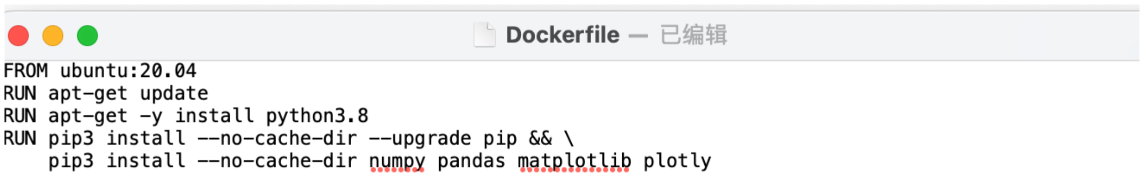
Assignment1

Yang Yao

January 13th 2023

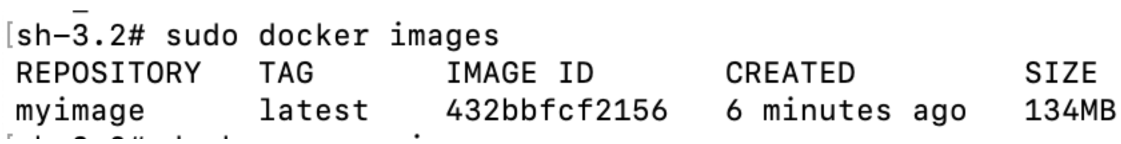
1 Docker

Install Docker and create image through Dockerfile. Here is my what my Dockerfile looks like.



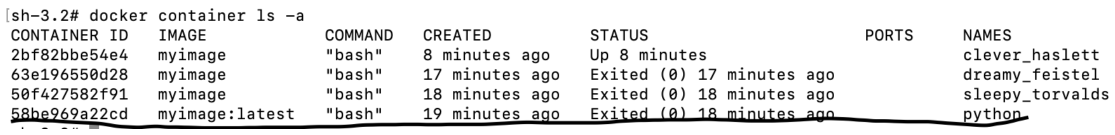
```
FROM ubuntu:20.04
RUN apt-get update
RUN apt-get -y install python3.8
RUN pip3 install --no-cache-dir --upgrade pip && \
    pip3 install --no-cache-dir numpy pandas matplotlib plotly
```

Then execute docker build command and now I have my image myimage:



```
[sh-3.2# sudo docker images
 REPOSITORY   TAG      IMAGE ID       CREATED         SIZE
 myimage      latest   432bbfcf2156   6 minutes ago   134MB
```

Run the image and the container list is as shown below:

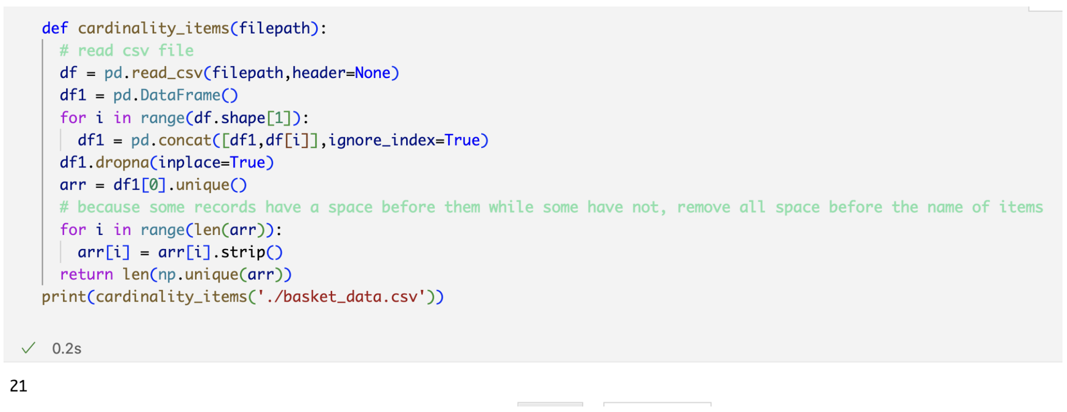


```
[sh-3.2# docker container ls -a
 CONTAINER ID   IMAGE            COMMAND   CREATED          STATUS                      PORTS     NAMES
 2bf82bbe54e4   myimage          "bash"    8 minutes ago    Up 8 minutes                          clever_haslett
 63e196550d28   myimage          "bash"    17 minutes ago   Exited (0) 17 minutes ago             dreamy_feistel
 50f427582f91   myimage          "bash"    18 minutes ago   Exited (0) 18 minutes ago             sleepy_torvalds
 58be969a22cd   myimage:latest   "bash"    19 minutes ago   Exited (0) 18 minutes ago             python
```

2 Identifying All Sets

2.1 question 1

I use pandas library to solve the question. Use read_csv function to read csv file and convert it to DataFrame. Then remove duplicate records and the rest are unique items. Next, I just have to count the number.



```python
def cardinality_items(filepath):
    # read csv file
    df = pd.read_csv(filepath,header=None)
    df1 = pd.DataFrame()
    for i in range(df.shape[1]):
        df1 = pd.concat([df1,df[i]],ignore_index=True)
    df1.dropna(inplace=True)
    arr = df1[0].unique()
    # because some records have a space before them while some have not, remove all space before the name of items
    for i in range(len(arr)):
        arr[i] = arr[i].strip()
    return len(np.unique(arr))
print(cardinality_items('./basket_data.csv'))
```

✓ 0.2s

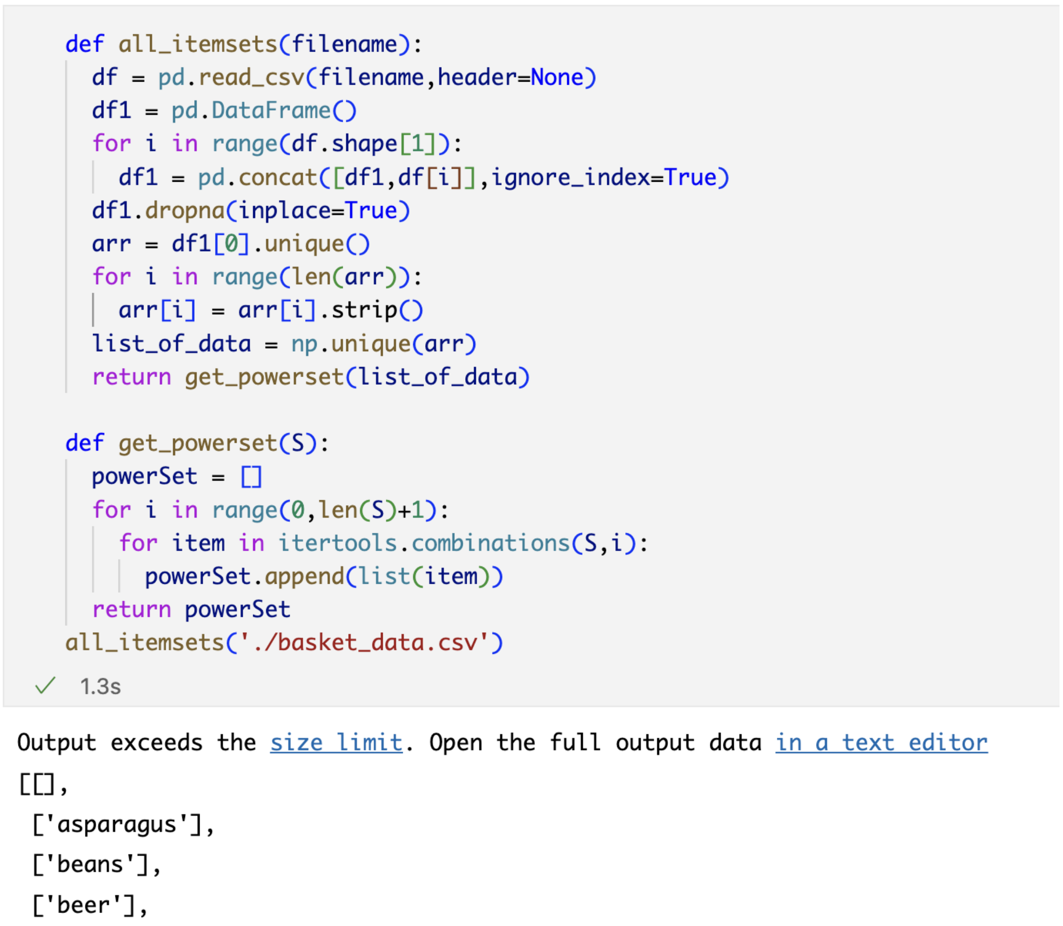
```
21
```

2.2 question 2

All possible sets that form by unique items is power set. The cardinality of power set of set of size n is 2^n.That is because every item in the set can be included or not included in a subset, so there are $(\underbrace{2 * 2 * 2 * ...2 * 2}_{\text{Total n twos}}) = 2^n$ different subsets.
And because we are asked to ignore null set, so the number of sets with unique items can possibly be is $2^n - 1$, where n is the number of unique items in the dataset.

2.3 question 3

To generate all possible subsets from N unique items, we should firstly deduplicate the dataset. And then, generate the power set of N unique items. Here I use itertools.combinations(S,n) function to generate all possible subsets with the size of n from set S. So I just need to generate all possible with the size from 0 to N, then I gain the power set.



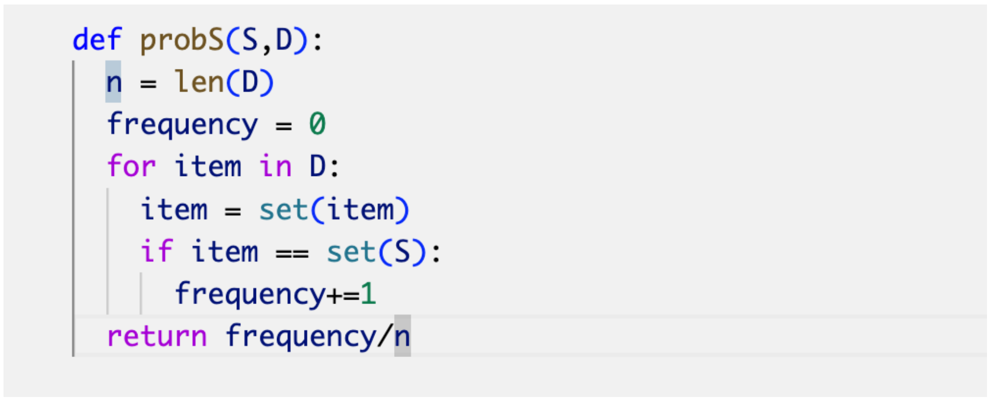
```
def all_itemsets(filename):
    df = pd.read_csv(filename,header=None)
    df1 = pd.DataFrame()
    for i in range(df.shape[1]):
        df1 = pd.concat([df1,df[i]],ignore_index=True)
    df1.dropna(inplace=True)
    arr = df1[0].unique()
    for i in range(len(arr)):
        arr[i] = arr[i].strip()
    list_of_data = np.unique(arr)
    return get_powerset(list_of_data)

def get_powerset(S):
    powerSet = []
    for i in range(0,len(S)+1):
        for item in itertools.combinations(S,i):
            powerSet.append(list(item))
    return powerSet
all_itemsets('./basket_data.csv')
```

✓ 1.3s

```
Output exceeds the size limit. Open the full output data in a text editor
[[],
 ['asparagus'],
 ['beans'],
 ['beer'],
```

2.4 question 4

Count the number of items in D and count the occurrences of S, then you get the probability.



```
def probS(S,D):
    n = len(D)
    frequency = 0
    for item in D:
        item = set(item)
        if item == set(S):
            frequency+=1
    return frequency/n
```

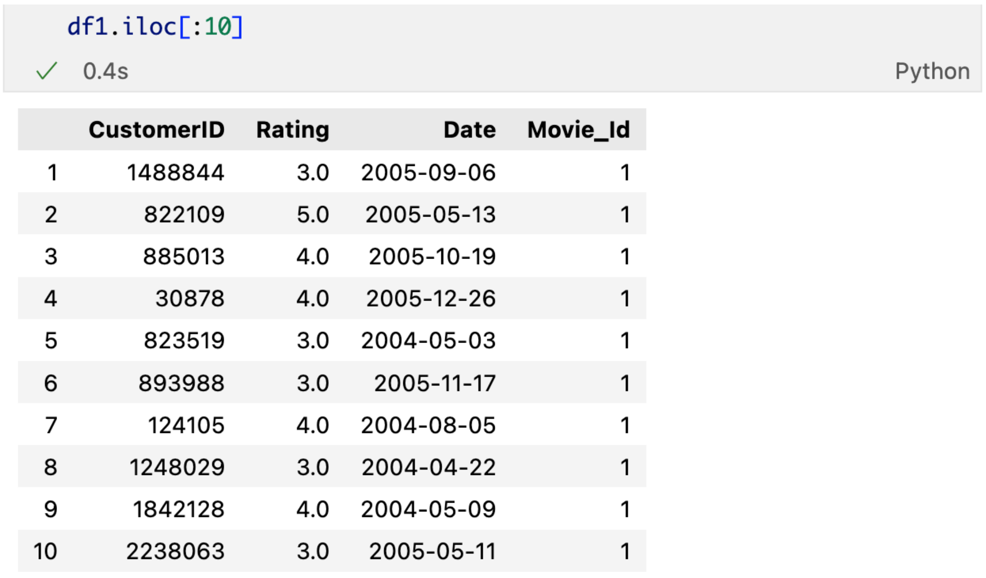
3 The Netflix Challenge

3.1 Data Verification

As dataset description said, every data has 4 colons: MovieID, CustomerID, Ratting and Date.

```
TRAINING DATASET FILE DESCRIPTION
================================================================================

The file "training_set.tar" is a tar of a directory containing 17770 files, one
per movie.  The first line of each file contains the movie id followed by a
colon.  Each subsequent line in the file corresponds to a rating from a customer
and its date in the following format:

CustomerID,Rating,Date

- MovieIDs range from 1 to 17770 sequentially.
- CustomerIDs range from 1 to 2649429, with gaps. There are 480189 users.
- Ratings are on a five star (integral) scale from 1 to 5.
- Dates have the format YYYY-MM-DD.
```

We can open the combined_data1.txt and find the format is the same as description said. Then I load a part of data and preprocess the data to the format declared in the description, as shown below:

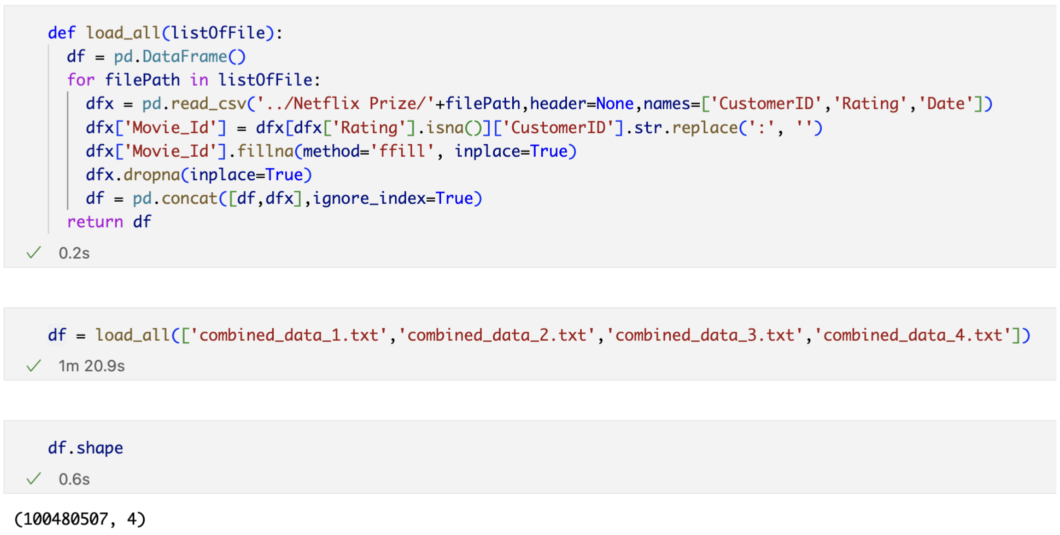


```python
df1.iloc[:10]
```

	CustomerID	Rating	Date	Movie_Id
1	1488844	3.0	2005-09-06	1
2	822109	5.0	2005-05-13	1
3	885013	4.0	2005-10-19	1
4	30878	4.0	2005-12-26	1
5	823519	3.0	2004-05-03	1
6	893988	3.0	2005-11-17	1
7	124105	4.0	2004-08-05	1
8	1248029	3.0	2004-04-22	1
9	1842128	4.0	2004-05-09	1
10	2238063	3.0	2005-05-11	1

3.2 Data Analysis

1. **How many total records are there?**
   There are totally 26847523 records.



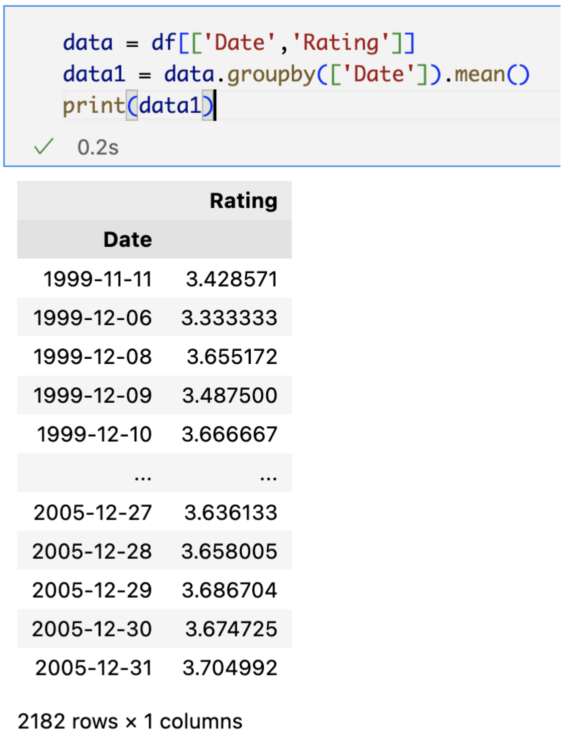
```
def load_all(listOfFile):
    df = pd.DataFrame()
    for filePath in listOfFile:
        dfx = pd.read_csv('../Netflix Prize/'+filePath,header=None,names=['CustomerID','Rating','Date'])
        dfx['Movie_Id'] = dfx[dfx['Rating'].isna()]['CustomerID'].str.replace(':', '')
        dfx['Movie_Id'].fillna(method='ffill', inplace=True)
        dfx.dropna(inplace=True)
        df = pd.concat([df,dfx],ignore_index=True)
    return df
```
✓ 0.2s

```
df = load_all(['combined_data_1.txt','combined_data_2.txt','combined_data_3.txt','combined_data_4.txt'])
```
✓ 1m 20.9s

```
df.shape
```
✓ 0.6s

```
(100480507, 4)
```

2. **Can you plot the distribution of star ratings over users and time? The granularity of the sliding window is at your discretion. Are there any trends?** To gain the distribution of ratings over time, group the rating by Date, then calculate average ratings for each Date:

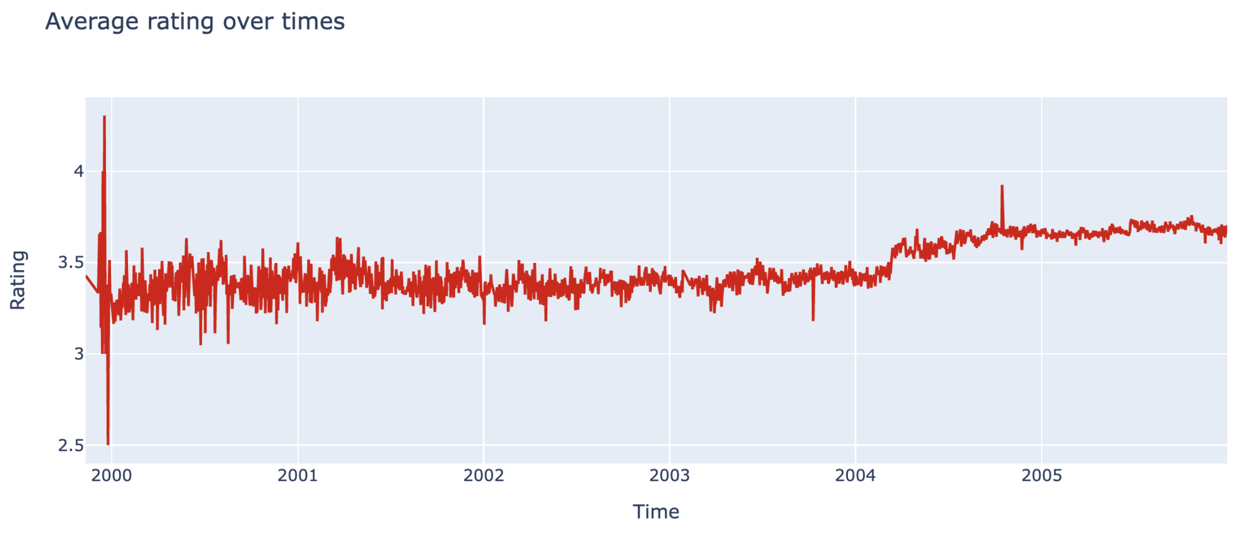


```
data = df[['Date','Rating']]
data1 = data.groupby(['Date']).mean()
print(data1)
```
✓ 0.2s

	Rating
Date	
1999-11-11	3.428571
1999-12-06	3.333333
1999-12-08	3.655172
1999-12-09	3.487500
1999-12-10	3.666667
...	...
2005-12-27	3.636133
2005-12-28	3.658005
2005-12-29	3.686704
2005-12-30	3.674725
2005-12-31	3.704992

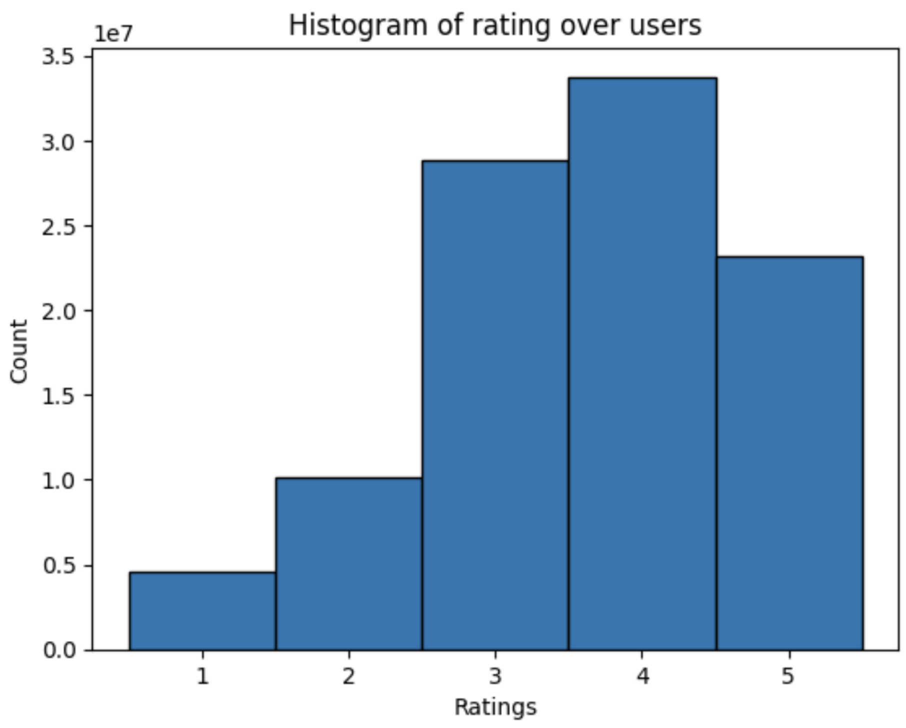
2182 rows × 1 columns

I plot the graph about the average rating stars over time, as shown below.



It shows the rating stars are shock greatly near 2000. I think might be some excellent movies come at the end of 1999 and soon after that, because customers had seen those excellent movies, later movies were rated low for their relatively low quality comparing to excellent movies released right before them. And after 2004, ratings start growing higher, smoother and show upward trend and at Oct 2004 reach regional peak. In total, the rating stars show a slow upward trend.

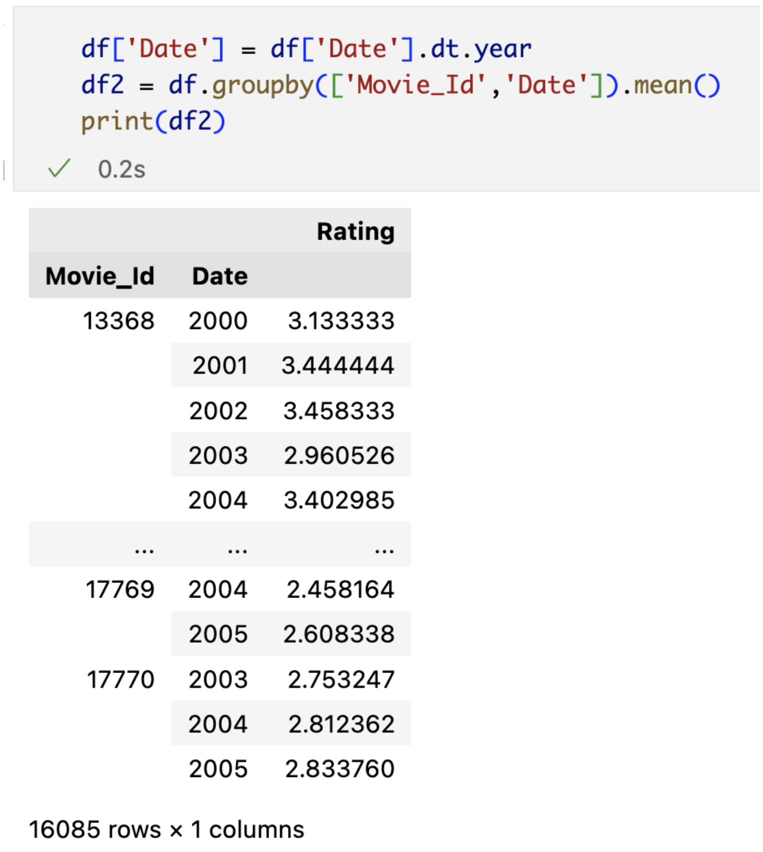
Also, I plot a histogram shows ratings over users. It shows most users rate 4 stars and 1 or 2 star rating is relatively rear.



3. **What percentage of the films have gotten more popular over time?**

   There are about 64% movies have gotten more popular over time. To clarify, a movie becomes more popular over time if the average film rating in the final year is greater than that of the first year. For example, if a movie got 3.5 stars on average in 2000 and 3.7 stars on average in 2004, I can say this movie have gotten more popular over the years.

   To gain the answer, I round data info to year and group the data by MovieId and Date, then calculate it's mean. Then I get data as shown below:



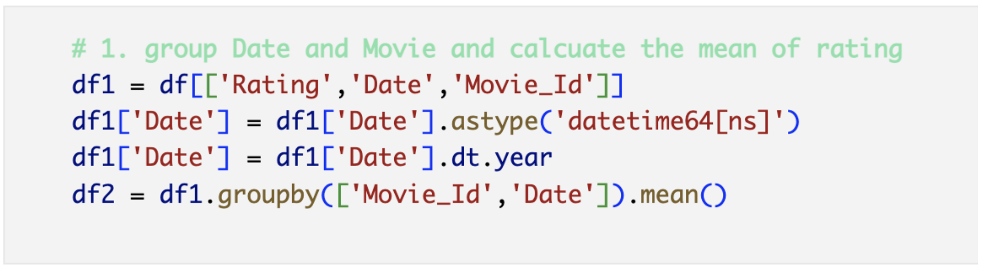
```python
df['Date'] = df['Date'].dt.year
df2 = df.groupby(['Movie_Id','Date']).mean()
print(df2)
```

✓ 0.2s

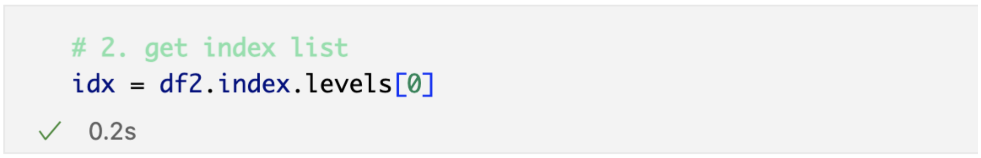
		Rating
Movie_Id	**Date**	
13368	2000	3.133333
	2001	3.444444
	2002	3.458333
	2003	2.960526
	2004	3.402985
...	...	...
17769	2004	2.458164
	2005	2.608338
17770	2003	2.753247
	2004	2.812362
	2005	2.833760

16085 rows × 1 columns

Then I count how many movies got popular over time. I go through all MoiveId and check if its last year average rating greater than the first year. Finally, calculate the percentage:

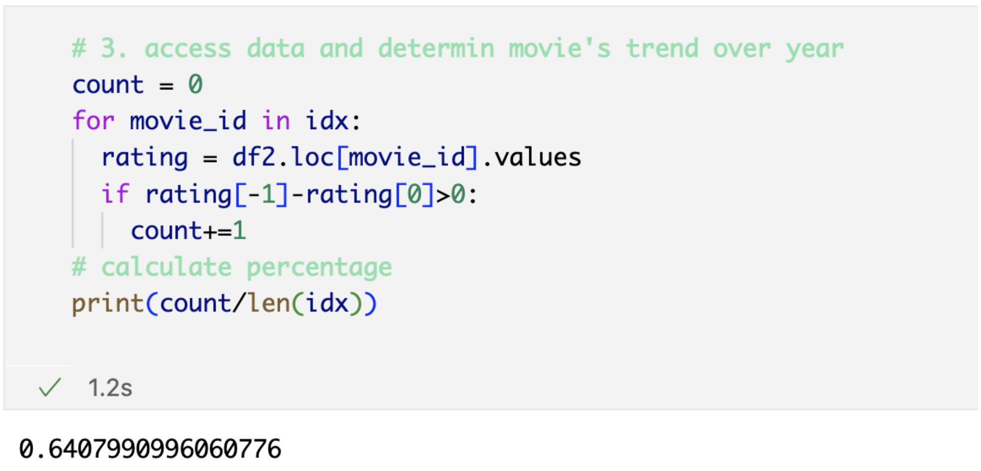


```python
# 1. group Date and Movie and calcuate the mean of rating
df1 = df[['Rating','Date','Movie_Id']]
df1['Date'] = df1['Date'].astype('datetime64[ns]')
df1['Date'] = df1['Date'].dt.year
df2 = df1.groupby(['Movie_Id','Date']).mean()
```



```python
# 2. get index list
idx = df2.index.levels[0]
```

✓ 0.2s



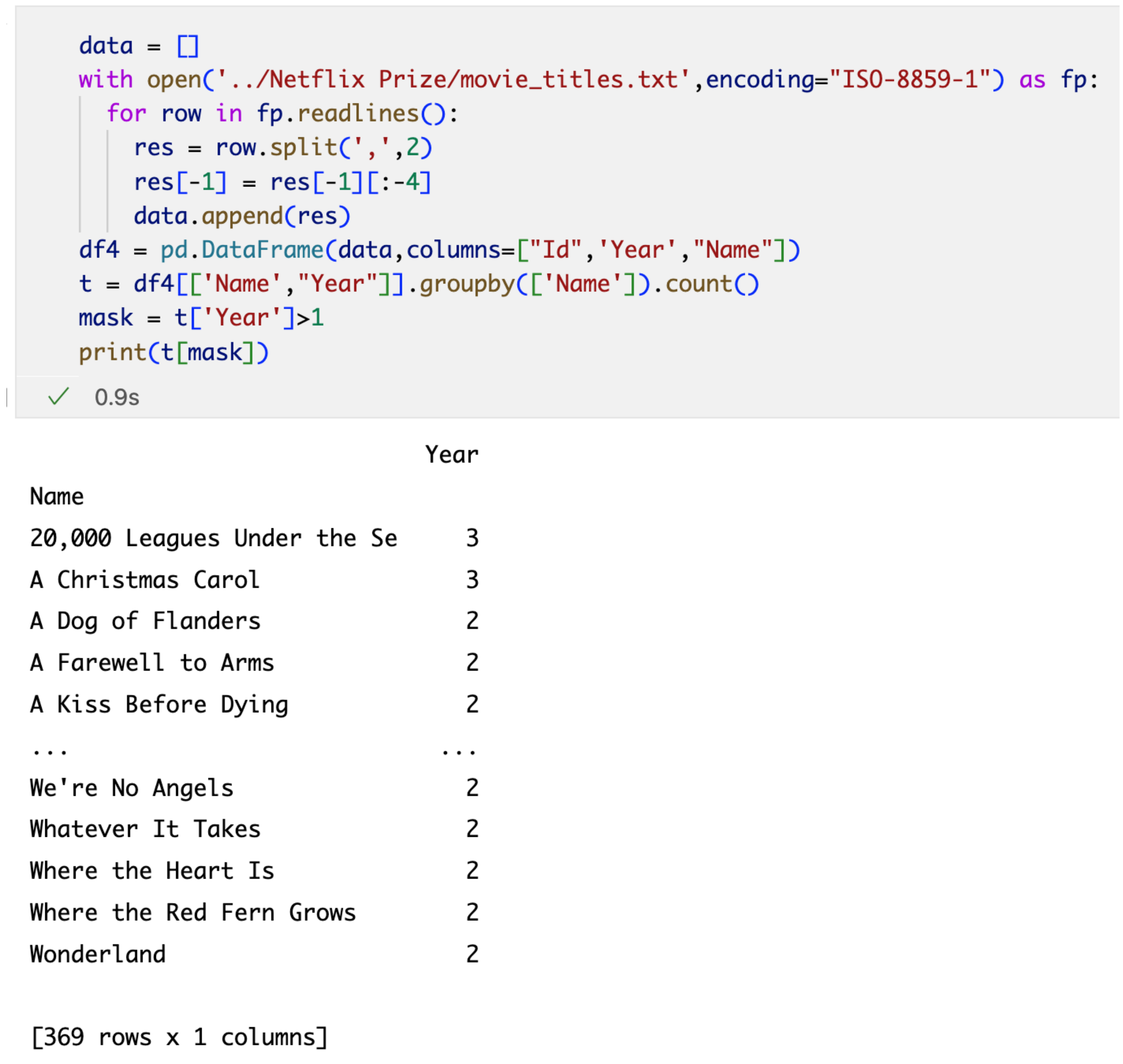
```python
# 3. access data and determin movie's trend over year
count = 0
for movie_id in idx:
    rating = df2.loc[movie_id].values
    if rating[-1]-rating[0]>0:
        count+=1
# calculate percentage
print(count/len(idx))
```

✓ 1.2s

```
0.6407990996060776
```

4. **How many films have been re-released? How do you know?**
There are 369 movies have been re-released. If a movie has been re-released, it should have more than one record in the file "movie_titles.csv" with the same name. So just have to count how many movie names have more than one record. By the way, the "movie_titles.csv" is original a txt file and Kaggle convert it into csv file, but this causes some problems. The csv file use ',' to split data, but some movie name include ',', which leads some movies' name to be separated to several pieces and may lead to wrong answer.

   Here I change "movie_titles.csv" 's extension name to 'txt' and read raw data manually, then convert data to pandas DataFrame and execute groupby operation. Next just need to count how many movies meet the conditions. The code is as shown below:

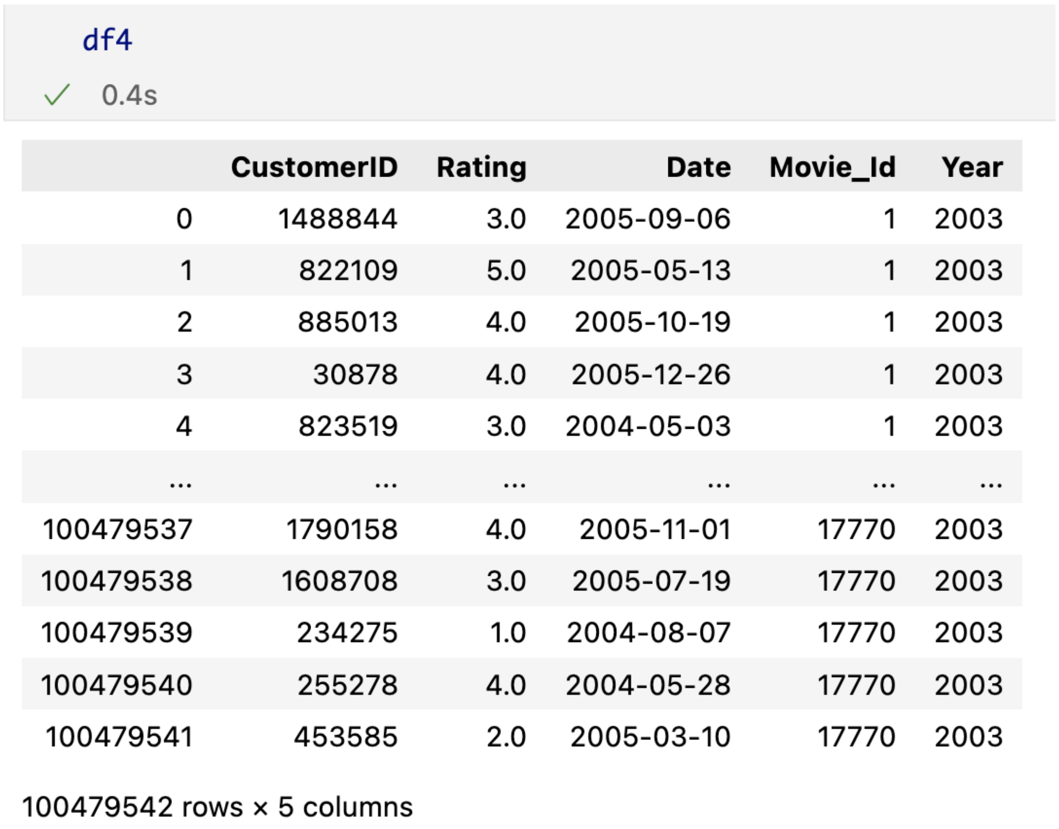


```python
data = []
with open('../Netflix Prize/movie_titles.txt',encoding="ISO-8859-1") as fp:
    for row in fp.readlines():
        res = row.split(',',2)
        res[-1] = res[-1][:-4]
        data.append(res)
df4 = pd.DataFrame(data,columns=["Id",'Year',"Name"])
t = df4[['Name',"Year"]].groupby(['Name']).count()
mask = t['Year']>1
print(t[mask])
```

```
✓ 0.9s
```

```
                             Year
Name
20,000 Leagues Under the Se     3
A Christmas Carol               3
A Dog of Flanders               2
A Farewell to Arms              2
A Kiss Before Dying             2
...                           ...
We're No Angels                 2
Whatever It Takes               2
Where the Heart Is              2
Where the Red Fern Grows        2
Wonderland                      2

[369 rows x 1 columns]
```

5. **What other information might we try to extract to better understand the data? For the questions that you may come up with (especially any time series data), make sure you back up your assertions with plots. Go ahead and play around with the data, and explore.**
I have tried to extract information of relation between rating stars and time since movies released. I merge combined_data with movie_titles so that we have information of released year for every movie. The data looks like below

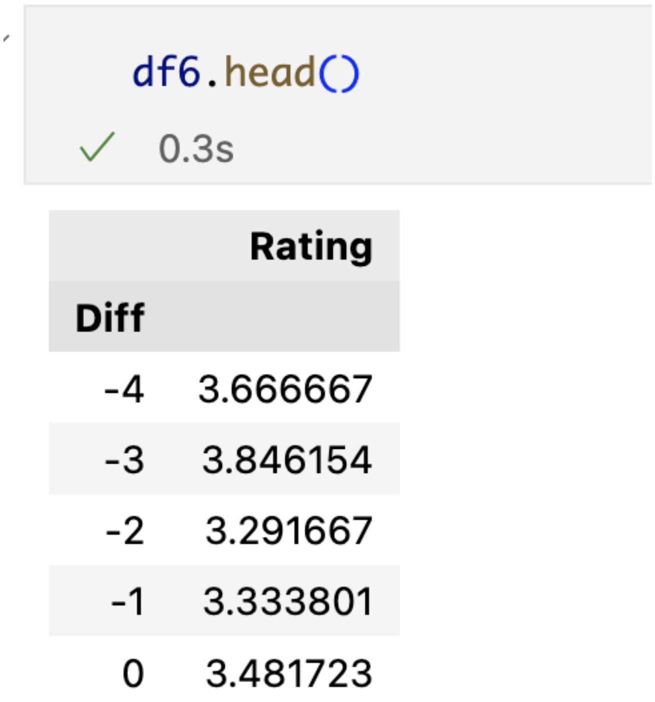


```
df4
✓ 0.4s
```

	CustomerID	Rating	Date	Movie_Id	Year
0	1488844	3.0	2005-09-06	1	2003
1	822109	5.0	2005-05-13	1	2003
2	885013	4.0	2005-10-19	1	2003
3	30878	4.0	2005-12-26	1	2003
4	823519	3.0	2004-05-03	1	2003
...	...	...	...	...	...
100479537	1790158	4.0	2005-11-01	17770	2003
100479538	1608708	3.0	2005-07-19	17770	2003
100479539	234275	1.0	2004-08-07	17770	2003
100479540	255278	4.0	2004-05-28	17770	2003
100479541	453585	2.0	2005-03-10	17770	2003

100479542 rows × 5 columns

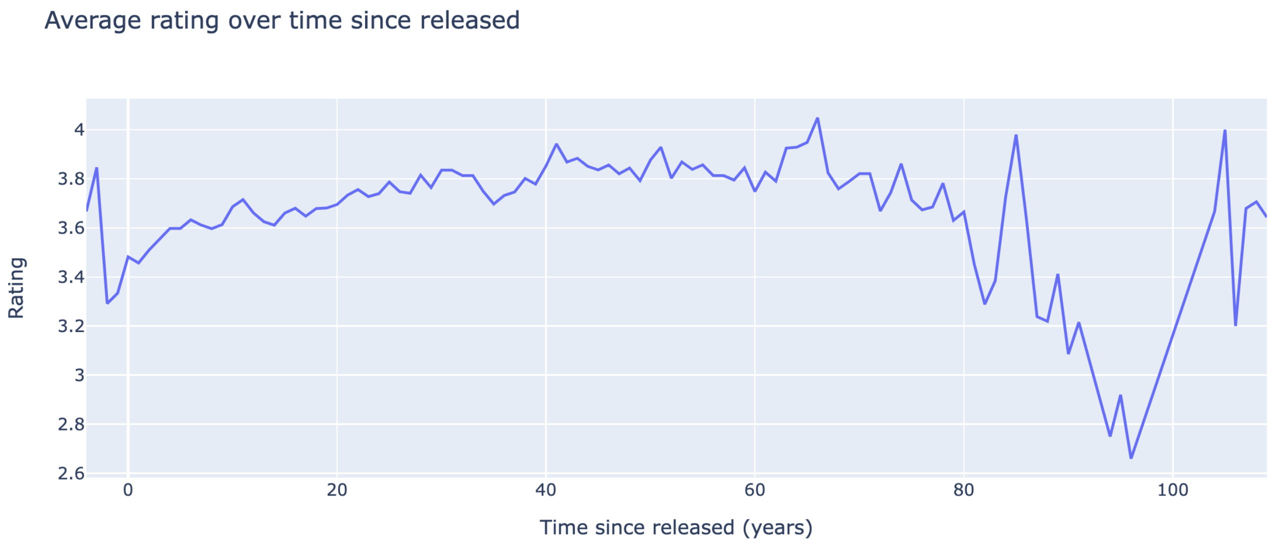
Then we make difference between Date and Year, and calculate the mean then we get ratings over time since released:



```
df6.head()
✓ 0.3s
```

	Rating
Diff	
-4	3.666667
-3	3.846154
-2	3.291667
-1	3.333801
0	3.481723

Finally, I plot a line to visualize the data:



It shows that the ratings become higher as time since movie release increased in it's first 65 years. And an interesting thing is that the movies released 90 years ago are rated low, I think it might be movies at that time were early movies and were out of date for 21st century costumers. However, the movies released more than 100 years were rated relatively high, may be those movies are the very first movies and have important historical meaning, so audience rate high.

6. **What are some interesting problems that we might solve? (No need to actually solve them!)**
   We might solve the problem of the trend of rating over time and users. So given a date and movie, we may use the thing we found before to predict what stars will a user most likely rates.