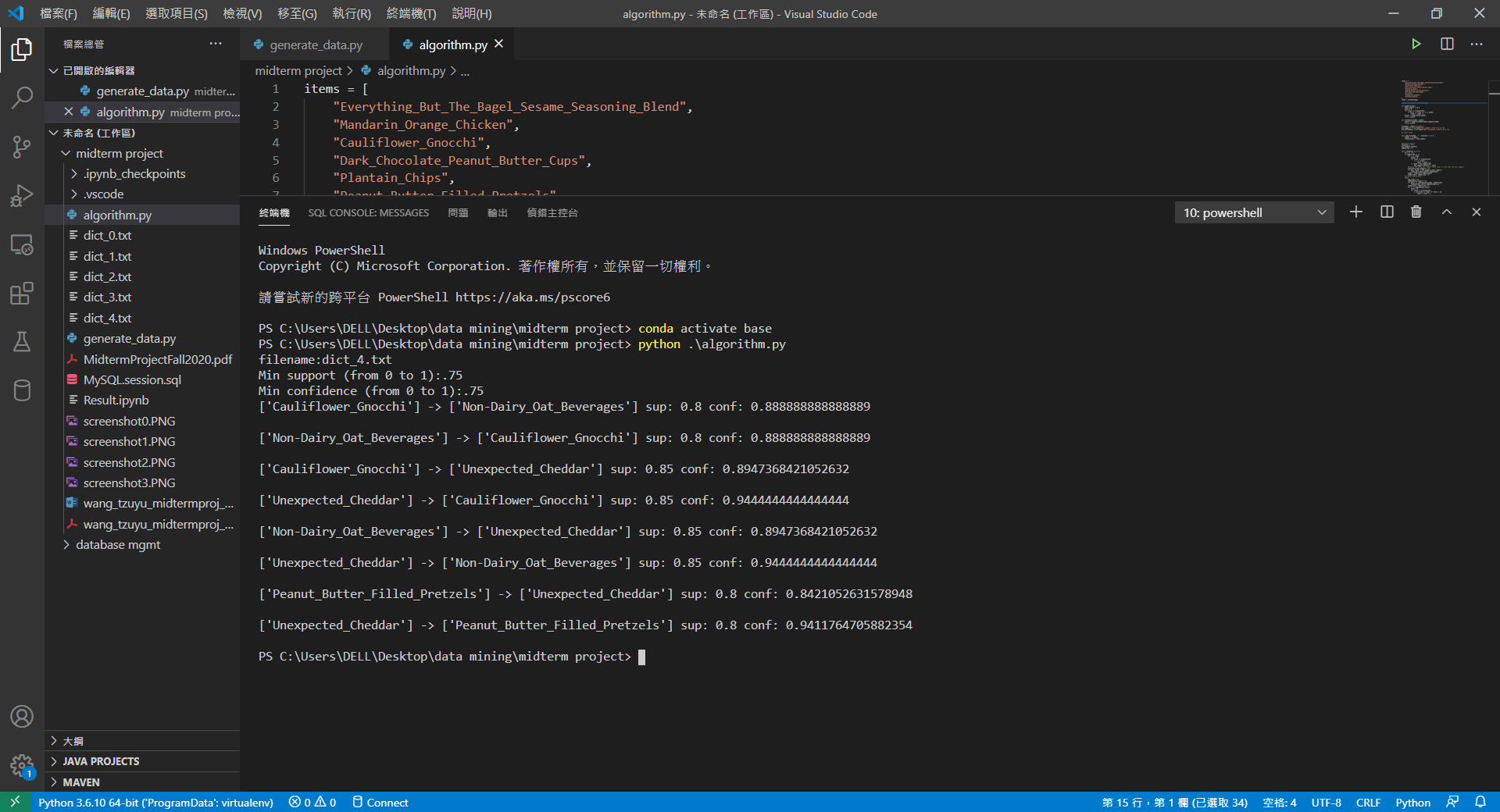
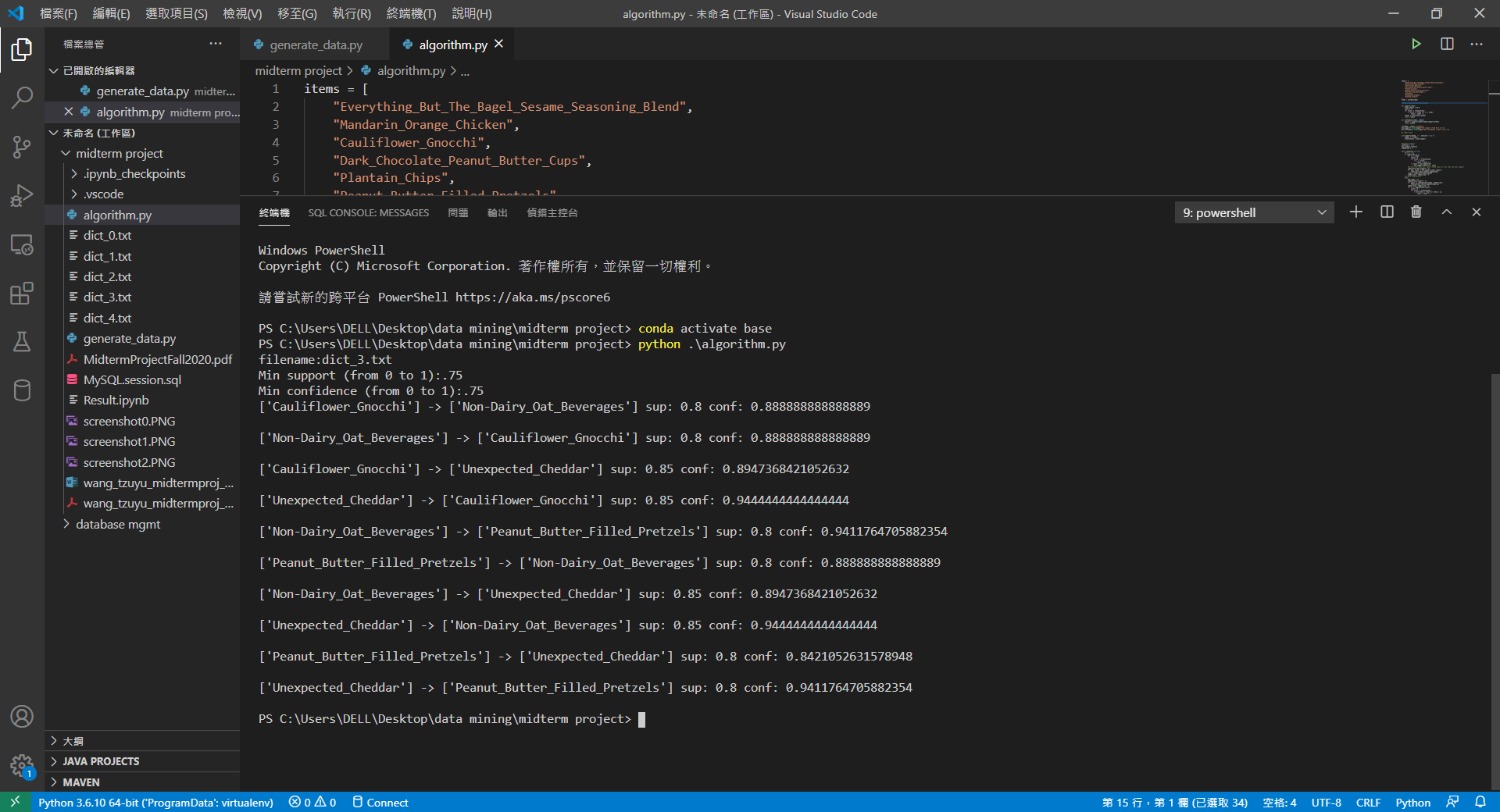
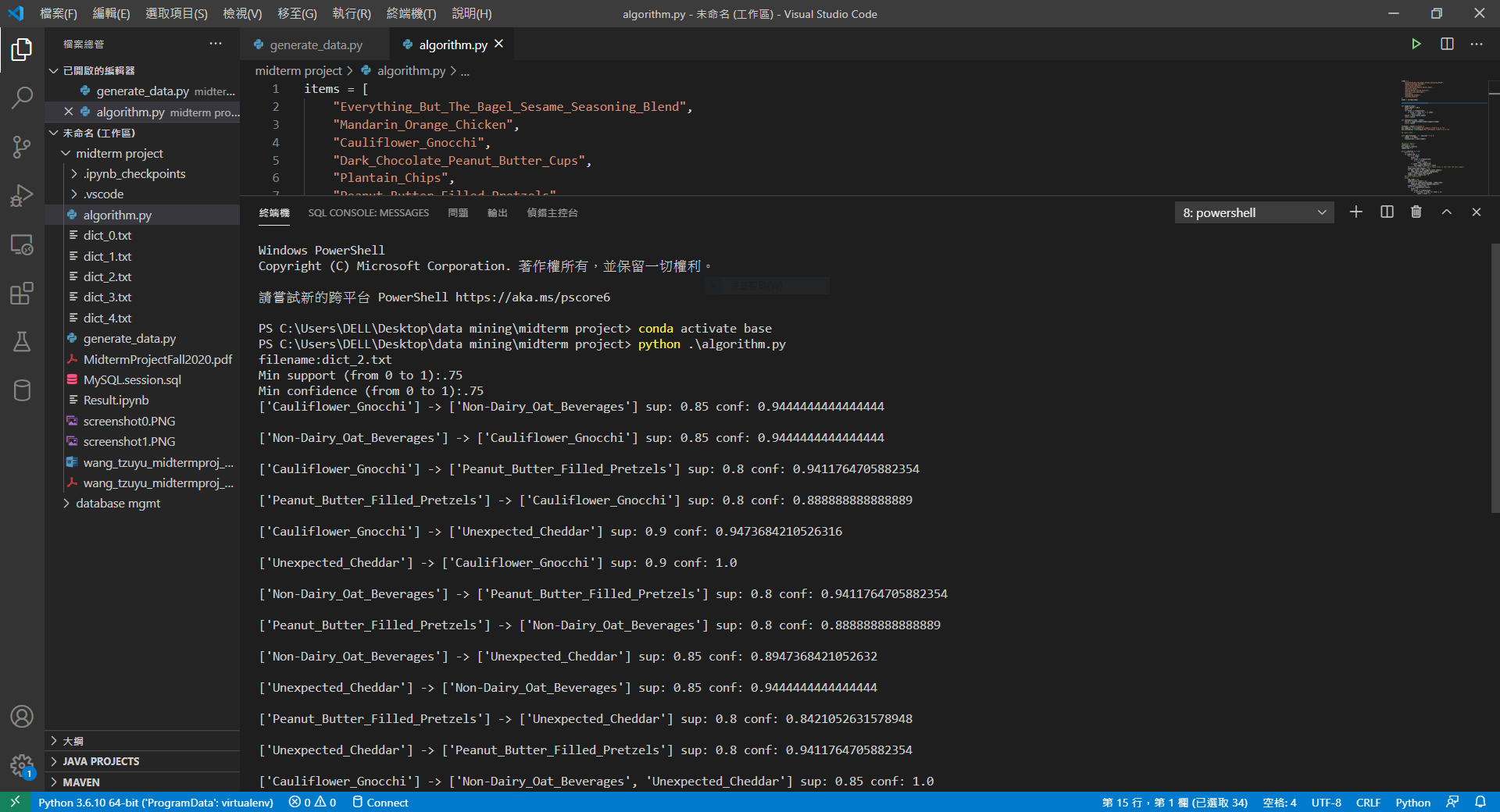
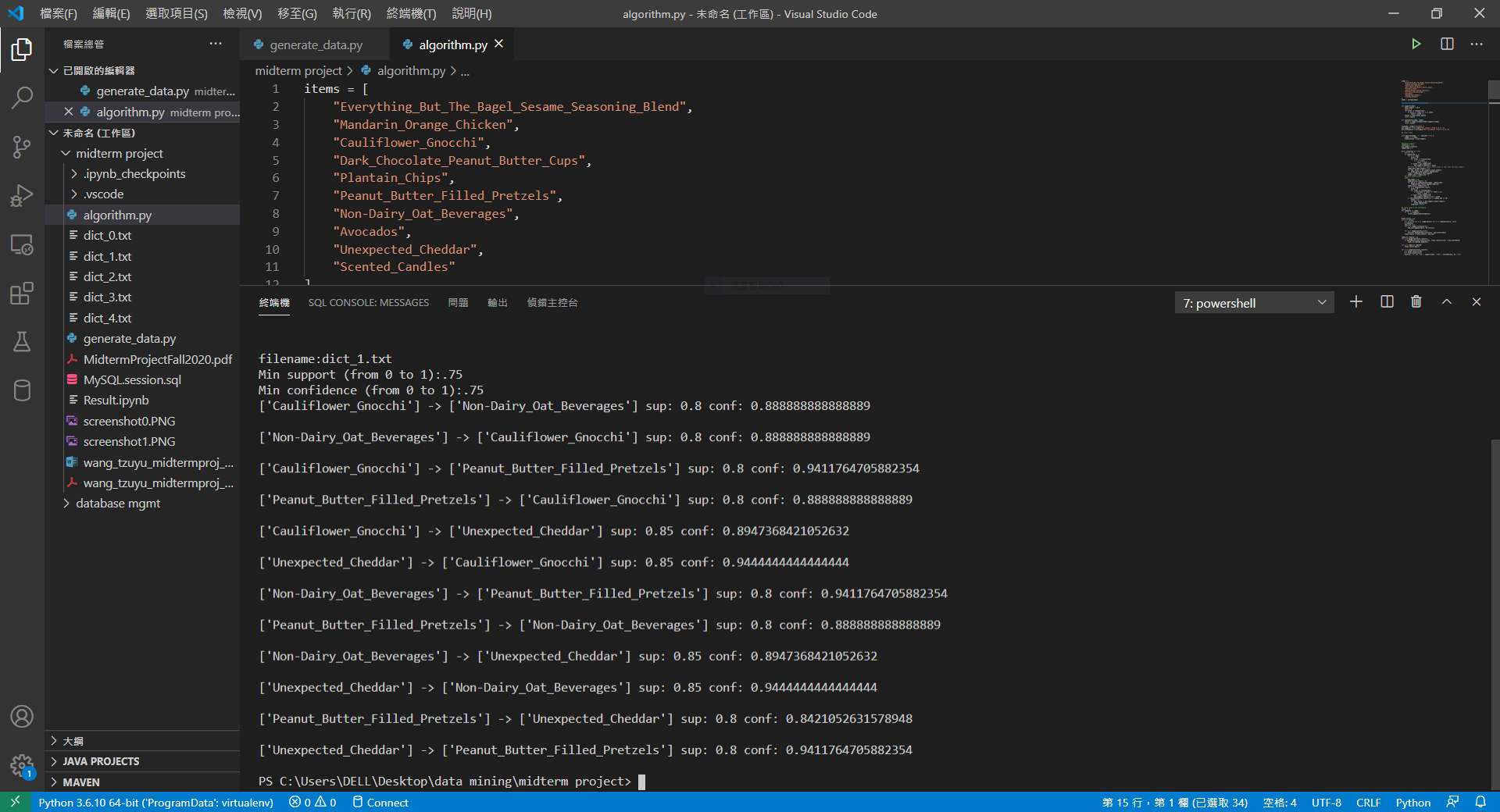
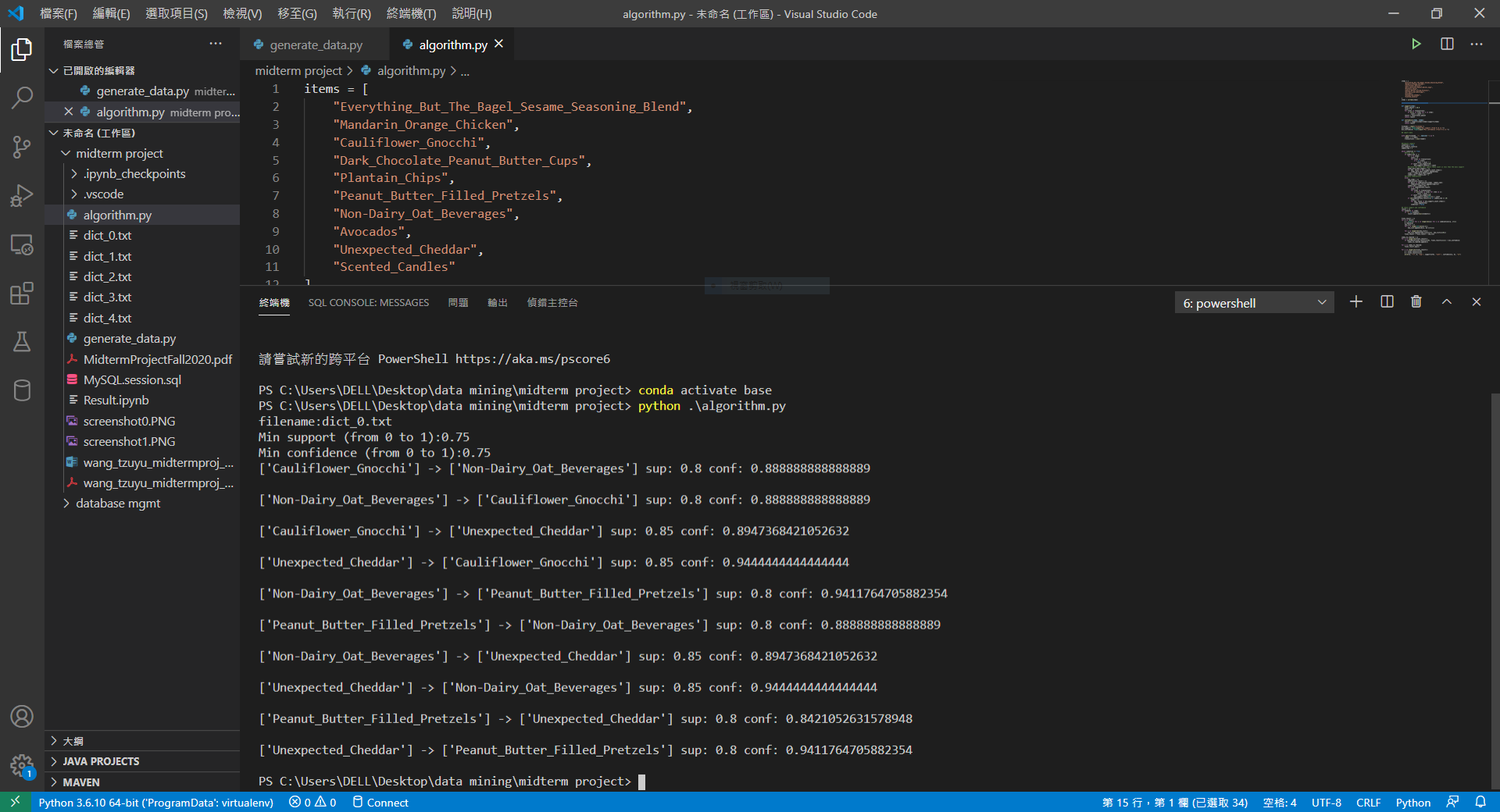
There will be three parts in the report. First, I will explain how I done the project and how should the codes work. The second part will be the screenshots of the results. And the Final part will be the codes.

There are two python files, generate\_data.py and algorithm.py, in my project. First one is to generate transactions data, and the other one is for the algorithm. I wrote it into two different files because in this way I can check if different support or confidence gave different result. And there is a Result.ipynb which combine both files and show the output of the code. The packages used are random and itertools.

In generate.py, the first idea is to generate the transactions randomly, but it seems that it is not easy to see the association if the transactions are all random. Therefore, I create a list which I can let every item show exactly how many times I want. And now, the frequency of each item is controllable. The output of generate.py is five files, dict\_0.txt, dict\_1.txt, dict\_2.txt, dict\_3.txt, and dict\_4.txt. Each of them is a database contains 20 transactions.

In algorithm.py, it required 3 input data, filename, min\_support, and min\_confidence. The filename is for the users to choose which of five databases, and the min\_support and the min\_condfidence is for the users to set the minimum of support and confidence value. After reading in the database, the first thing I do is compute the 1-item support. Keep whose support is greater than the minimum support, and go on for the k-items support. After having the final result, generate the all the association rules and compute the confidence. Thus, there is the final result. I print out every rule, and its support and condifence.

The screenshots followed are results of every database with 75% min\_support and 75% min\_condfidence.



The code of generate\_data.py:

import random

items = [

    "Everything\_But\_The\_Bagel\_Sesame\_Seasoning\_Blend",

    "Mandarin\_Orange\_Chicken",

    "Cauliflower\_Gnocchi",

    "Dark\_Chocolate\_Peanut\_Butter\_Cups",

    "Plantain\_Chips",

    "Peanut\_Butter\_Filled\_Pretzels",

    "Non-Dairy\_Oat\_Beverages",

    "Avocados",

    "Unexpected\_Cheddar",

    "Scented\_Candles"

]

count = [19, 18, 17, 10, 7 ,8 ,2 ,4, 18, 6]

random.shuffle(count)

for i in range(0, 5, 1):

    transaction = []

    for j in range(0, 20, 1):

        transaction.append([])

    for m in range(len(items)):

        for n in random.sample(range(0, 20), k=count[m]):

            transaction[n].append(items[m])

    with open('dict\_'+str(i)+'.txt', 'w', newline='') as f:

        print(transaction, file=f)

The code of algorithm.py:

items = [

    "Everything\_But\_The\_Bagel\_Sesame\_Seasoning\_Blend",

    "Mandarin\_Orange\_Chicken",

    "Cauliflower\_Gnocchi",

    "Dark\_Chocolate\_Peanut\_Butter\_Cups",

    "Plantain\_Chips",

    "Peanut\_Butter\_Filled\_Pretzels",

    "Non-Dairy\_Oat\_Beverages",

    "Avocados",

    "Unexpected\_Cheddar",

    "Scented\_Candles"

]

items = sorted(items)

from itertools import combinations

def support(item):

    total\_amount = 20.0

    count = 0

    for trans in transactions:

        if all(i in trans for i in item):

            count = count + 1

    result = count/total\_amount

    return result

def confidence(item1, item2):

    result = support(item1+item2)/support(item2)

    return result

filename = input("filename:")

min\_support= float(input("Min support (from 0 to 1):"))

min\_confidence= float(input("Min confidence (from 0 to 1):"))

## import data

with open(filename, 'r', newline='') as f:

    reader = f.read()

    transactions = eval(reader)

##compute support

condition = True

min\_support\_count={}

combin\_num = 1

while condition == True:

    ##first step

    if combin\_num == 1:

        for i in items:

            count = 0

            for trans in transactions:

                if i in trans:

                    count = count + 1

            if count > min\_support\*20:

                min\_support\_count[i] = count

        ##create new items set, ignore those count is less than the mini support

        items\_more\_than\_support = []

        for key, value in min\_support\_count.items():

            items\_more\_than\_support.append(key)

        items = items\_more\_than\_support

        combin\_num = combin\_num + 1

    ##create combinations

    else:

        new\_items = []

        combinations\_result = []

        for pairs in combinations(items, combin\_num):

            combinations\_result.append([\*pairs])

        combin\_num = combin\_num + 1

        for i in combinations\_result:

            count = 0

            for trans in transactions:

                if all(item in trans for item in i):

                    count = count + 1

            if count > min\_support\*20:

                min\_support\_count[str(i)] = count

        if len(combinations\_result)==1 or combin\_num == 10:

            items = []

            for key, value in min\_support\_count.items():

                items.append(key)

            condition = False

## check support and confidence

result = []

for elements in items:

    if "[" in elements:

        result.append(eval(elements))

final\_result = []

for a in result:

    b = [list(c) for i in range(len(a)) for c in combinations(a, i+1)]

    b.remove(a)

    new\_list = []

    for i in range(int(len(b)/2)):

        new\_list.append([b[i], b[-(i+1)]])

    for i in range(len(new\_list)):

        new\_list.append([new\_list[i][1], new\_list[i][0]])

    final\_result = final\_result + new\_list

index\_to\_removed = []

for i in range(len(final\_result)):

    if confidence(final\_result[i][0], final\_result[i][1]) < min\_confidence:

        index\_to\_removed.append(i)

for i in index\_to\_removed:

    final\_result.pop(i)

for i in range(len(final\_result)):

    a = final\_result[i][0]

    b = final\_result[i][1]

    print(a, "->", b, "sup:", support(a+b), "conf:", confidence(a, b), "\n")