671 Homework 1: Itemset Mining

uniqname: ellali

In [41]:
```
# import packages
```

In [42]:
```
# !pip install mlxtend
```

In [43]:
```
import sklearn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
%matplotlib inline

import mlxtend
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

In [44]:
```
pd.set_option('display.max_colwidth', None)
```

Part 1: Data Exploration [20 points]

1.1 Load & transform the Twitter emoji dataset [11 points]

a) [3 points] First, read in the itemsets_data/food_drink_emoji_tweets.txt file and create a pandas DataFrame with the contents. You should find that every line of the data is a Tweet.

In [45]:
```
tweets = pd.read_csv('itemsets_data/food_drink_emoji_tweets.txt',
                     sep="\t", names=['tweet_content'])
```

In [46]:
```
tweets.head(10)
```

Out[46]:

	tweet_content
0	RT @CalorieFixess: 🍗🍫🍔🍒 400 Calories https://t.co/9OaPOWUSht
1	RT @1_F_I_R_S_T: _ 🍋[1]⟩ Grow your account fast! [2]⟩ 🍑Retweet Now!!! 🍉 🍇[3]⟩ Follow all Retweets [4]⟩ 🍊Follow back everybody 🍓[5]⟩ Follow me & @1f_sts🍇 [6]...
2	RT @LegendDeols: 👉👉👉G€T Ready to dance💃💃💃💃💃💃 With #LittleLittlepag 🥃🍸🍸🍸🥃 with Da$HING DEOLS 👧👧👧👧👧. Song 🎧🎧🎧out today at 🔟 am .@ypdphirse...
3	@britch_x Hubby's friend bought us Wendy's-cheeseburger (no onions), fries and a Coke. 🍔🍟
4	RT @DAILYPUPPIES: Workout partner ☕💪😍 https://t.co/3p0VZs6RKp
5	RT @Yggdrasillrealm: congrats fam!! #1 shanawa of @maymayentrata07 & #3 bmg of @Barber_Edward_ ❤️❤️🙌🍾 #MORPinoyBiga10 @mor1019 BMG by Edwa...
6	@martinmozotegui 👹🐏🏠🐑🐘🍦🕷🍆1⃣⭐🍋🐏 es yogur helado
7	RT @Tangerine2525: FOLLOW EVERYONE WHO LIKES THIS 🍊🍊🍊🍇
8	RT @sweetrosepetal: @YaOnlyLivvOnce Me EVERY day, girl. 🍩🍦🍙🍧🍪🍰🍘🥤
9	RT @NAQI5110: 🤠Bam Bam Bam🤠 🍤Lets Gain Followers🍤 🍻Follow @iefun 🍪Gain With #NAKI #IFBDrive #1DDrive #Jen2Gain #BossLady #JerriTricks #ieFu...

b) [3 points] Using this DataFrame, extract the emojis that appear in each Tweet as an itemset. For this assignment, we are only interested in emojis that are food and drink. As such, you are supplied with the following emoji_set to filter to food and drink emojis:

In [47]:
```
emoji_set = {🥫, 🍾, 🍏, 🍈, 🍯, 🍗, 🍪, 🌰, 🥯, 🍋, 🌯, 🍧, 🥚, 🍎, 🍍, 🍤,
🍡, 🍲, 🍊, 🍘, 🍆, 🌶, 🥐, 🌮, 🍢, 🍥, 🍳, 🥔, 🍦, 🍉, 🍱, 🍄, 🎂, 🧀,
🥩, 🍺, 🦀, 🥦, 🍗, 🍒, 🍤, 🍣, 🥝, 🥙, 🍖, 🍠, 🍬, 🍷, 🍾,
🍛, 🧂, 🥪, 🍔, 🥒, 🥑, 🍶, 🍫, 🥃, 🥗, 🍕, 🍟, 🥕, 🧁, 🍋, 🍰, 🥛,
🍗, 🍝, 🥤, 🍮, 🍿, 🍅, 🥂, 🍸, 🥥, 🔪, 🍌, 🍝, 🍲, 🧂, 🍭, 🍑, 🍹,
🍇, 🥨, 🍱, 🦞, ☕, 🌶, 🍺, 🍳, 🥒, 🍣, 🌮, 🍾, 🍖, 🍽, 🍓, 🦑, 🍩, 🥖}
```

In [48]:
```
# extract emoji for each sentence
def extract_emojis(sentence):
    return np.unique([x for x in sentence if x in emoji_set])



# apply to df
```

```
tweets["food_drink_emoji"] = tweets["tweet_content"].apply(extract_emojis)
tweets.head(5)
```

Out[48]:

	tweet_content	food_drink_emoji
0	RT @CalorieFixess: 🍗🍉🍔🍒 400 Calories https://t.co/9OaPOWUSht	[🍉, 🍒, 🍔, 🍗]
1	RT @1_F_I_R_S_T: _ 🍋[1]⟩ Grow your account fast! [2]⟩ 🍊Retweet Now!!! 🍉 🍇[3]⟩ Follow all Retweets [4]⟩ 🍑Follow back everybody 🍓[5]⟩ Follow me & @1f_sts🍇 [6]...	[🍇, 🍋, 🍉, 🍑, 🍊, 🍓]
2	RT @LegendDeols: 👉👉👉G€T Ready to dance💃🕺💃🕺💃 With #LittleLittlepag 🥃🍸🍸🍸🥃 with Da$HING DEOLS 🌼🌼🌼🌼🌼. Song 🎧🎧🎧out today at 🔟 am .@ypdphirse...	[🍸, 🥃]
3	@britch_x Hubby's friend bought us Wendy's-cheeseburger (no onions), fries and a Coke. 🍔🍟	[🍔, 🍟]
4	RT @DAILYPUPPIES: Workout partner ☕💪🤩 https://t.co/3p0VZs6RKp	[☕, 💪]

c) [5 points] Next, as we recommend utilizing the mlxtend package in this assignment to perform frequent itemset mining, you will need to transform the data. The mlxtend package requires that the itemsets be transformed into a matrix before being passed to its APIs, where each row represents an itemset and each column represents an item. Within this matrix, each cell encodes whether an item is in an itemset or not. We recommend the MultiLabelBinarizer function in scikit-learn to implement this transformation. Here is the documentation for the function: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MultiLabelBinarizer.html

In [49]:
```
from sklearn.preprocessing import MultiLabelBinarizer
```

In [50]:
```
# tranform to matrix
mlb = MultiLabelBinarizer()
emoji_item_matrix = mlb.fit_transform(tweets["food_drink_emoji"])
emoji_item_matrix
```

Out[50]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [1, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

In [51]:
```
# list(mlb.classes_)
```

In [52]:
```
# matrix to dataframe


emoji_item_matrix_df = pd.DataFrame(emoji_item_matrix, columns=mlb.classes_)
emoji_item_matrix_df.head(5)
```

Out[52]:

	☕	🥖	🌮	🍗	🍔	🌶	🌽	🍄	🍅	🍆	...	🥭	🍍	🥥	🦀	🦐	🦑	🦞	🧀	🧁	🧂
0	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 105 columns

1.2 Exploratory Data Analysis (EDA) [9 points]

[3 points] How many different emojis are used in the dataset?
[3 points] How many emojis are used in a Tweet, on average? What does the distribution look like? (hint: plot the distribution!)
[3 points] Which emojis are most popular (i.e., used most frequently) in the dataset? Please return the top 5.

In [53]:
```
print("Different emojis in the dataset:", len(mlb.classes_))
```

```
Different emojis in the dataset: 105
```

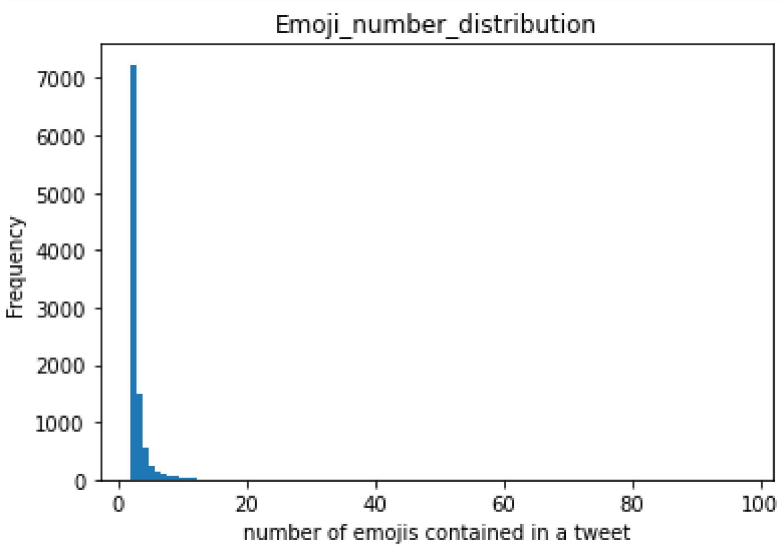
In [54]:
```
mean_emoji_num = tweets["food_drink_emoji"].apply(len).mean()
print("Average emojis are used in a Tweet", mean_emoji_num)
```

```
Average emojis are used in a Tweet 2.6340630648724646
```

In [55]:
```
# distirbution


emoji_num_distr = tweets["food_drink_emoji"].apply(len)
emoji_num_distr.plot(kind='hist', bins=100, title="Emoji_number_distribution")
plt.xlabel('number of emojis contained in a tweet')
```

```
print(emoji_num_distr.value_counts())
```

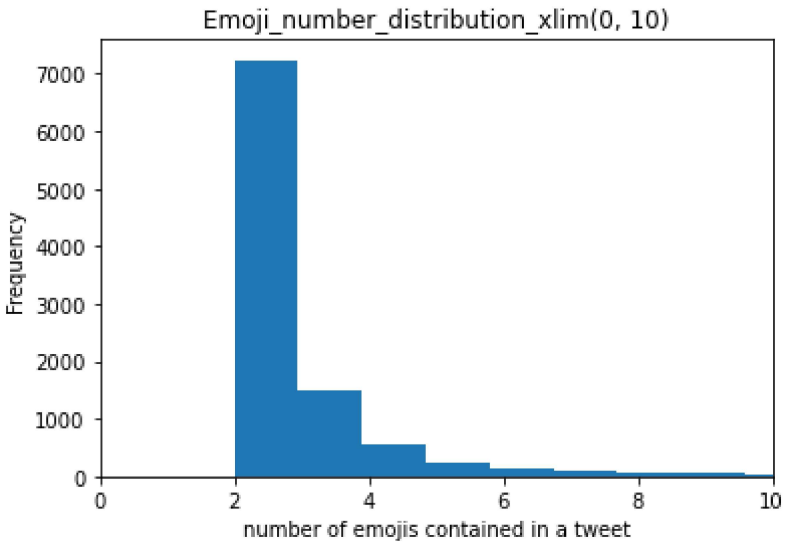
```
2     7237
3     1494
4      559
5      252
6      123
7       90
8       65
9       46
11      20
12      12
10      12
13      10
14       7
17       7
15       5
24       4
23       2
20       2
16       2
51       1
19       1
43       1
18       1
32       1
21       1
22       1
53       1
97       1
Name: food_drink_emoji, dtype: int64
```



here, we can see that in a tweet, the number of emojis are usually 0~10, we can also have a look of xlim(0, 10):

In [56]:
```
emoji_num_distr.plot(kind='hist', bins=100,
                     title="Emoji_number_distribution_xlim(0, 10)")
plt.xlim(0, 10)
plt.xlabel('number of emojis contained in a tweet')
```

Out[56]:
```
Text(0.5, 0, 'number of emojis contained in a tweet')
```



This distribution looks like a power-law distribution.

In [57]:
```
# most popular emojis (the emojis show up in the most total tweets)

top_5_pop_emojis = emoji_item_matrix_df.sum(
    axis=0).sort_values(ascending=False).head(5)

print("Top 5 emojis are: ", top_5_pop_emojis)
```

```
Top 5 emojis are:  🍺    1809
🍔    1481
🍔    1382
🍾    1078
🥂    1028
dtype: int64
```

In [ ]:

```
```

Part 2: The Apriori Algorithm

Now, it is time to apply the Apriori algorithm to the emoji dataset. The documentation for the apriori function can be found at http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/.

a) [10 points] Using apriori, create a function, `emoji_frequent_itemsets`, to find all the frequent k-itemsets with a minimal support of `min_support` in the emoji dataset. In other words, k and min_support should be arguments that are passed when the function is called, in addition to the matrix itself.

Your function should return a Pandas DataFrame object with two columns:

- The first one is named `support` and stores the support of the frequent itemsets.
- The second column is named `itemsets` and stores the frequent itemset as a frozenset (the default return type of the apriori API).

Make sure that you are only returning the frequent itemsets that have the specified number of emojis (k).

b) [5 points] Using this function, find all frequent 3-itemsets with a min support of 0.007.

In [58]:

```
def emoji_frequent_itemsets(matrix, k, min_support):

    frequent_itemsets = apriori(matrix, min_support, use_colnames=True)
    frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(len)

    k_itemsets = frequent_itemsets[frequent_itemsets["length"] == k][[
        "support", 'itemsets']].reset_index(drop=True)

    return k_itemsets



# frequent 3-itemsets (with a min support of 0.007)
emoji_frequent_itemsets(emoji_item_matrix_df, 3, 0.007)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types resu
lt in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
```

Out[58]:

	support	itemsets
0	0.007833	(🍉, 🍊, 🍇)
1	0.009239	(🍕, 🍔, 🍟)
2	0.007030	(🍭, 🍗, 🥫)
3	0.011749	(🍸, 🍷, 🍹)
4	0.007532	(🍸, 🍷, 🍺)
5	0.007632	(🍸, 🍷, 🍻)
6	0.007230	(🍸, 🍷, 🥂)
7	0.007230	(🍷, 🍹, 🍺)
8	0.007732	(🍷, 🍹, 🍻)
9	0.007030	(🥂, 🍷, 🍹)
10	0.007732	(🥂, 🍷, 🍾)
11	0.007632	(🍸, 🍹, 🍺)
12	0.007532	(🍸, 🍹, 🍻)
13	0.007230	(🍸, 🍾, 🍹)
14	0.007933	(🍸, 🍹, 🥂)
15	0.010544	(🥂, 🍩, 🍾)

2.1 Apriori Algorithm under the Hood

2.1.1 Candidate Generation

A critical step of the Apriori algorithm is candidate generation. That is, candidate (k+1)-itemsets should be generated from frequent k-itemsets. In the following exercise, we want you to generate candidate 3-itemsets based on the frequent 2-itemsets.

a) [10 points] You will need to construct a `generate_candidate_3_itemsets` function, which takes in a list of frequent 2-itemsets and returns a list of the candidate 3-itemsets ("candidate" means that they may or may not be frequent). Please represent each itemset as a `set` in Python. Make sure that for each candidate 3-itemset in your returned list, at least one of its size-2 subset is a frequent 2-itemset, and your list does not contain duplicated itemsets.

We have prepared the frequent 2-itemsets for you, which you can load from the file named itemsets_data/food_emoji_frequent_2_itemsets.csv. We will evaluate your function by feeding in the loaded 2-itemsets and examining the return value. Note that the loaded 2-itemsets are different from what you will get with the `emoji_frequent_itemsets` function you implemented in the previous part, as we have eliminated the drink-related emojis to make the exercise more trackable.

You will receive full points for this part of the question if (1) every candidate 3-itemset in your returned list is a superset of at least one frequent 2-itemset, (2) every 3-itemset that has a frequent size-2 subset is already in your list, and (3) your list does not contain duplicated sets.

In [59]:
```
freq_2_itemsets_df = pd.read_csv(
    'itemsets_data/food_emoji_frequent_2_itemsets.csv', names=['itemsets'])
freq_2_itemsets_df.head()
```

Out[59]:

	itemsets
0	🥖🍔
1	🥖🍕
2	🍔🌮
3	🍕🌮
4	🍆🍑

In [60]:
```
# trun series to set
freq_2_itemsets_df['itemsets'] = freq_2_itemsets_df['itemsets'].apply(
    lambda x: set([i for i in x]))
freq_2_itemsets_df.head()
```

Out[60]:

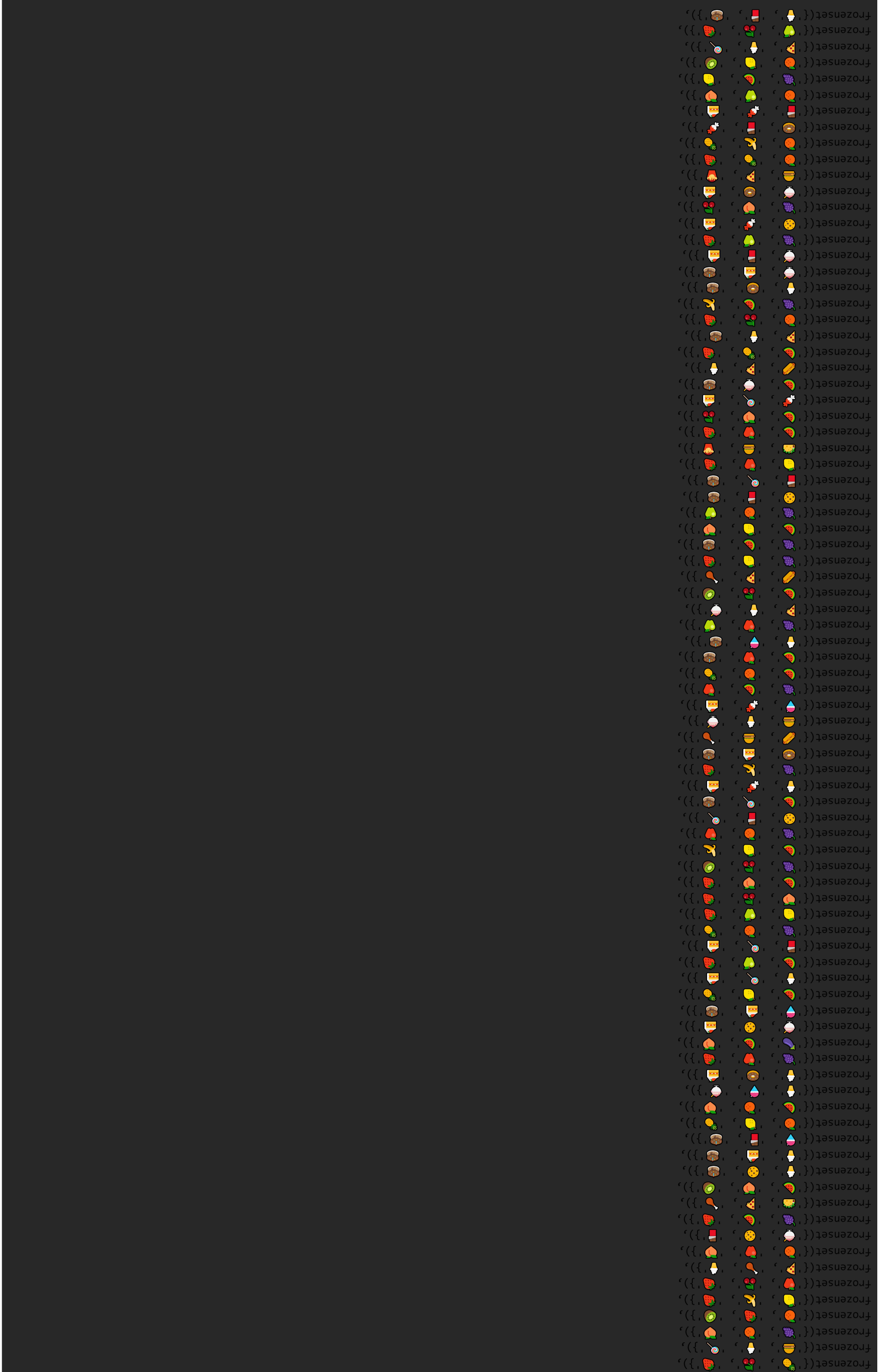
	itemsets
0	{🍔, 🥖}
1	{🍕, 🥖}
2	{🌮, 🍔}
3	{🍕, 🌮}
4	{🍑, 🍆}

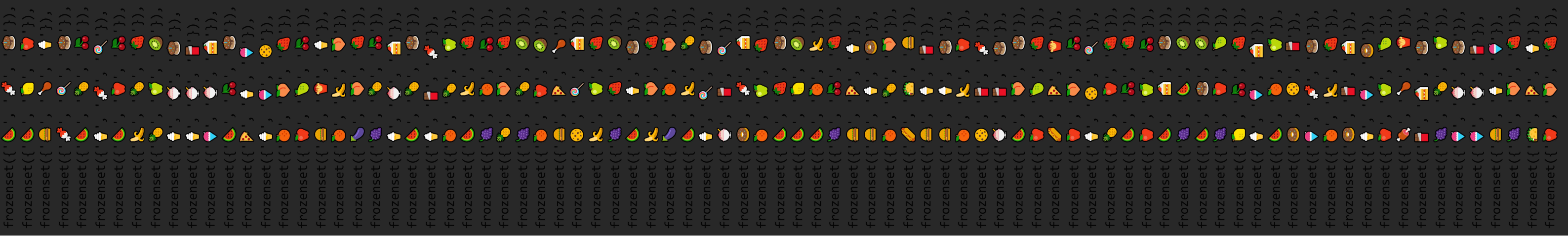
In [61]:
```
# candidate_3_itemsets
# comes from union of frequent 2-itemsets, doing self joins on 2-frequent-itemsets
# frequent 2-itemsets can generate 3--itemsets, which are just candidates but not must frequent

def generate_candidate_3_itemsets(freq_2_itemsets):

    candi_3_itemsets = set()

    for x1 in freq_2_itemsets:
        for x2 in freq_2_itemsets:
            new_set = x1.union(x2)
            if len(new_set) == 3:
                candi_3_itemsets.add(frozenset(new_set))

    return list(candi_3_itemsets)
```

In [62]:
```
candidate_3_itemsets = generate_candidate_3_itemsets(
    freq_2_itemsets_df['itemsets'])
candidate_3_itemsets
```

Out[62]:
```
[frozenset({'🍕', '🍦', '🍰'}),
 frozenset({'🍧', '🥤', '🍰'}),
 frozenset({'🍆', '🍑', '🍒'}),
 frozenset({'🍇', '🍍', '🍓'}),
 frozenset({'🍕', '🍦', '🍪'}),
 frozenset({'🍊', '🍋', '🍏'}),
 frozenset({'🍊', '🍑', '🍒'}),
 frozenset({'🍊', '🍑', '🥝'}),
 frozenset({'🍔', '🍖', '🍗'}),
 frozenset({'🍦', '🍩', '🍪'}),
 frozenset({'🍡', '🍰', '🍩'}),
 frozenset({'🍉', '🍍', '🍑'}),
```


frozenset({
})),





```
frozenset({'🍩', '🍭', '🍰'}),
frozenset({'🍉', '🍍', '🍎'}),
frozenset({'🥖', '🍔', '🍕'}),
frozenset({'🍇', '🍉', '🍊'}),
frozenset({'🍌', '🍒', '🍓'}),
frozenset({'🌮', '🍔', '🍕'}),
frozenset({'🍇', '🍊', '🍋'}),
frozenset({'🍦', '🍭', '🍰'}),
frozenset({'🍨', '🍭', '🍩'}),
frozenset({'🍔', '🍦', '🍰'}),
frozenset({'🍊', '🍋', '🍎'}),
frozenset({'🍦', '🍪', '🍰'}),
frozenset({'🍦', '🍩', '🥤'}),
frozenset({'🍦', '🥤', '🍰'}),
frozenset({'🍉', '🍊', '🍓'}),
frozenset({'🍔', '🍦', '🍪'}),
frozenset({'🥤', '🍬', '🎂'}),
frozenset({'🍉', '🍊', '🍋'}),
frozenset({'🌮', '🍔', '🍗'})]
```

In [63]:

```
print('The number of the candidate 3-itemsets generated:',
      len(candidate_3_itemsets))
```

```
The number of the candidate 3-itemsets generated: 286
```

b) [10 points] Note that this pruning procedure won't give us the smallest set of candidates. Therefore, we can further prune the candidate itemsets by requiring all size-2 subsets of each candidate 3-itemset to be frequent.

Think about the example you've seen in the lecture. Suppose {🍺, 🕯}, {🍺, 🍋}, {🕯, 🍭}, {🕯, 🍋} are all the frequent 2-itemsets. In the lecture, we said {🍺, 🕯, 🍋}, {🍺, 🕯, 🍭}, {🕯, 🍭, 🍋} are candidate 3-itemsets because each 3-itemset is a superset of at least one frequent 2-itemset. However, a larger itemset can never be frequent as long as one of its subset is not frequent. In this case, {🍺, 🕯, 🍭} can never been frequent because {🍺, 🍭} is not frequent. Neither can {🕯, 🍭, 🍋}, as the subset {🍭, 🍋} is not frequent. Ideally, we should be able to exclude the two candidate 3-itemsets {🍺, 🕯, 🍭} and {🕯, 🍭, 🍋} even without scanning the database for counting.

For this exercise, please prune your generated candidate 3-itemsets by requiring all their subsets to be frequent. You will receive full points for this part if the pruning is done correctly.

In [64]:

```
def subsets(org_set):

    sets = set()

    for i in org_set:
        for j in org_set:
            this_set = (i, j)
            if len(frozenset(this_set)) == 2:
                sets.add(frozenset(this_set))
    return sets
```

In [65]:

```
all_freq_2_itemsets = freq_2_itemsets_df['itemsets'].to_list()
pruned_3_itemsets = set()

for i in candidate_3_itemsets:
    if all(element in all_freq_2_itemsets for element in subsets(i)):
        pruned_3_itemsets.add(frozenset(i))
```

In [66]:

```
print("length of the pruned candidate 3-itemsets:", len(pruned_3_itemsets))
pruned_3_itemsets
```

```
length of the pruned candidate 3-itemsets: 88
```

Out[66]:

```
{frozenset({'🍇', '🍍', '🍓'}),
 frozenset({'🎂', '🍰', '🍩'}),
 frozenset({'🍦', '🍩', '🍪'}),
 frozenset({'🥤', '🎂', '🍭'}),
 frozenset({'🍇', '🍊', '🍓'}),
 frozenset({'🍭', '🍰', '🍩'}),
 frozenset({'🍉', '🍋', '🍓'}),
 frozenset({'🍇', '🍉', '🍒'}),
 frozenset({'🍪', '🥤', '🍰'}),
 frozenset({'🍊', '🍋', '🍓'}),
 frozenset({'🍊', '🍎', '🍏'}),
 frozenset({'🍇', '🍉', '🍍'}),
 frozenset({'🍉', '🍊', '🍎'}),
 frozenset({'🍊', '🍍', '🥝'}),
 frozenset({'🍔', '🍗', '🍟'}),
 frozenset({'🍍', '🍏', '🍓'}),
 frozenset({'🍎', '🍏', '🍓'}),
 frozenset({'🍦', '🥤', '🍭'}),
```

2.1.2 Database Scan

a) [10 points] With the candidate itemsets ready, the final step in one iteration of the Apriori algorithm is to scan the database (in our case, you can scan the `emoji_matrix` created above in 1.1c) and count the occurrence of each candidate itemset, divide it by the total number of records to derive the support, and output candidate itemsets whose support meets a chosen threshold.

To do so, please construct a new function, `calculate_frequent_itemsets`, where the input (`candidate_itemsets`) is a list of the candidate 3-itemsets from the previous section. Your function should return a complete list of frequent 3-itemsets with a minimal support of `min_support` (i.e., an argument passed to the function). The returned list should not contain duplicated itemsets. The order of the list does not matter.

In [67]:

```
# list_of_pruned_3_itemsets
```

```
list_of_pruned_3_itemsets = list(pruned_3_itemsets)
# list_of_pruned_3_itemsets
```

In [68]:

```
def calculate_frequent_itemsets(candidate_itemsets, min_support):

    # prepare the blank-support df for 3_itemsets we have
    candidate_df = pd.DataFrame(
        {'itemset': candidate_itemsets, 'support': 0.0})

    for i, row in emoji_item_matrix_df.iterrows():  # loop each tweet
        for j, candidate in candidate_df.iterrows():  # for each tweet, loop each candidate
            x, y, z = candidate['itemset']
            # if each emoji in this candidate set, are also in this tweet (occurence>0)
            # if row[x]~=0, and row[y]~=0, and...
            if row[x] and row[y] and row[z]:
                # the frequncy count for this 3-itemset can +=1
                candidate_df['support'][j] += 1

    # calculate the support value for all candidates
    candidate_df['support'] = candidate_df['support'] / \
        len(emoji_item_matrix_df)

    # list of frequent 3-itemsets with a minimal support
    candidates_with_min_sup = candidate_df[candidate_df['support']
                                           >= min_support]["itemset"].tolist()
    # here, cause our input itemset is set, so the returned list should not contain duplicated itemsets

    return candidates_with_min_sup


calculate_frequent_itemsets(list_of_pruned_3_itemsets, 0.007)
```

```
C:\Users\Jiaqi\AppData\Local\Temp/ipykernel_88232/1387887279.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  candidate_df['support'][j] += 1
```

Out[68]:

```
[frozenset({'🥃', '🐓', '🍭'}),
 frozenset({'🍔', '🍕', '🍟'}),
 frozenset({'🍇', '🍉', '🍊'})]
```

So, we can see the same/ similar results with using the Apriori Algorithm, but here the instructor team eliminated the drink-related emojis of our input dataset, so here do not contain the drink-related results.

In [ ]:

```

```

Part 3: Evaluating Frequent Itemsets

Even though you may have found all the frequent itemsets, not all of them are “interesting”.

People have developed various measurements of the interestingness of patterns. Most of them split the itemset into an antecedent item(set) and a consequent item(set), and then measure the correlation between the antecedent and the consequent. Let's try some of such measurements implemented by the `mlxtend.frequent_patterns.association_rules` API. For more information about the API, visit the documentation at http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/.

a) [10 points] First, apply the apriori function to 'emoji_matrix' created in section 1.1c with a min_support of 0.005 and use_colnames = True. Then, apply the association_rules function to the result with metric = "lift" and min_threshold = 3 (meaning to only return values where lift is equal to or greater than 3).

b) [10 points] Next, we ask that you implement another interestingness measurement, the (full) mutual information. The measurement is defined as such:

$$I(X;Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(X = x, Y = y) \log_2 \frac{P(X = x, Y = y)}{P(X = x)P(Y = y)}.$$

Note that the logarithm requires that the joint probability $P(X = x, Y = y) > 0$, which does not hold for some (x, y). However, since we know that when $P(X = x, Y = y) = 0$, it would not contribute to the sum, you may assume $P(X = x, Y = y) \log_2 \frac{P(X=x,Y=y)}{P(X=x)P(P=y)} = 0$ in that case.

x, y are possible values of X and Y; in the case of appearance or absence of an item, 1 or 0. Therefore, we need to consider all possible combinations of x and y, that is, $(X = 1, Y = 1)$, $(X = 1, Y = 0)$, $(X = 0, Y = 1)$, $(X = 0, Y = 0)$.

Please construct a function, 'mi', that uses the three support values ((1) antecedent support, (2) consequent support and (3) support) to compute the mutual information. All three parameters are in [0, 1], and you can assume the validity of the input. Use 2 as the log base.

c) [5 points] Then, use this function to add the mutual information value to each row of the DataFrame above (i.e., the results of applying association_rules).

a)

In [69]:
```
emoji_item_matrix_df
```

Out[69]:

											...										
0	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9953	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9954	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9955	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9956	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9957	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

9958 rows × 105 columns

In [70]:
```
# get frequent itemsets
frequent_itemsets = apriori(
    emoji_item_matrix_df, min_support=0.005, use_colnames=True)
# get life>=3 frequent_itemsets's association_rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=3)
rules
```

```
C:\ProgramData\Anaconda3\lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
```

Out[70]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	()	()	0.058747	0.031332	0.008435	0.143590	4.582906	0.006595	1.131080
1	()	()	0.031332	0.058747	0.008435	0.269231	4.582906	0.006595	1.288031
2	()	()	0.138783	0.019381	0.011348	0.081766	4.218764	0.008658	1.067939
3	()	()	0.019381	0.138783	0.011348	0.585492	4.218764	0.008658	2.077686
4	()	()	0.044286	0.019381	0.005021	0.113379	5.849870	0.004163	1.106017
...	...	...	...	...	...	...	...	...	...
511	(,)	(,)	0.012352	0.014260	0.005122	0.414634	29.076950	0.004945	1.683973
512	()	(, ,)	0.052018	0.006527	0.005122	0.098456	15.083398	0.004782	1.101967
513	()	(, ,)	0.181663	0.007632	0.005122	0.028192	3.693943	0.003735	1.021157
514	()	(, ,)	0.061759	0.006929	0.005122	0.082927	11.967904	0.004694	1.082870
515	()	(, ,)	0.079634	0.007532	0.005122	0.064313	8.539016	0.004522	1.060684

516 rows × 9 columns

b)

Support [P(x,y)]: probability that a transaction contains both X and Y

we know:

P(X=1) = antecedent_support [P(X=1) means P(antecedent appears in the set)]
P(Y=1) = consequent_support
P (X=1, Y=1) = support

we can write as:

P(x=1, y=1) = support

P(x=1, y=0) = P(x=1) - P(x=1, y=1) = antecedent_support - support

P(x=0, y=1) = P(y=1) - P(x=1, y=1) = consequent_support - support

P(x=0, y=0) = 1 - rest of these three = 1-(support + antecedent_support - support + consequent_support - support) = 1 - antecedent_support + support - consequent_support

also,

P(x=0) = 1 - P(x=1) = 1-antecedent_support
P(y=0) = 1 - P(y=1) = 1-consequent_support

In [71]:

```
# another interestingness measurement
# define function to compute the (full) mutual information

# input are three support values in [0, 1]
def mi(antecedent_support, consequent_support, support):

    # get values
    P_x1 = antecedent_support
    P_y1 = consequent_support
    P_x1_y1 = support
    P_x1_y0 = antecedent_support - support
    P_x0_y1 = consequent_support - support
    P_x0_y0 = 1 - antecedent_support + support - consequent_support
    P_x0 = 1-antecedent_support
    P_y0 = 1-consequent_support

    # calculate mi
    mutual_info = P_x1_y0*np.log2(P_x1_y0/(P_x1*P_y0)) + P_x1_y1*np.log2(P_x1_y1/(
        P_x1*P_y1)) + P_x0_y1*np.log2(P_x0_y1/(P_x0*P_y1)) + P_x0_y0*np.log2(P_x0_y0/(P_x0*P_y0))

    return mutual_info
```

In [72]:

```
# test with example in class


mi(0.6, 0.75, 0.4)
```

Out[72]:

```
0.04287484674660057
```

c)

In [73]:

```
rules
```

Out[73]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(🍲)	(🍩)	0.058747	0.031332	0.008435	0.143590	4.582906	0.006595	1.131080
1	(🍩)	(🍲)	0.031332	0.058747	0.008435	0.269231	4.582906	0.006595	1.288031
2	(🍔)	(🥖)	0.138783	0.019381	0.011348	0.081766	4.218764	0.008658	1.067939
3	(🥖)	(🍔)	0.019381	0.138783	0.011348	0.585492	4.218764	0.008658	2.077686
4	(🍕)	(🥖)	0.044286	0.019381	0.005021	0.113379	5.849870	0.004163	1.106017
...	...	...	...	...	...	...	...	...	...
511	(🍹, 🍺)	(🍸, 🍻)	0.012352	0.014260	0.005122	0.414634	29.076950	0.004945	1.683973
512	(🍸)	(🍺, 🍹, 🍻)	0.052018	0.006527	0.005122	0.098456	15.083398	0.004782	1.101967
513	(🍻)	(🍸, 🍹, 🍺)	0.181663	0.007632	0.005122	0.028192	3.693943	0.003735	1.021157
514	(🍹)	(🍺, 🍸, 🍻)	0.061759	0.006929	0.005122	0.082927	11.967904	0.004694	1.082870

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
515	(🍺)	(🍸, 🍹, 🍤)	0.079634	0.007532	0.005122	0.064313	8.539016	0.004522	1.060684

516 rows × 9 columns

In [74]:
```
rules['mutual_information'] = rules.apply(lambda rule: mi(
    rule['antecedent support'], rule['consequent support'], rule['support']), axis=1)

rules
```

Out[74]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	mutual_information
0	(☕)	(🍩)	0.058747	0.031332	0.008435	0.143590	4.582906	0.006595	1.131080	0.010774
1	(🍩)	(☕)	0.031332	0.058747	0.008435	0.269231	4.582906	0.006595	1.288031	0.010774
2	(🍔)	(🥖)	0.138783	0.019381	0.011348	0.081766	4.218764	0.008658	1.067939	0.015561
3	(🥖)	(🍔)	0.019381	0.138783	0.011348	0.585492	4.218764	0.008658	2.077686	0.015561
4	(🍕)	(🥖)	0.044286	0.019381	0.005021	0.113379	5.849870	0.004163	1.106017	0.007833
...	...	...	...	...	...	...	...	...	...	...
511	(🍹, 🍤)	(🍸, 🍤)	0.012352	0.014260	0.005122	0.414634	29.076950	0.004945	1.683973	0.020914
512	(🍸)	(🍺, 🍹, 🍤)	0.052018	0.006527	0.005122	0.098456	15.083398	0.004782	1.101967	0.017391
513	(🍤)	(🍸, 🍹, 🍺)	0.181663	0.007632	0.005122	0.028192	3.693943	0.003735	1.021157	0.006422
514	(🍹)	(🍺, 🍸, 🍤)	0.061759	0.006929	0.005122	0.082927	11.967904	0.004694	1.082870	0.015286
515	(🍺)	(🍸, 🍹, 🍤)	0.079634	0.007532	0.005122	0.064313	8.539016	0.004522	1.060684	0.012379

516 rows × 10 columns

In [ ]:
```

```

Part 4: Itemset Similarity

Recall that pattern and similarity are two basic outputs of data mining. So far, we have been playing with patterns - frequent itemsets and association rules can all be seen as "patterns". In the last part, let's work on itemset similarities.

4.1 Jaccard Similarity

Jaccard similarity is a simple but powerful measurement of itemset similarity, defined as follows:

$$\text{Jaccard\_similarity(A, B)} = \frac{|A \cap B|}{|A \cup B|}$$

a) [5 points] Complete a function, 'jaccard_similarity', to calculate the Jaccard similarity between two sets. You may assume that at least one of the sets is not empty.

b) [5 points] With this Jaccard similarity function, please calculate the Jaccard similarity between any given Tweet with all other Tweets and find the Tweets that are most similar (i.e., have the highest jaccard similarity values) in terms of the set of food/drink emojis used. How would you interpret the results?

a)

In [75]:
```
# assume that at least one of the sets is not empty

def jaccard_similarity(set_1, set_2):

    # intersection and union
    intersection = len(set_1.intersection(set_2))
    union = len(set_1.union(set_2))

    # calculate similarity
    similarity = intersection/union

    return similarity
```

b)

In [76]:
```
# only need to pick one Tweet and calculate the Jaccard similarity of that Tweet with all other Tweets
```

```
# in terms of the set of food/drink emojis used
tweets_simi = tweets.copy()
tweets_simi
```

Out[76]:

	tweet_content	food_drink_emoji
0	RT @CalorieFixess: 🍗🌯🍔🍒 400 Calories https://t.co/9OaPOWUSht	[🌯, 🍒, 🍔, 🍗]
1	RT @1_F_I_R_S_T: _ 🍋[1]⟩ Grow your account fast! [2]⟩ 🍍Retweet Now!!! 🍉 🍇[3]⟩ Follow all Retweets [4]⟩ 🍊Follow back everybody 🍓[5]⟩ Follow me & @1f_sts🍇 [6]...	[🍇, 🍋, 🍉, 🍊, 🍍, 🍓]
2	RT @LegendDeols: 👉👉👉G€T Ready to dance💃🕺💃🕺💃💃 With #LittleLittlepag 🥃🍸🍸🍸🥃 with Da$HING DEOLS 🌼🌼🌼🌼🌼. Song 🎧🎧🎧 out today at 🔟 am .@ypdphirse...	[🍸, 🥃]
3	@britch_x Hubby's friend bought us Wendy's-cheeseburger (no onions), fries and a Coke. 🍔🍟	[🍔, 🍟]
4	RT @DAILYPUPPIES: Workout partner ☕💪😍 https://t.co/3p0VZs6RKp	[☕, 💪]
...	...	...
9953	#1stTest Brilliant Tension #England need 2wks to Win While #India Need just 42runs To Win this fantastic game of Cricket 🏏😁👏👏🤜🥃	[🤜, 🥃]
9954	RT @Thabang92416252: "@PinexAndApplex: 🍍🍏 @EmteeSA ft @Nasty_CSA - Winning🔥 https://t.co/cNDLstzJ7e"	[🍍, 🍏]
9955	@nuttysteph84 Steph!! 😭😭 🥓☕🍳💁❤️x	[☕, 🍳, 🥓]
9956	I'm heading to a wedding. Weekend session afoot! 🎉🤜🍷🍹🤜 🏠	[🍷, 🍹, 🤜]
9957	Bubbles time 🍾🥂😍 https://t.co/7UTbBRIVeb	[🍾, 🥂]

9958 rows × 2 columns

In [77]:

```
# create the similarity column
tweets_simi["jaccard_similarity_with_1st_tweet"] = 0.0
tweets_simi
```

Out[77]:

	tweet_content	food_drink_emoji	jaccard_similarity_with_1st_tweet
0	RT @CalorieFixess: 🍗🌯🍔🍒 400 Calories https://t.co/9OaPOWUSht	[🌯, 🍒, 🍔, 🍗]	0.0
1	RT @1_F_I_R_S_T: _ 🍋[1]⟩ Grow your account fast! [2]⟩ 🍍Retweet Now!!! 🍉 🍇[3]⟩ Follow all Retweets [4]⟩ 🍊Follow back everybody 🍓[5]⟩ Follow me & @1f_sts🍇 [6]...	[🍇, 🍋, 🍉, 🍊, 🍍, 🍓]	0.0
2	RT @LegendDeols: 👉👉👉G€T Ready to dance💃🕺💃🕺💃💃 With #LittleLittlepag 🥃🍸🍸🍸🥃 with Da$HING DEOLS 🌼🌼🌼🌼🌼. Song 🎧🎧🎧 out today at 🔟 am .@ypdphirse...	[🍸, 🥃]	0.0
3	@britch_x Hubby's friend bought us Wendy's-cheeseburger (no onions), fries and a Coke. 🍔🍟	[🍔, 🍟]	0.0
4	RT @DAILYPUPPIES: Workout partner ☕💪😍 https://t.co/3p0VZs6RKp	[☕, 💪]	0.0
...	...	...	...
9953	#1stTest Brilliant Tension #England need 2wks to Win While #India Need just 42runs To Win this fantastic game of Cricket 🏏😁👏👏🤜🥃	[🤜, 🥃]	0.0
9954	RT @Thabang92416252: "@PinexAndApplex: 🍍🍏 @EmteeSA ft @Nasty_CSA - Winning🔥 https://t.co/cNDLstzJ7e"	[🍍, 🍏]	0.0
9955	@nuttysteph84 Steph!! 😭😭 🥓☕🍳💁❤️x	[☕, 🍳, 🥓]	0.0
9956	I'm heading to a wedding. Weekend session afoot! 🎉🤜🍷🍹🤜 🏠	[🍷, 🍹, 🤜]	0.0
9957	Bubbles time 🍾🥂😍 https://t.co/7UTbBRIVeb	[🍾, 🥂]	0.0

9958 rows × 3 columns

In [78]:

```
pick_set = set(tweets_simi['food_drink_emoji'][0])
pick_set
```

Out[78]:

```
{'🌯', '🍒', '🍔', '🍗'}
```

In [79]:

```
# get jaccard_similarity

tweets_simi["jaccard_similarity_with_1st_tweet"] = tweets_simi["food_drink_emoji"].apply(
    lambda x: jaccard_similarity(pick_set, x))

tweets_simi
```

Out[79]:

	tweet_content	food_drink_emoji	jaccard_similarity_with_1st_tweet
0	RT @CalorieFixess: 🍗🌯🍔🍒 400 Calories https://t.co/9OaPOWUSht	[🌯, 🍒, 🍔, 🍗]	1.0
1	RT @1_F_I_R_S_T: _ 🍋[1]⟩ Grow your account fast! [2]⟩ 🍍Retweet Now!!! 🍉 🍇[3]⟩ Follow all Retweets [4]⟩ 🍊Follow back everybody 🍓[5]⟩ Follow me & @1f_sts🍇 [6]...	[🍇, 🍋, 🍉, 🍊, 🍍, 🍓]	0.0
2	RT @LegendDeols: 👉👉👉G€T Ready to dance💃🕺💃🕺💃💃 With #LittleLittlepag 🥃🍸🍸🍸🥃 with Da$HING DEOLS 🌼🌼🌼🌼🌼. Song 🎧🎧🎧 out today at 🔟 am .@ypdphirse...	[🍸, 🥃]	0.0
3	@britch_x Hubby's friend bought us Wendy's-cheeseburger (no onions), fries and a Coke. 🍔🍟	[🍔, 🍟]	0.2
4	RT @DAILYPUPPIES: Workout partner ☕💪😍 https://t.co/3p0VZs6RKp	[☕, 💪]	0.0

	tweet_content	food_drink_emoji	jaccard_similarity_with_1st_tweet
...	...	...	...
9953	#1stTest Brilliant Tension #England need 2wks to Win While #India Need just 42runs To Win this fantastic game of Cricket 🏏😂👏👏🍻🥃	[🍻, 🥃]	0.0
9954	RT @Thabang92416252: "@PinexAndApplex: 🍍🍏 @EmteeSA ft @Nasty_CSA - Winning🔥 https://t.co/cNDLstzJ7e"	[🍍, 🍏]	0.0
9955	@nuttysteph84 Steph!! 😭😭 🍗☕🍳🤦‍♀️❤️x	[☕, 🍳, 🍗]	0.0
9956	I'm heading to a wedding. Weekend session afoot! 🎉🍻🍷🍹🍻 🏠	[🍷, 🍹, 🍻]	0.0
9957	Bubbles time 🍾🥂🤩 https://t.co/7UTbBRlVeb	[🍾, 🥂]	0.0

9958 rows × 3 columns

In [80]:
```
# find the Tweets that are most similar (have the highest jaccard similarity values) with picked set

ordered_tweets = tweets_simi.sort_values(by="jaccard_similarity_with_1st_tweet", ascending=False)
ordered_tweets.head(10)
```

Out[80]:

	tweet_content	food_drink_emoji	jaccard_similarity_with_1st_tweet
0	RT @CalorieFixess: 🍗🌯🍔🍒 400 Calories https://t.co/9OaPOWUSht	[🌯, 🍒, 🍔, 🍗]	1.00
6800	RT @levelscafeabuja: Chow! 😍🫧🍗🌯🍔 #LevelsCafeAbuja https://t.co/r9YvzPNFpi	[🌯, 🍔, 🍗]	0.75
9158	RT @AStateRedWolves: ✅ Countertops: Installed (Not pictured: The awesome wings, burgers, brisket and brats you'll be eating off of it) 🍗🍔...	[🍔, 🍗]	0.50
777	@SunnyAnderson @rosannascotto I don't think KFC, McD's count as "grocery" shopping 🍗🍔 Thanks for the laugh Sunny. Have a great day!	[🍔, 🍗]	0.50
6226	RT @yooojax: 3. Free Food 🍗🍔 will ready by 2PM #DayRockPt2 (Caribbean Food Trays Available)JMGYBBAG https://t.co/EyuGEYaqZ8	[🍔, 🍗]	0.50
7428	Kicking off the weekend with a cheeky BBQ? Here's our recipe for a safe and stress-free meal! 🚒🔥 =😭 🍔🍗=😎... https://t.co/DtQtpvTQoB	[🍔, 🍗]	0.50
7877	@tafarireid07 Did you say bbq? 🔥🍔🍗🚙	[🍔, 🍗]	0.50
5328	RT @MAPSTTU: @EtaUpAlphas is starting the semester right with #B2SBBQ2K18 🍔🍗 Oh and don't forget, we will be in attendance!! Can't wait...	[🍔, 🍗]	0.50
5334	RT @thatssochioma: You don't want to miss this 😎 come and chop 🍗 🍔 and come by our table! Win-win 😉 #MAPSTTU #TTU22 #EtaUpAlphas #B2SBBQ2k1...	[🍔, 🍗]	0.50
7788	I'm hungry for chicken 🍗 wings or burrito 🌯	[🌯, 🍗]	0.50

The Tweets that are most similar (have the highest jaccard similarity values) with our picked tweet, are listed above. The picked tweet has the highest similarity with itself = 1, then have similarity of 0.75 with "tweet 6800", which contains 3 common emojis [🌯, 🍔, 🍗].

> the most similar tweets are also food related emojis, even some similar kind of food, e.g. Hamburgers and chicken thighs, the more emojis they have in common, the more similar they are.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: