COMP7930 Big Data Analytics

Assignment 2 Data Clustering and Frequent Itemset

Students Name	ID
💡 LIU YONGYU	💡 19414617

- Lecturer: [Dr. Kevin Wang (mailto:kevinw@comp.hkbu.edu.hk)](mailto:kevinw@comp.hkbu.edu.hk)
- TA in-charge: [Kejing Yin (mailto:cskjyin@comp.hkbu.edu.hk)](mailto:cskjyin@comp.hkbu.edu.hk)

Assignment Due: 19/4/2020 (Sunday)

Submission: Print this `ipynb` file to a PDF and also attach the written work that you have done in the same PDF file.

Overview

There are two parts in the assignments. In the first part, you will need to perform some programming task to cluster the data set, *EV charging slot* which is abstracted from the Hong Kong government open data webpage: data.gov.hk. The data includes 223 electrical vehicle charging stations in Hong Kong. We want you to cluster the locations of these stations. Again, the intention for this is to equip you the ability of "calling" a library, rather than reimplement certain algorithms. Spot the symbol 💡 where actions are need to be taken. Thus, it is expected that you only need to spend a very few lines of code to achieve what you need to do. Shall you encounter any difficulty, please approach us on Piazza!

In the second part of the assignment, we will be practicing some data analytics algorithm with our bare hands. You are required to finish these questions on a separated paper or word documents and merge it with this notebook into a single PDF file.

Part 1 - Programming Task

Packages you should install

In assignment 1 you should have install several package already. On top of that, we are going to use the machine learning package `sklearn` . Again, type the following in your command prompt/terminal to install the package.

```
pip install sklearn
```

The packages required in this assignment would be:

- `jupyter notebook`
- `seaborn`
- `matplotlib`
- `pandas`
- `sklearn` *new package*

Again we are using Jupyter notebook, or the *notebook*, as our integrated development environment (IDE).

Instructions

Download the dataset `charging_slots.json` which contains the locations of electric vehicle charging stations in Hong Kong. Run the following code to read the data. Make sure you have placed `charging_slots.json` in the same directory as your notebook.

(To run the code below, click the cell and click `Cell > Run Cells` on the menu bar to run it. Or simple press `Ctrl-Enter`)

You may want to browse the data before going on. Do it by opening with `notepad` or type `more charging_slots.json` in your command prompt/terminal. This time we are handling `json` file instead of `csv` .

In [1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_json('charging_slots.json', orient='value', encoding='UTF8')
df.head()
```

Out[1]:

	district-s-en	location-en	address-zh	img	district-l-en	parking-no	district-s-zh	address-en	provider	type	di
1	Wong Tai Sin	Temple Mall North	九龍黃大仙龍翔道136號,黃大仙中心北館,三樓停車場,	/EV/PublishingImages/common/map/map_thumb/Entr...	Kowloon	320-322	黃大仙	Temple Mall North Carpark, Level 3,\n136 Lung ...	CLP	SemiQuick	
2	Yuen Long	Fu Shing Building	新界元朗西菁街9號富盛大廈停車場一樓	/EV/PublishingImages/common/map/map_thumb/Entr...	New Territories	33-35	元朗	Fu Shing Building Carpark, 1/F\n9 Sai Ching St...	CLP	SemiQuick	
3	Wong Tai Sin	Lok Fu Plaza Carpark	九龍黃大仙橫頭磡樂富中心地下停車場	/EV/PublishingImages/common/map/map_thumb/Entr...	Kowloon	67-69	黃大仙	Lok Fu Plaza Carpark, G/F\nWang Tau Hom, Wong ...	CLP	SemiQuick	
4	Kwun Tong	MegaBox	九龍九龍灣宏照道38號MegaBox地庫停車場	/EV/PublishingImages/common/map/map_thumb/Mega...	Kowloon	139-141	觀塘	MegaBox Carpark, B/F\n38 Wang Chiu Road, Kowl...	CLP	SemiQuick	
6	Kwai Tsing	Shek Lei Shopping Centre II	新界葵涌石籬邨石籬商場二期停車場四樓	/EV/PublishingImages/common/map/map_thumb/Entr...	New Territories	33-35	葵青區	Shek Lei Shopping Centre Phase II Carpark, 4/...	CLP	SemiQuick	

The above code read the data into the variable `df` as a **DataFrame**. A DataFrame can be considered as a row of data in Pandas. We printed the first five rows above by `df.head()` .

In [2]:
```python
df.describe()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.map_locations()

TypeError: unhashable type: 'list'


Exception ignored in: 'pandas._libs.index.IndexEngine._call_map_locations'
Traceback (most recent call last):
  File "pandas/_libs/hashtable_class_helper.pxi", line 1652, in pandas._libs.hashtable.PyObjectHashTable.map_
locations
TypeError: unhashable type: 'list'
```

Out[2]:

	district-s-en	location-en	address-zh	img	district-l-en	parking-no	district-s-zh	address-en	provider	type	dis
count	223	223	223	98	223	71	223	223	223	223	
unique	18	215	212	91	4	46	18	216	2	5	
top	Yau Tsim Mong	Tin Shing Court	新界沙田香港科學園科技大道西8-10號香港科技園二期地庫停車場	/EV/PublishingImages/common/map/map_thumb/Entr...	Kowloon	Tesla only	油尖旺	Hong Kong Science Park Carpark P2, B/F,\n8-10 ...	Others	Standard	
freq	31	2	2	2	81	21	31	2	180	114	

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 223 entries, 1 to 239
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   district-s-en  223 non-null    object
 1   location-en    223 non-null    object
 2   address-zh     223 non-null    object
 3   img            98 non-null     object
 4   district-l-en  223 non-null    object
 5   parking-no     71 non-null     object
 6   district-s-zh  223 non-null    object
 7   address-en     223 non-null    object
 8   provider       223 non-null    object
 9   type           223 non-null    object
 10  district-l-zh  223 non-null    object
 11  lat-long       223 non-null    object
 12  location-zh    223 non-null    object
dtypes: object(13)
memory usage: 24.4+ KB
```

The command `df.info()` explain in more details about what are the types of these columns. These columns can be understood as

Column Name	Explanation
district-s-en	Name of the district in English
location-en	Name of the location of the charging slot in English
address-zh	Detailed address of the charging slot in Chinese
img	URL to the image of the charing slot
district-l-en	Name of the region in English
parking-no	Parking space number
district-s-zh	Name of the district in Chinese
address-en	Detailed address of the charging slot in English
provider	Provider of the charging slot
type	Type of the charging slot
district-l-zh	Name of the region in Chinese
lat-long	Latitude and Longitude of the charging slot
location-zh	Name of the location of the charging slot in Chinese

💡 Counting numbers of park spaces

If we look at the column `parking-no` , some of the location has more than one parking slot. e.g.

In [4]:
```
df.loc[1]
```

Out[4]:
```
district-s-en                                                Wong Tai Sin
location-en                                             Temple Mall North
address-zh                             九龍黃大仙龍翔道136號,黃大仙中心北館,三樓停車場,
img              /EV/PublishingImages/common/map/map_thumb/Entr...
district-l-en                                                     Kowloon
parking-no                                                        320-322
district-s-zh                                                         黃大仙
address-en       Temple Mall North Carpark, Level 3,\n136 Lung ...
provider                                                              CLP
type                                                            SemiQuick
district-l-zh                                                          九龍
lat-long                                [22.342590332, 114.1907196045]
location-zh                                                      黃大仙中心北館
Name: 1, dtype: object
```

which says 320-322. That contains 3 slots. We want to add a field `no_slot` to the dataframe which say how many slots are there. Unfortunately the field isn't always that beautiful.

In [5]:
```python
df['parking-no'].to_list()
```

Out[5]:
```
['320-322',
 '33-35',
 '67-69',
 '139-141',
 '33-35',
 '1001-1003',
 '79-81',
 '1250-1251, 1305',
 'B1-B3',
 '22-24',
 '33,35',
 'P15-P16',
 '107-108',
 '189-190',
 'B111',
 'C2',
 'D042 - D052, D106 - D112',
 '3177-3186, 3189-3206',
 '73-75',
```

In [6]:
```python
# Complete the function count_parking so that it will returns the number of slots

import re

# This function helps you to computes like 73-75 = 3 or 042-052 = 11.
# But it cannot handle D042-D052
# You may use this function to implement count_parking
def count_k(s):
    e = s.split('-')
    if len(e) < 2:
        return 1
    return int(e[1]) - int(e[0]) + 1


def count_parking(s):
    if s is None:
        return 1
    total = 0
    # Take away all letter from s that is not ',' or a number or '-'.
    s = re.sub('[a-zA-Z .]', '', s)
    if s == '':
        total = 1
    # Then split the string s into a list of string by ','. Sometimes we call this steps tokenize
    if s.__contains__(','):
        s = s.split(',')
    # Accumulate the answer obtained from each token.
        for num in range(len(s)):
            total = total + count_k(s[num])
    else:
        total = count_k(s)
    # Finally return the total value of your count.
    return total


df['no_slot'] = df['parking-no'].apply(count_parking)
df[['no_slot', 'parking-no']].values
```

Out[6]:
```
array([[3, '320-322'],
       [3, '33-35'],
       [3, '67-69'],
       [3, '139-141'],
       [3, '33-35'],
       [3, '1001-1003'],
       [3, '79-81'],
       [3, '1250-1251, 1305'],
       [3, 'B1-B3'],
       [3, '22-24'],
       [2, '33,35'],
       [2, 'P15-P16'],
       [2, '107-108'],
       [2, '189-190'],
       [1, 'B111'],
       [1, 'C2'],
       [18, 'D042 - D052, D106 - D112'],
       [28, '3177-3186, 3189-3206'],
       [3, '73-75'],
```

In [7]:
```python
df['no_slot'].sum()
```

Out[7]:
```
337
```

💡 Extract the latitudes and longitudes

We use the latitudes and the longitudes of the charging slots to conduct the clustering. First, we need to extract the latitudes and the longitudes from the dataframe into the following format:

A python list with 223 rows and 2 cols like `mylist`

In [8]:
```python
# complete the following line
mylist = df['lat-long'].to_list()
```

In [9]:
```python
mylist
```

Out[9]:
```
[[22.342590332, 114.1907196045],
 [22.4420719147, 114.027671814],
 [22.3386573792, 114.1861038208],
 [22.3203277588, 114.2085266113],
 [22.3658618927, 114.14012146],
 [22.2882328033, 113.94190979],
 [22.5029144287, 114.1275863647],
 [22.2956161499, 114.1693572998],
 [22.4504833221, 114.1608352661],
 [22.4922847748, 114.1389007568],
 [22.3152523041, 114.1625061035],
 [22.3381347656, 114.1739120483],
 [22.3194198608, 114.1565704346],
 [22.3718738556, 113.993019104],
 [22.3010005951, 114.1679153442],
 [22.3752117157, 114.111328125],
 [22.4262580872, 114.2098770142],
 [22.4493846893, 114.0018539429],
 [22.3821163177, 114.1900787354],
```

💡 Then, for each row `r` in the dataframe `df`, we will append `r['no_slot'] - 1` coordinates to the list.

In [10]:
```python
for i, r in df.iterrows():
    # put your code here
    ns = r['no_slot']-1
    ll = r['lat-long']
    if ns > 0:
        for num in range(0, ns):
            num_ll = ll
            mylist.append(num_ll)
```

In [11]:
```python
mylist
```

Out[11]:
```
[[22.342590332, 114.1907196045],
 [22.4420719147, 114.027671814],
 [22.3386573792, 114.1861038208],
 [22.3203277588, 114.2085266113],
 [22.3658618927, 114.14012146],
 [22.2882328033, 113.94190979],
 [22.5029144287, 114.1275863647],
 [22.2956161499, 114.1693572998],
 [22.4504833221, 114.1608352661],
 [22.4922847748, 114.1389007568],
 [22.3152523041, 114.1625061035],
 [22.3381347656, 114.1739120483],
 [22.3194198608, 114.1565704346],
 [22.3718738556, 113.993019104],
 [22.3010005951, 114.1679153442],
 [22.3752117157, 114.111328125],
 [22.4262580872, 114.2098770142],
 [22.4493846893, 114.0018539429],
 [22.3821163177, 114.1900787354],
```

Finally we will turn it into a numpy element `X` which is ready to be clustered and visualized.

In [12]:
```python
X = np.array(mylist)
print(type(X), X.shape)
```

```
<class 'numpy.ndarray'> (337, 2)
```

In [13]:
```
X  # X is now a 2-D array.
```

Out[13]:
```
array([[ 22.34259033, 114.1907196 ],
       [ 22.44207191, 114.02767181],
       [ 22.33865738, 114.18610382],
       [ 22.32032776, 114.20852661],
       [ 22.36586189, 114.14012146],
       [ 22.2882328 , 113.94190979],
       [ 22.50291443, 114.12758636],
       [ 22.29561615, 114.1693573 ],
       [ 22.45048332, 114.16083527],
       [ 22.49228477, 114.13890076],
       [ 22.3152523 , 114.1625061 ],
       [ 22.33813477, 114.17391205],
       [ 22.31941986, 114.15657043],
       [ 22.37187386, 113.9930191 ],
       [ 22.3010006 , 114.16791534],
       [ 22.37521172, 114.11132812],
       [ 22.42625809, 114.20987701],
       [ 22.44938469, 114.00185394],
       [ 22.38211632, 114.19007874],
```

Clustering using K-Means

The K-means algorithm clusters data into several seperate groups by minimizing a distance measure within each cluster. We use the `sklearn` package to do the clustering. You may refer to the [K-Means document (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html) for more details on the implementation provided by this package.

In [14]:
```
# First, we import the KMeans class from the sklearn package.
from sklearn.cluster import KMeans
```

In [15]:
```
kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
```

In [16]:
```
kmeans.cluster_centers_
```

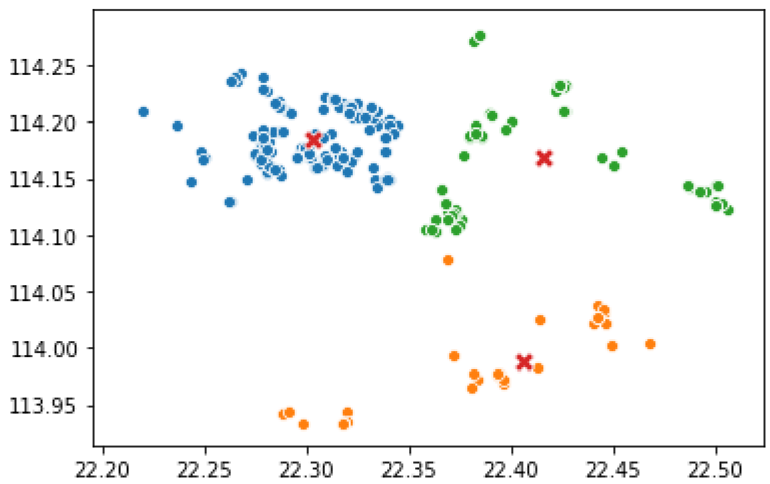
Out[16]:
```
array([[ 22.30316053, 114.18519197],
       [ 22.405908  , 113.98810079],
       [ 22.41616649, 114.16882621]])
```

In [17]:
```
kmeans.labels_
```

Out[17]:
```
array([0, 1, 0, 0, 2, 1, 2, 0, 2, 2, 0, 0, 0, 1, 0, 2, 2, 1, 2, 0, 0, 1,
       0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 1, 1, 1, 1, 1, 1, 1, 2, 0, 0, 0, 2, 2, 0, 1, 0, 0, 1, 2, 2, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 2, 1, 0, 0, 1, 0, 2, 0, 0, 2, 1,
       2, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       2, 0, 0, 1, 0, 0, 0, 2, 0, 2, 0, 1, 0, 1, 0, 1, 0, 1, 1, 2, 2, 2,
       2, 2, 0, 0, 0, 1, 1, 0, 0, 0, 0, 2, 2, 1, 1, 2, 2, 0, 0, 2, 2, 2,
       2, 0, 0, 0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 2, 2, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 0,
       0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 2, 2, 0, 2, 1, 1, 1, 1, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 0, 0], dtype=int32)
```

In [18]:
```
for i in range(3):
    sns.scatterplot(X[kmeans.labels_==i, 0], X[kmeans.labels_==i, 1])
sns.scatterplot(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker='X', s=100)
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1a188ba690>



Clustering using DBSCAN

The DBSCAN is another useful algorithm for clustering. Similarly, we use the `sklearn` package. You may refer to the [DBSCAN document (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluster.DBSCAN)](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluster.DBSCAN) for more details on the implementation provided by this package.

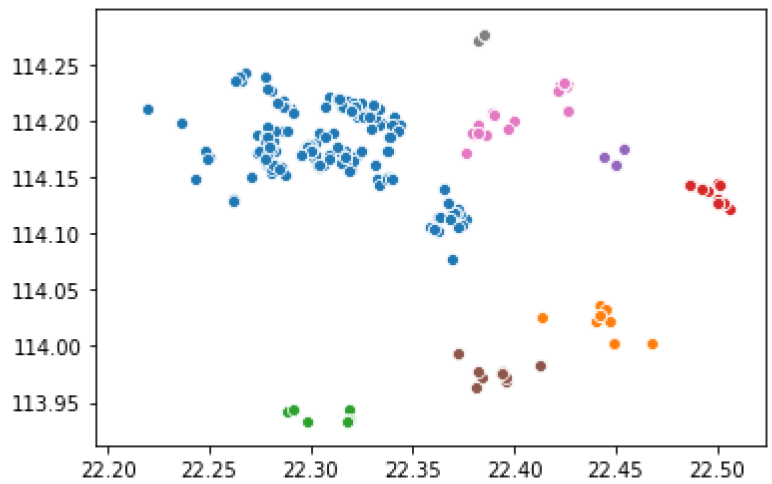
In [19]:
```
# First, we import the KMeans class from the sklearn package.
from sklearn.cluster import DBSCAN
```

In [20]:
```
dbscan = DBSCAN(eps=0.03, min_samples=3).fit(X)
```

In [21]:
```
n_clusters = dbscan.labels_.max()+1
```

In [22]:
```
# first plot the noisy samples
sns.scatterplot(X[dbscan.labels_==-1, 0], X[dbscan.labels_==-1, 1], color='grey')

# then plot the clusters one-by-one
for i in range(n_clusters):
    sns.scatterplot(X[dbscan.labels_==i, 0], X[dbscan.labels_==i, 1])
```



Part 2 - Written Assignment

You are allowed to write it on paper or type it properly in a word documents or even type it in this notebook. No matter what you plan to do, please combine your work into one single PDF so that our TA will be able to grade your work.

Consider the following set of transactions which will be used in Q1 and Q2:

Transaction ID (TID)	Items
1	a,b,c
2	a,c,d
3	d,e,f
4	e, f
5	a,c,e,f
6	a,b,c,f
7	b,c,d,e
8	e,f
9	a,c,d,f
10	d,f

Q1. (**FP-growth**) FP-growth algorithm to find all frequent pattern with minimum support = 3. To verify your work, there are 14 of them. Show your steps.

Q2. (**Apriori**) Find all frequent 3-itemsets **candidates** using Apriori algorithm with alternate $F_{k-1} \times F_{k-1}$ Method mentioned page 45 of the chap 4 slides. To save your work, assume we have found all 2-itemsets already (you are allowed to reuse the result found in Q1). Then, perform candidate pruning over your result.

Q3. (**Min-Hash**) Given the set of shingles {A,B,C,D,E,F,G,H} and the following three documents D_1, D_2, D_3, compute the MinHash for them against each of the permutation p_1, p_2, p_3. Calculate the Jaccard similarity between these documents and the similarity of MinHash of these documents.

Documents	D_1	D_2	D_3
Shingle	{B,D,F,H}	{A,B,H}	{E,F}

Documents	D_1	D_2	D_3
A	0	1	0
B	1	1	0
C	0	0	0
D	1	0	0

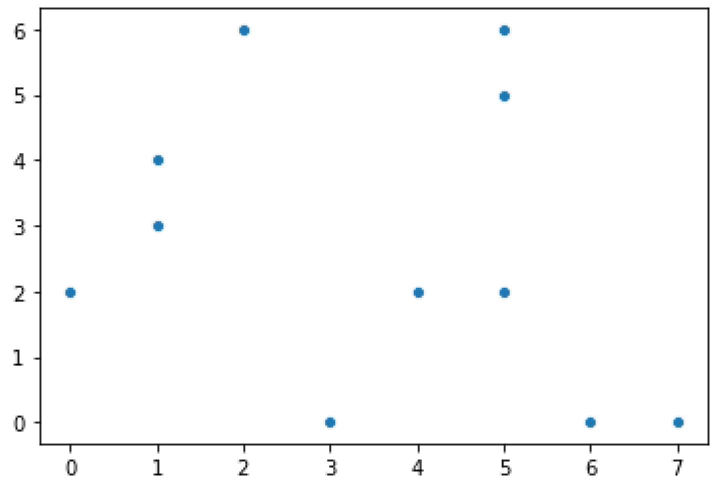
Documents	D_1	D_2	D_3
E	0	0	1
F	1	0	1
G	0	0	0
H	1	1	0

Permutation	Order
P_1	BDEFHGAC
P_2	CDEFABHG
P_3	ACBDGFEH

Q4. (**MST**) Create 3 clusters using minimum spanning tree (MST) with the following coordinates. Please reproduce the diagram in your solution.

In [23]:
```
points = np.array([[1,3],[1,4], [0,2],[2,6],[3,0],[4,2],[5,2],[5,5],[6,0],[5,6],[7,0]] )
sns.scatterplot(points[:,0],points[:,1])
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1a181667d0>



Q1: (**FP-growth**) FP-growth algorithm to find all frequent pattern with minimum support = 3. To verify your work, there are 14 of them. Show your steps.

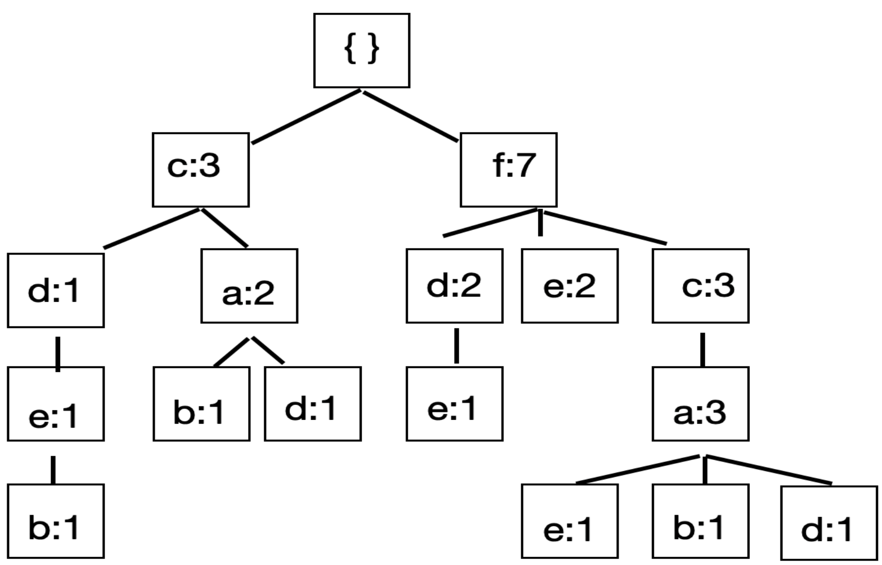
Scan DB once, find frequent 1-itemset and sort it:

Item	frequency head
f	7
c	6
a	5
d	5
e	5
b	3

F-list = f-c-a-d-e-b

TID	Items	ordered
1	a,b,c	c,a,b
2	a,c,d	c,a,d
3	d,e,f	f,d,e
4	e, f	f,e
5	a,c,e,f	f,c,a,e
6	a,b,c,f	f,c,a,b
7	b,c,d,e	c,d,e,b
8	e,f	f,e
9	a,c,d,f	f,c,a,b
10	d,f	f,d

Scan DB again, construct FP-tree:



Find frequent itemsets:

Patterns containing	Conditional pattern base	Frequent
f	null	{f}

c	f:3	{c, fc}
a	c:2, fc:3	{a, ca, fa, fca}
d	c:1, ca:1, f:2, fca:1,	{d, cd, fd}
e	cd:1, fd:1, f:2	{e, fe}
b	cde:1, ca:1, fca:1,	{b, cb}

There are 14 frequent patterns with minimum support = 3:
{f, c, a, d, e, b, fc, ca, fa, cd, fd, fe, cb, fca}.

Q2: (**Apriori**) Find all frequent 3-itemsets **candidates** using Apriori algorithm with alternate $F_{k-1} \times F_{k-1}$Fk−1×Fk−1 Method mentioned page 45 of the chap 4 slides. To save your work, assume we have found all 2-itemsets already (you are allowed to reuse the result found in Q1). Then, perform candidate pruning over your result.

According to Q1:
F_2 = {fc, ca, fa, cd, fd, fe, cb} is the set of frequent 2-itemsets.
Merge each of them from F_2 to generate the set of candidates 3-itemset:
L_3 = {fca, fcd, fda, fec, fcb, cad, cab, fae, cdb, fde}
Candidate pruning:
Prune {fda} because (da) is infrequent.
Prune {fec}because {ec} is infrequent.
Prune {fcb} because {fb} is infrequent.
Prune {cad} because {ad} is infrequent.
Prune {cab} because {ab} is infrequent
Prune {fae} because {ae} is infrequent.
Prune {cdb} because {cd} is infrequent.
Prune {fde} because {de} is infrequent.
Therefore, after candidates pruning: **candidate 3-itemsets**: L_3 = {fca, fcd}
Support counting:
Count the support by scanning the DB: {fca:3}, {fcd:1}
Candidate elimination
Eliminate candidates {fcd}
Therefore, frequent 3-itemsets is{fca}

Q3: (**Min-Hash**) Given the set of shingles {A,B,C,D,E,F,G,H} and the following three documents D_1,D_2,D_3, compute the MinHash for them against each of the permutation p_1,p_2,p_3 Calculate the Jaccard similarity between these documents and the similarity of MinHash of these documents.

Documents	D_1	D_2	D_3
Shingle	{B,D,F,H}	{A,B,H}	{E,F}

Documents	D_1	D_2	D_3
A	0	1	0
B	1	1	0
C	0	0	0
D	1	0	0
E	0	0	1
F	1	0	1
G	0	0	0
H	1	1	0

Permutation	Order
P_1	BDEFHGAC
P_2	CDEFABHG
P_3	ACBDGFEH

As for p_1, BDEFHGAC,

p_1	D_1	D_2	D_3
B (1)	1	1	0
D (2)	1	0	0
E (3)	0	0	1
F (4)	1	0	1
H (5)	1	1	0
G (6)	0	0	0
A (7)	0	1	0
C (8)	0	0	0

As for p_2, CDEFABHG,

p_2	D_1	D_2	D_3
C (1)	0	0	0
D (2)	1	0	0
E (3)	0	0	1
F (4)	1	0	1
A (5)	0	1	0
B (6)	1	1	0
H (7)	1	1	0
G (8)	0	0	0

As for p_3, ACBDGFEH,

p_3	D_1	D_2	D_3
A (1)	0	1	0
C (2)	0	0	0
B (3)	1	1	0
D (4)	1	0	0
G (5)	0	0	0

F (6)	1	0	1
E (7)	0	0	1
H (8)	1	1	0

Signature matrix:

	D_1	D_2	D_3
p_1	1	1	3
p_2	2	5	3
p_3	3	1	6

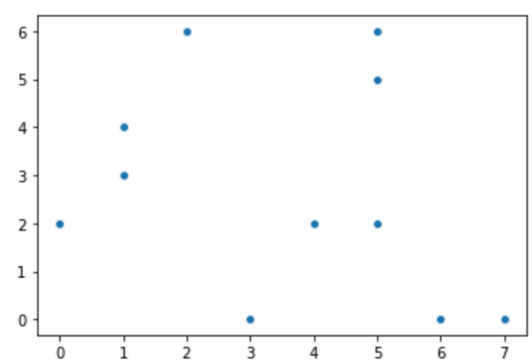
Calculate the similarities:

	D_1-D_2	D_1-D_3	D_2-D_3
Jaccard similarity	0.4	0.2	0
MinHash similarity	0.33	0	0

Q4: (**MST**) Create 3 clusters using minimum spanning tree (MST) with the following coordinates. Please reproduce the diagram in your

```
points = np.array([[1,3],[1,4],  [0,2],[2,6],[3,0],[4,2],[5,2],[5,5],[6,0],[5,6],[7,0]] )
sns.scatterplot(points[:,0],points[:,1])
```

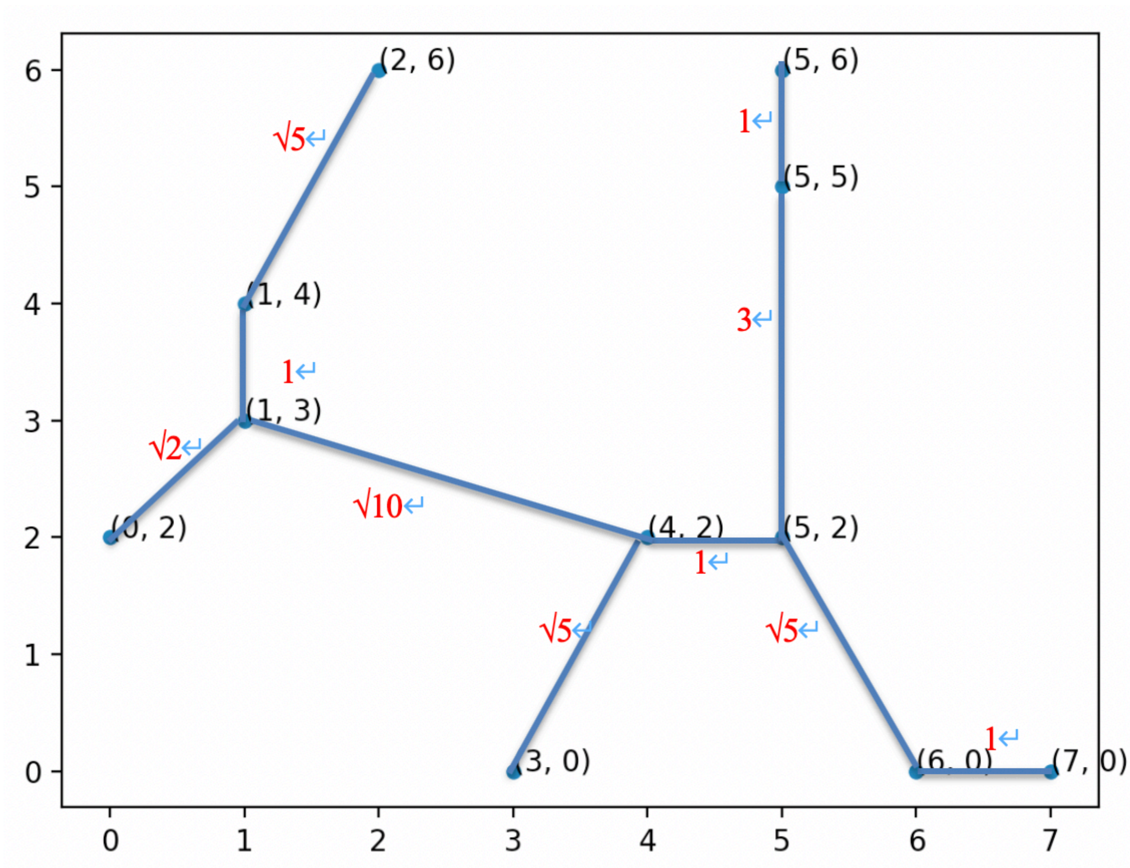
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a26f105d0>
```



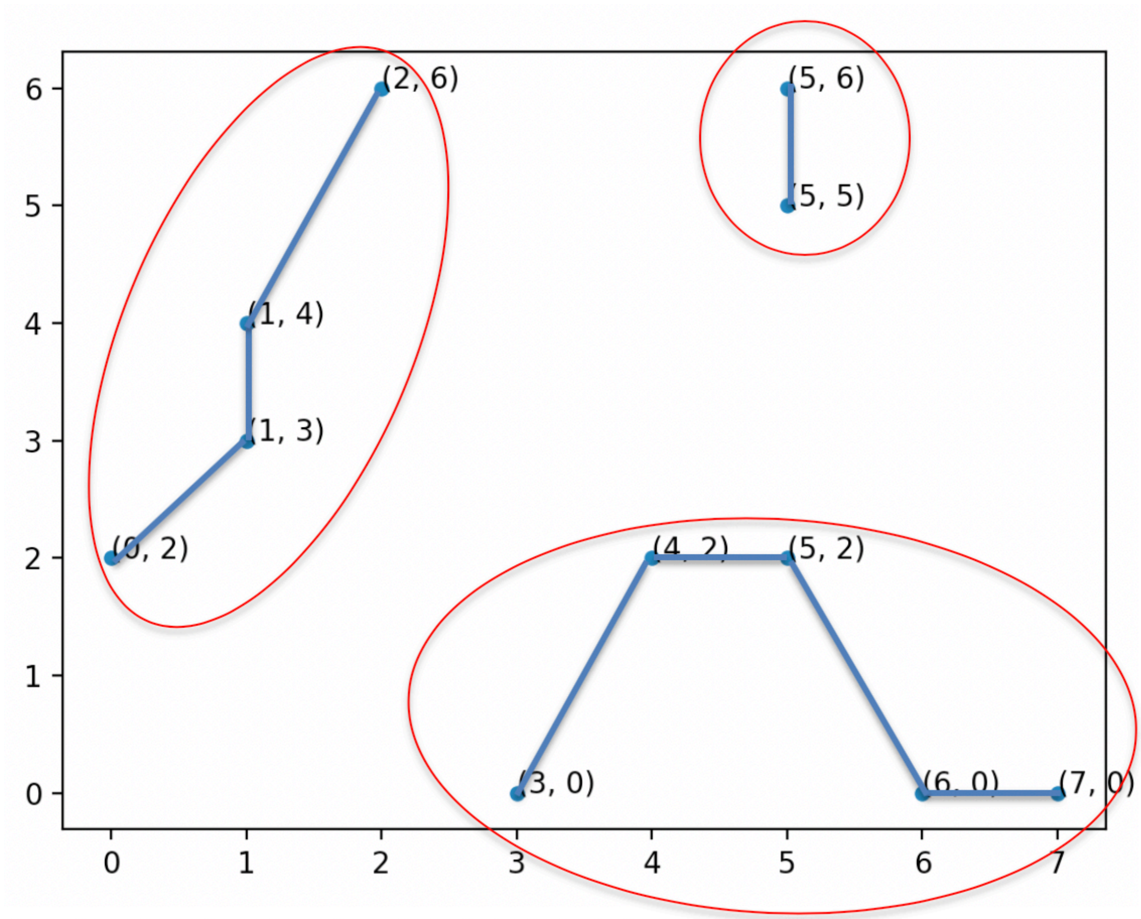
Calculate the distance between each point:

(1,3)	0										
(1,4)	1	0									
(0,2)	$\sqrt{2}$	$\sqrt{5}$	0								
(2,6)	$\sqrt{10}$	$\sqrt{5}$	$2\sqrt{5}$	0							
(3,0)	$\sqrt{13}$	$2\sqrt{5}$	$\sqrt{13}$	$\sqrt{37}$	0						
(4,2)	$\sqrt{10}$	$\sqrt{13}$	4	$2\sqrt{5}$	$\sqrt{5}$	0					
(5,2)	$\sqrt{17}$	$2\sqrt{5}$	5	5	$2\sqrt{2}$	1	0				
(5,5)	$2\sqrt{5}$	$\sqrt{17}$	$\sqrt{34}$	$\sqrt{10}$	$\sqrt{29}$	$\sqrt{10}$	3	0			
(6,0)	$\sqrt{34}$	$\sqrt{41}$	$2\sqrt{10}$	$2\sqrt{13}$	3	$2\sqrt{2}$	$\sqrt{5}$	$\sqrt{26}$	0		
(5,6)	5	$2\sqrt{5}$	$\sqrt{41}$	$\sqrt{9}$	$2\sqrt{10}$	$\sqrt{17}$	4	1	$\sqrt{37}$	0	
(7,0)	$3\sqrt{5}$	$2\sqrt{13}$	$\sqrt{53}$	$\sqrt{61}$	4	$\sqrt{11}$	$2\sqrt{2}$	$\sqrt{29}$	1	$2\sqrt{10}$	0
distance	**(1,3)**	**(1,4)**	**(0,2)**	**(2,6)**	**(3,0)**	**(4,2)**	**(5,2)**	**(5,5)**	**(6,0)**	**(5,6)**	**(7,0)**

Generate the minimum spanning tree (MST):



Erase two longest lines √10 (point (1,3) and point (4.2)) and 3(point (5,2) and point (5,5)) and generate 3 clusters:



Cluster1: (0,2), (1,3), (1,4), (2,6),
Cluster2: (5,5), (5,6),
Cluster3: (3,0), (4,2), (5,2), (6,0), (7,0),