Fall 2022 MAT265 Assignment 04: Frequency Mining with Singer's Discography

- Shaokang Li

Introduction:

Frequency-pattern related algorithm is used as an analytical process that finds frequent patterns or associations from data sets. For example, grocery store transaction data might have a frequent pattern that people usually buy chips and beer together. With this tool, I would like to check if there's any pattern in Album checkouts, specifically, what albums would people like to checkout together? To take a deeper dive in the CD checkout data, questions below are also considered:

- What does people like to checkout together with a specific singer (in this case, Adele's / Taylor Swift's) CD?
  - FP-Grow (Problem)
  - Generally, what do people like to checkout together?
- How long did each CD last in Library.
- New purchase by year together compared with its popularity

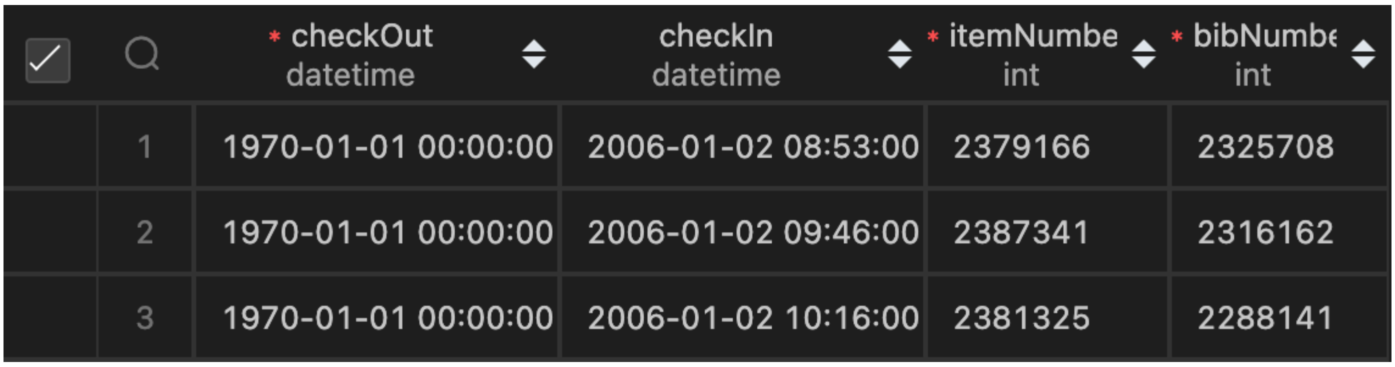
These results may give us finding about CD itself and information about the singer's career status.

Tryout FP-Grow Algorithm

We need to have transaction data like this to utilize the algorithm: `[[book1,book2...] [book3,book2...]...]`, which is a `vector<vector<Book>>` in C++ or `list(list(Book))` in Python. Each element in the list is a transaction history about:

At a certain timestamp, one person checkout some items together.

However, such data isn't directly available in the database. What we have for a single transaction is checkout timestamp and a single itemNumber. We may not know what people checkout together at a certain timestamp if we don't make any assumption.



	* checkOut datetime	checkIn datetime	* itemNumbe int	* bibNumbe int
1	1970-01-01 00:00:00	2006-01-02 08:53:00	2379166	2325708
2	1970-01-01 00:00:00	2006-01-02 09:46:00	2387341	2316162
3	1970-01-01 00:00:00	2006-01-02 10:16:00	2381325	2288141

Data in Transaction table

Assumption: items checkout at the same time are by the same person

With assumption above, I firstly look into all transaction in one day, find out if there's any pattern.

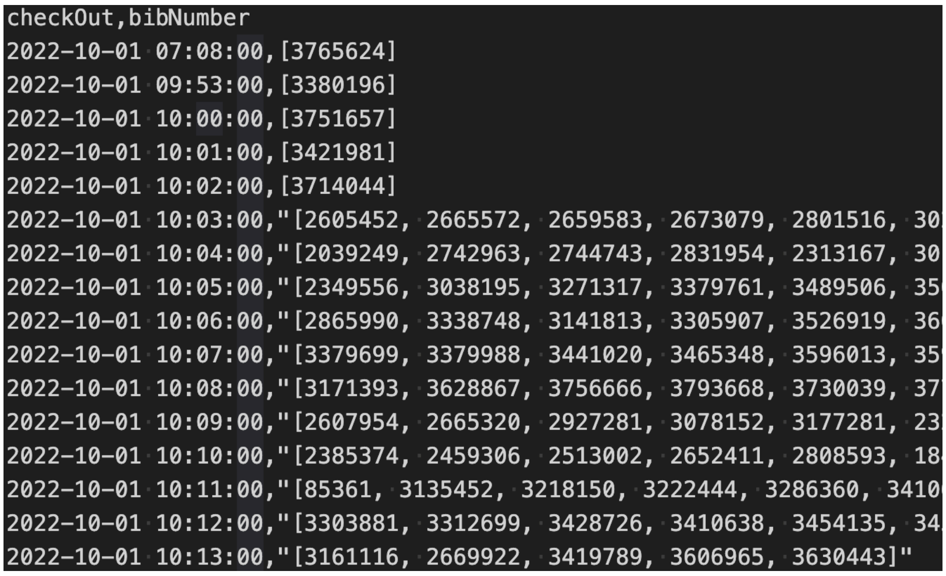
- Step 01: Get raw transaction data

```sql
select checkOut, bibNumber
from transactions
where checkOut > '2022-10-01';
```

- Step 02: Convert data format

Since MySQL doesn't support the type of list, I use Python (with Pandas library) to handle further operations.

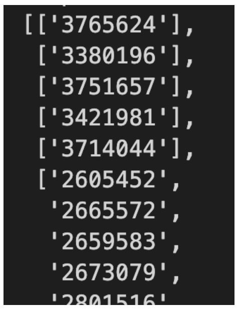
```python
import pandas as pd
import numpy as np

# Read in the data
df = pd.read_csv('transactions.csv')
df.head()
# Group data by checkout time, return with a list of list
df_grouped = df.groupby('checkOut')['bibNumber'].apply(list)
df_grouped.head()
df_grouped.to_csv('grouped.csv')
```



```
checkOut,bibNumber
2022-10-01 07:08:00,[3765624]
2022-10-01 09:53:00,[3380196]
2022-10-01 10:00:00,[3751657]
2022-10-01 10:01:00,[3421981]
2022-10-01 10:02:00,[3714044]
2022-10-01 10:03:00,"[2605452, 2665572, 2659583, 2673079, 2801516, 30
2022-10-01 10:04:00,"[2039249, 2742963, 2744743, 2831954, 2313167, 30
2022-10-01 10:05:00,"[2349556, 3038195, 3271317, 3379761, 3489506, 35
2022-10-01 10:06:00,"[2865990, 3338748, 3141813, 3305907, 3526919, 36
2022-10-01 10:07:00,"[3379699, 3379988, 3441020, 3465348, 3596013, 35
2022-10-01 10:08:00,"[3171393, 3628867, 3756666, 3793668, 3730039, 37
2022-10-01 10:09:00,"[2607954, 2665320, 2927281, 3078152, 3177281, 23
2022-10-01 10:10:00,"[2385374, 2459306, 2513002, 2652411, 2808593, 18
2022-10-01 10:11:00,"[85361, 3135452, 3218150, 3222444, 3286360, 3410
2022-10-01 10:12:00,"[3303881, 3312699, 3428726, 3410638, 3454135, 34
2022-10-01 10:13:00,"[3161116, 2669922, 3419789, 3606965, 3630443]"
```

Now that the data has been converted to the seemingly desired format. Note that there are double quotes in some rows while others doesn't have. We have to perform an extra step to make sure they are in the same Python format.

```
#Format handling
fp = df_grouped.to_dict()
transactions = list(fp.values())
res = [[str(e) for e in ele] for ele in transactions]
```



```
[['3765624'],
 ['3380196'],
 ['3751657'],
 ['3421981'],
 ['3714044'],
 ['2605452',
  '2665572',
  '2659583',
  '2673079',
```

Also, the timestamp info has been discarded since it's not necessary in FP-Grow Algorithm. Each element in the list is a list of strings. (Rather than list of integer).

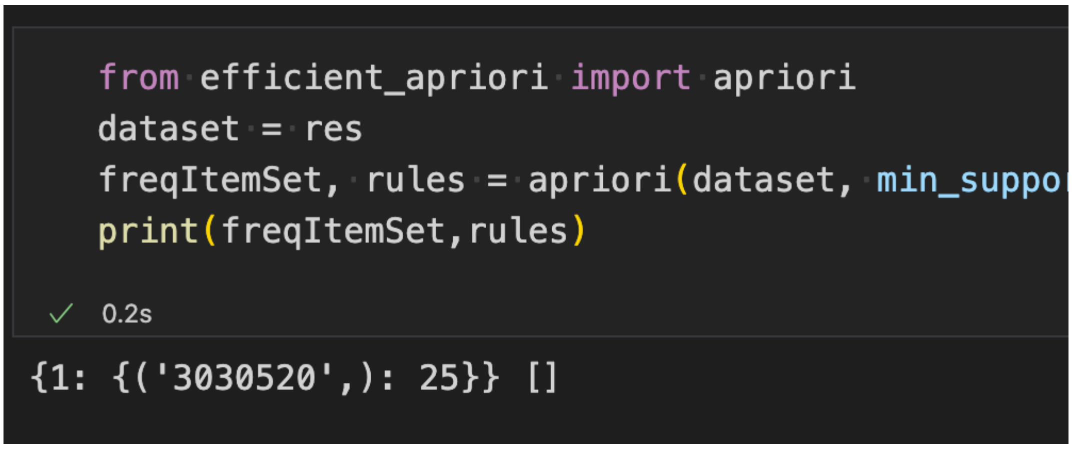
- Step 03: Feed data into algorithm

I performed the algorithm with 2 different libraries: `efficient_apriori` and `FP-Growth`. The implementation of the latter one is somewhat buggy so I choose the first one instead. These two algorithms gives the same result. The difference is their time complexity, in which FP-Growth is `O(n^2)` and Apriori is `O(2^n)`. Since the dataset is limited to 1 day, the difference in time complexity won't impose a great influence in algorithm's execution speed.

```
from efficient_apriori import apriori
dataset = res
freqItemSet, rules = apriori(dataset, min_support=0.3, min_confidence=0.6)
print(rules)
```

- Result:

Unfortunately, even with low threshold like 0.2, the algorithm won't return any rules. It can be said that, there's no pattern within one day using approach above.



```
from efficient_apriori import apriori
dataset = res
freqItemSet, rules = apriori(dataset, min_suppo
print(freqItemSet,rules)
```

✓ 0.2s

```
{1: {('3030520',): 25}} []
```

> Can it be a matter of time range? One day seems too short.

- I also use approach above to look any patterns within one month’s transaction history.
    - However, still no pattern found.

> Maybe the pattern can be found within a certain category?

Then, I narrow down the scope to CD. Getting transaction data starts from this September till today and perform above procedure.

```sql
select
    cout,
    bibNumber
from spl_2016.inraw
where
    itemtype in (
        'arcd',
        'nacd',
        'cacdnf',
        'jrcd',
        'accd',
        'cacd',
        'cabocd',
        'cccd',
        'calncd',
        'nabocdc',
        'ncbocd',
        'nybocd',
        'jccd',
        'nccd'
    )
    and cout > '2022-09-01';
```

- No pattern found.

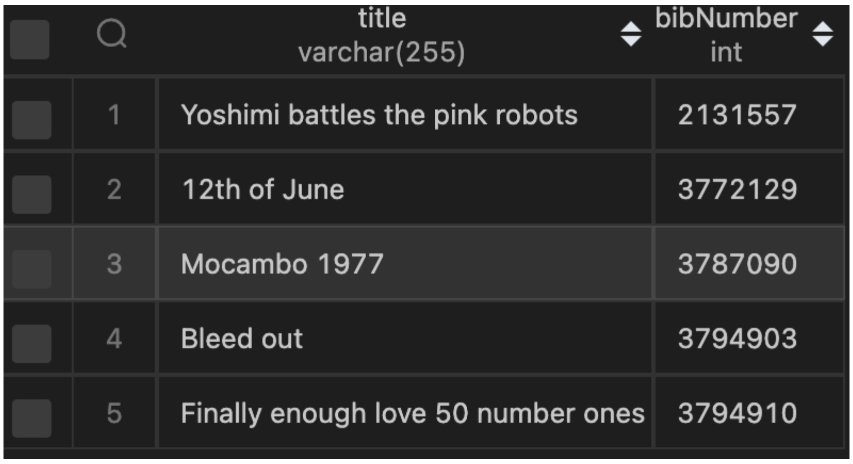
- Finally, I make a loose assumption. Assume checkouts within same hour is the made by the same person.

```sql
select
    date_format(COUT, '%Y-%m-%d %H') as my_date,
    bibNumber
from spl_2016.inraw
where
    itemtype in (
        'arcd',
        'nacd',
        'cacdnf',
        'jrcd',
        'accd',
        'cacd',
        'cabocd',
        'cccd',
        'calncd',
        'nabocdc',
        'ncbocd',
        'nybocd',
        'jccd',
        'nccd'
    )
    and cout > '2022-09-01';
```

There are some results with a loosened assumption. Below are the dependency rules, which means, rule in the form `a->b` people who borrow item `a` is likely to borrow `b` together.

-
  ```
  [{2131557} -> {3772129}, {3787090} -> {3772129}, {3794903} -> {3772129}, {3794910} -> {3772129}]
  ```

  The BibNum to Title correspondence is shown as below



	title varchar(255)	bibNumber int
1	Yoshimi battles the pink robots	2131557
2	12th of June	3772129
3	Mocambo 1977	3787090
4	Bleed out	3794903
5	Finally enough love 50 number ones	3794910

Perform above approach on a certain singer?

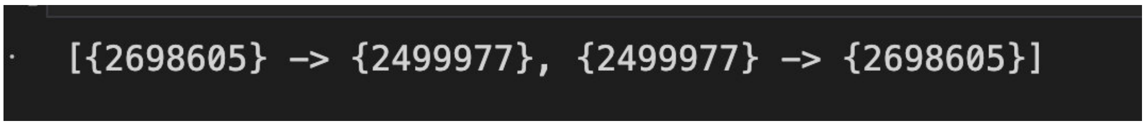
It would be interesting to look at a specific singer's discography, find out if people would like to checkout some of his/her works together.

- Narrow down the range to Adele

```
select
    count(*) as borrow,
    bibNumber,
    title
from spl_2016.inraw
where
    itemtype in (
```

```
            'arcd',
            'nacd',
            'cacdnf',
            'jrcd',
            'accd',
            'cacd',
            'cabocd',
            'cccd',
            'calncd',
            'nabocdc',
            'ncbocd',
            'nybocd',
            'jccd',
            'nccd'
        )
    and cout > '2005-01-01'
    and title in ('19', '21', '25', '30')
group by bibNumber, title
order by borrow desc;
```

Perform algorithms above, we have Album 19 and 21 are 2 albums that people love to checkout together.



```
[{2698605} -> {2499977}, {2499977} -> {2698605}]
```

- Narrow down to Taylor Swift:

```
select cout, itemNumber
from spl_2016.inraw
where
    bibNumber in (
        2545593,
        2678602,
        2851592,
        3049310,
        3297477,
        3463592,
        3595799,
        3629274
    )
where cout<'2018-01-01' and cout>'2010-01-01'
order by itemNumber, cout desc;
```

  - Those who borrow Speak Now is likely to borrow Red
  - People like to borrow 1989 and Red together.
    - It's interesting that people would like to check out consecutively released albums together. (Speak Now 2010, Red 2012, 1989 2014).

Other questions on CDs by Taylor Swift

Except for frequency pattern mining, below questions may also give some insights on singer's career and library's management.

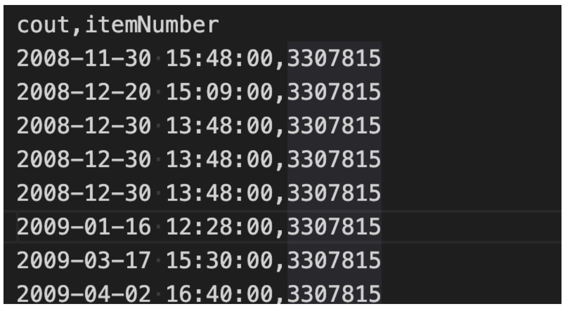
- How long did each CD last in the library?
- Dos a single item change its barcode?
- How about the new purchase each year compared to its checkout times?

Duration:

To calculate the duration of a single item last in the library, we need to know the start date and end date. Since there's no end date provided (No info on whether an item is missing or discarded), I made below criteria for a certain item considered as `missing` or `inactive`. And the calculation is performed on 4 different albums from her.

Criteria: Item that no one borrow for >= 1000 days from today as inactive. Otherwise, it's still in active state and will not be taken into account.

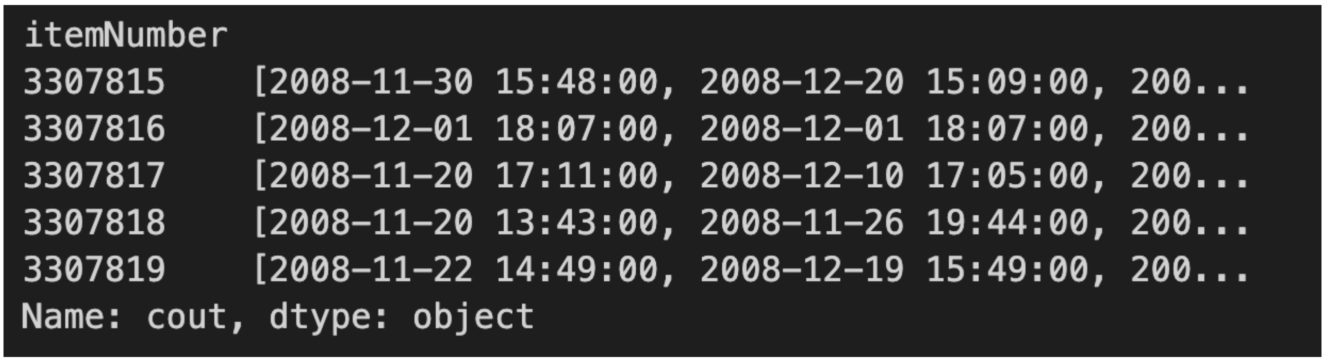
Step 01: Get raw data of the Album (Using BibNumber)



```
cout,itemNumber
2008-11-30 15:48:00,3307815
2008-12-20 15:09:00,3307815
2008-12-30 13:48:00,3307815
2008-12-30 13:48:00,3307815
2008-12-30 13:48:00,3307815
2009-01-16 12:28:00,3307815
2009-03-17 15:30:00,3307815
2009-04-02 16:40:00,3307815
```

Step 02: Convert the data format

To calculate the duration, we need to know the start and end date. A quick solution is to group the data by its itemNumber, and sort by checkout time. The first checkout time is the start date and the last one is the end date (if it's inactive).



```
itemNumber
3307815    [2008-11-30 15:48:00, 2008-12-20 15:09:00, 200...
3307816    [2008-12-01 18:07:00, 2008-12-01 18:07:00, 200...
3307817    [2008-11-20 17:11:00, 2008-12-10 17:05:00, 200...
3307818    [2008-11-20 13:43:00, 2008-11-26 19:44:00, 200...
3307819    [2008-11-22 14:49:00, 2008-12-19 15:49:00, 200...
Name: cout, dtype: object
```

```python
import pandas as pd
import numpy as np

# Read in the data
df = pd.read_csv('fearless.csv')
df.head()

df_grouped = df.groupby('itemNumber')['cout'].apply(list)
df_grouped.head()

```

Step 03: Calculate duration

```python
fp = df_grouped.to_dict()
missing_item_lifespan = []

for kv in fp.items():
    dt = pd.to_datetime(kv[1][0][0:10], format='%Y-%m-%d')
    dt1 = pd.to_datetime(kv[1][-1][0:10], format='%Y-%m-%d')
    today = pd.to_datetime('2022-11-01', format='%Y-%m-%d')
    if ((today-dt1).days>1000):
        missing_item_lifespan.append((dt1-dt).days)

avg = sum(missing_item_lifespan)/len(missing_item_lifespan)
print("The average is ", round(avg,2))
```

Barcode:

Query below is used to check if there's any change in barcode for the same itemNumber

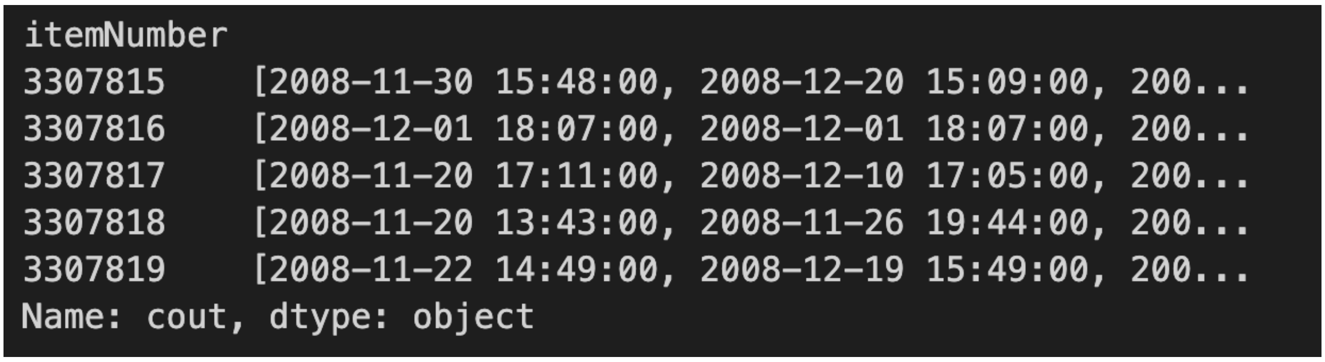
```sql
select *
from spl_2016.inraw t2
    inner join spl_2016.inraw t1 on t1.itemNumber = t2.itemNumber
where
    t1.barcode <> t2.barcode
    and t1.bibNumber = 2545593;
```

New purchase versus Checkouts:

Since there's no raw data on supply chain, I have to make yet another assumption.

Assumption: New purchase is approximately the time of the first checkout.

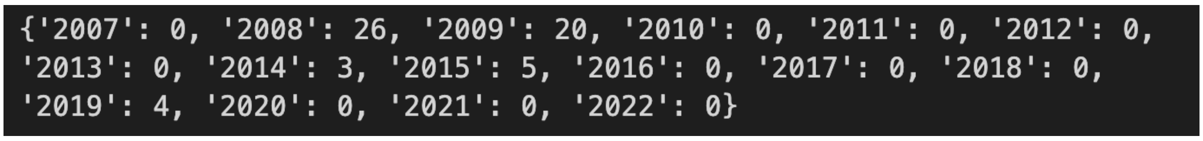
With the converted data, we can easily find out how many new purchases per year.



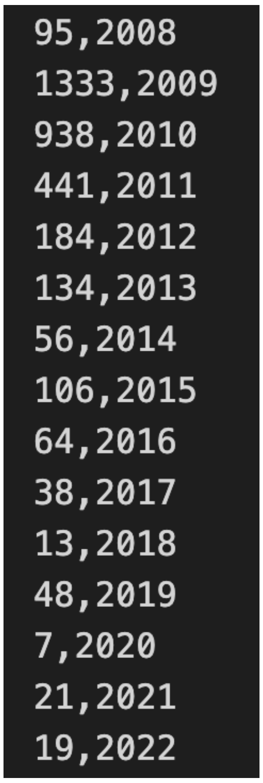
```
itemNumber
3307815    [2008-11-30 15:48:00, 2008-12-20 15:09:00, 200...
3307816    [2008-12-01 18:07:00, 2008-12-01 18:07:00, 200...
3307817    [2008-11-20 17:11:00, 2008-12-10 17:05:00, 200...
3307818    [2008-11-20 13:43:00, 2008-11-26 19:44:00, 200...
3307819    [2008-11-22 14:49:00, 2008-12-19 15:49:00, 200...
Name: cout, dtype: object
```

```python
time_dict = dict.fromkeys([str(x) for x in range(2007,2023)], 0)
for kv in fp.items():
    time_dict[kv[1][0][0:4]]+=1
print(time_dict)
```

Album Fearless, Overview:

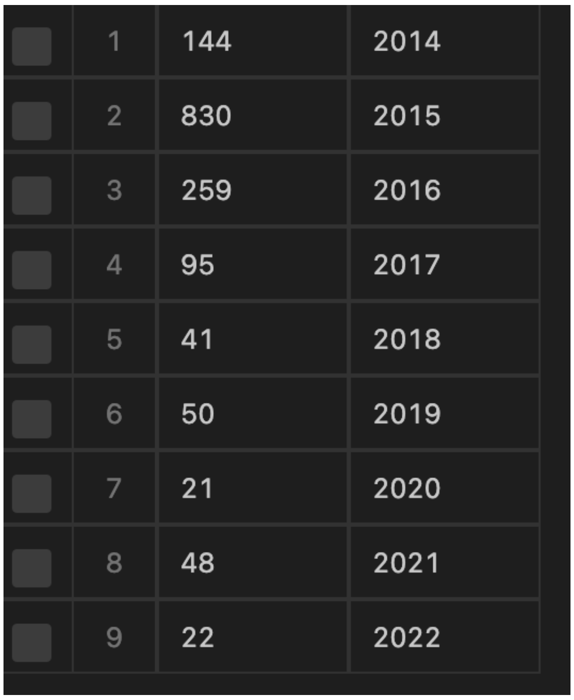
- Fearless: 2008.8 - 2022.7
  - Average Duration:
    - 1021.13 days (avg)
    - 1031 days (All items)
    - Each CD on average last for 3 years
  - Purchase:
    - 

      ```
      {'2007': 0, '2008': 26, '2009': 20, '2010': 0, '2011': 0, '2012': 0,
      '2013': 0, '2014': 3, '2015': 5, '2016': 0, '2017': 0, '2018': 0,
      '2019': 4, '2020': 0, '2021': 0, '2022': 0}
      ```
  - Checkouts:

- 
```
95,2008
1333,2009
938,2010
441,2011
184,2012
134,2013
56,2014
106,2015
64,2016
38,2017
13,2018
48,2019
7,2020
21,2021
19,2022
```

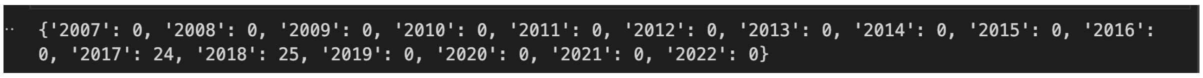
Album 1989, Overview:

- 1989: 2014.10 - 2022.9
  - Average Duration:
    - 551.12 days
    - 712 days (All items)
  - Purchase:

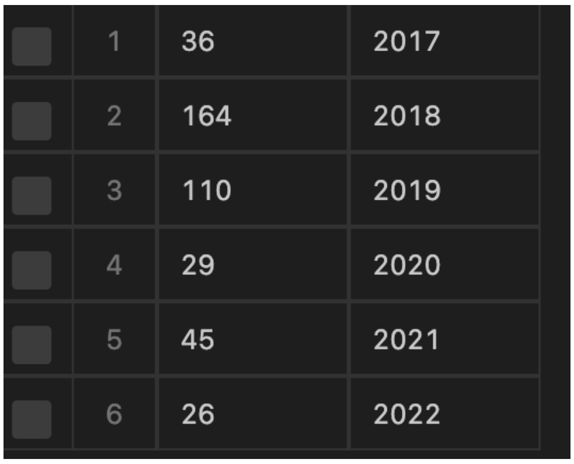
    -
      ```
      {'2007': 0, '2008': 0, '2009': 0, '2010': 0, '2011': 0, '2012': 0, '2013': 0, '2014': 65, '2015': 17, '2016': 0, '2017': 0, '2018': 0, '2019': 0, '2020': 0, '2021': 0, '2022': 0}
      ```
  - Borrow:
    - 

1	144	2014
2	830	2015
3	259	2016
4	95	2017
5	41	2018
6	50	2019
7	21	2020
8	48	2021
9	22	2022

Album Reputation, Overview:

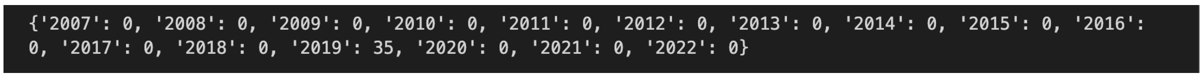
- Reputation: 2017.12-2022.09
  - Average Duration:
    - The average is 262.93 days
    - 695 days (All items)
  - Purchase:
    - 

```
{'2007': 0, '2008': 0, '2009': 0, '2010': 0, '2011': 0, '2012': 0, '2013': 0, '2014': 0, '2015': 0, '2016':
0, '2017': 24, '2018': 25, '2019': 0, '2020': 0, '2021': 0, '2022': 0}
```

  - Borrow:
    - 

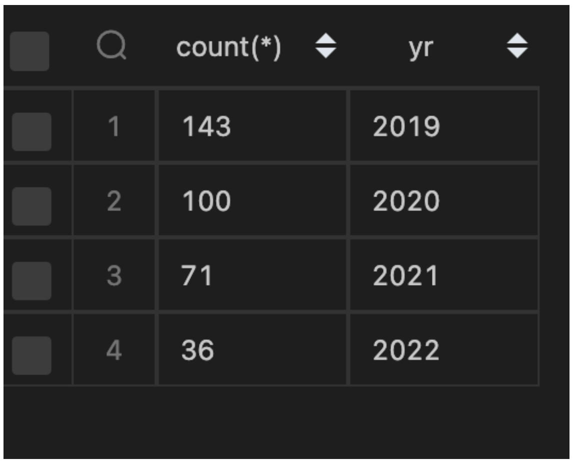
1	36	2017
2	164	2018
3	110	2019
4	29	2020
5	45	2021
6	26	2022

Album Lover, Overview:

- Lover: 2019.09-2022.09
  - Average Duration:
    - 88.5 days
    - 651 days (All items)
  - Purchase:
    - 

```
{'2007': 0, '2008': 0, '2009': 0, '2010': 0, '2011': 0, '2012': 0, '2013': 0, '2014': 0, '2015': 0, '2016':
0, '2017': 0, '2018': 0, '2019': 35, '2020': 0, '2021': 0, '2022': 0}
```

  - Borrow:



	count(*)	yr
1	143	2019
2	100	2020
3	71	2021
4	36	2022

Overall Visualization

- The visualization: Borrow_Purchase.html

Conclusion:

- It’s interesting that people would like to check out consecutively released albums together.
- Assumptions affect the result of the FP-algorithm in our case.
- For some items, purchase and checkout times can be hugely different.
- CDs last much shorter than expected. A single CD lasts around 3 years and then becomes inactive.
- Often the library purchase albums in consecutive 2 years after the album is released. They seldomly purchase new copies afterward.
- If take all CDs by Taylor Swift into account, each album last around 2 years.
- Music streaming service did impact the checkout amount of the CD.
  - 1989 has double the sale of Fearless, but the checkout times is almost halved.