亚马逊购物数据(来源与SNAP)

学号：3220232576

姓名：李德盛

GitHub仓库地址：https://github.com/0D1Lee0/data_mining_homework_2

In [2]:
```
#加载相应的库
import numpy as np
import pandas as pd
import networkx as nx
from itertools import combinations
from collections import defaultdict, Counter
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori,fpgrowth, association_rules
```

数据预处理

给定的数据格式为（物品1，物品2），因此构建一个购物有向图

In [2]:
```
#给出的数据格式是（物品1，物品2），因此构建一个购物图
G = nx.DiGraph()
count = 0
filename = 'C:/Users/pipi_s/Desktop/data_mining/amazon/Amazon0302.txt'
with open(filename, 'r') as file:
    # 跳过前n-1行
    for _ in range(4):
        next(file)
    for line in file:
        # 从第n行开始读取数据
        nodes = line.strip().split()
        # 添加边到图中
        G.add_edge(nodes[0], nodes[1])
        count = count + 1
```

In [ ]:
```
# 将长度为k的路径上的节点视为一次购物事务
k = 3
# 获取所有节点对之间的路径
with open('C:/Users/pipi_s/Desktop/data_mining/amazon/shopping.txt', 'w', encoding='ut
    for source in G.nodes():
        for target in G.nodes():
            if source != target:
                for path in nx.all_simple_paths(G,source=source,target=target, cutoff=
                    path_as_string = ', '.join(path)
                    f.write(path_as_string + '\n')
print('done!')
```

In [3]:
```
with open("C:/Users/pipi_s/Desktop/data_mining/amazon/shopping.txt", 'r') as f:
    dataset = [line.strip().split() for line in f]
print(len(dataset))
```

518464

关联规则挖掘

In [9]:
```
# 使用mlxtend的TransactionEncoder进行编码
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

# 使用FP-Growth算法找到频繁项集
frequent_itemsets = fpgrowth(df, min_support=0.0003, use_colnames=True)
if frequent_itemsets.empty:
    print("没有找到频繁项集，请检查数据集或调整min_support的值。")
else:
    # 从频繁项集中生成关联规则
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.

    # 打印关联规则
    print(rules)
```

```
   antecedents consequents   support  confidence        lift
0         (24)         (8,)  0.001254    0.863214   98.643440
1        (142)        (23,)  0.000640    0.864583  197.382357
2         (50)        (30,)  0.000615    0.952239  245.256602
3        (175)        (99,)  0.000600    0.987302  228.009056
4        (177)        (99,)  0.000596    0.990385  228.721055
5        (430)       (100,)  0.000320    0.813725  299.635918
6        (283)        (76,)  0.000312    0.857143  317.426939
7         (96)        (56,)  0.000378    0.855895  360.187376
8        (451)       (158,)  0.000309    0.874317  355.809936
9        (457)       (167,)  0.000309    0.935673  352.809102
10       (364)       (303,)  0.000374    0.866071  281.168978
11       (365)       (303,)  0.000374    0.866071  281.168978
12      (1663)       (449,)  0.000365    0.825328  304.773933
13      (1672)       (481,)  0.000916    0.944334  120.177511
14      (1963)      (1241,)  0.000415    0.860000  258.780638
15       (393)       (368,)  0.000401    0.842105  254.875227
16      (2646)       (976,)  0.000359    0.902913  294.605217
17      (2647)       (976,)  0.000363    0.954315  311.376858
```

In [10]:
```
rules_sorted = rules.sort_values(by=['confidence', 'lift'], ascending=[False, False])
print(rules_sorted)
```

```
   antecedents consequents   support  confidence        lift
4        (177)        (99,)  0.000596    0.990385  228.721055
3        (175)        (99,)  0.000600    0.987302  228.009056
17      (2647)       (976,)  0.000363    0.954315  311.376858
2         (50)        (30,)  0.000615    0.952239  245.256602
13      (1672)       (481,)  0.000916    0.944334  120.177511
9        (457)       (167,)  0.000309    0.935673  352.809102
16      (2646)       (976,)  0.000359    0.902913  294.605217
8        (451)       (158,)  0.000309    0.874317  355.809936
10       (364)       (303,)  0.000374    0.866071  281.168978
11       (365)       (303,)  0.000374    0.866071  281.168978
1        (142)        (23,)  0.000640    0.864583  197.382357
0         (24)         (8,)  0.001254    0.863214   98.643440
14      (1963)      (1241,)  0.000415    0.860000  258.780638
6        (283)        (76,)  0.000312    0.857143  317.426939
7         (96)        (56,)  0.000378    0.855895  360.187376
15       (393)       (368,)  0.000401    0.842105  254.875227
12      (1663)       (449,)  0.000365    0.825328  304.773933
5        (430)       (100,)  0.000320    0.813725  299.635918
```

挖掘结果分析

说明

由于购物图中节点数和边数有点大，节点间的路径需要很大的内存存储，同时需要很久的时间，因此选用部分路径进行关联规则分析。因为路径数据不完整，导致支持度不能选择太高，因此从置信度和支持度进行分析。

分析

置信度前5的关联规则如下

- [177]->[99]，置信度为0.990385，提升度为228.721025
- [175]->[99]，置信度为0.987302，提升度为228.009056
- [2647]->[976]，置信度为0.954315，提升度为311.376851
- [50]->[30]，置信度为0.952239，提升度为245.256602
- [1672]->[481]，置信度为0.944334，提升度为120.177511

因此，在用户浏览商品[177,175,2647,50,1672]时，可以在商品相关页面推荐以下商品[99,99,976,30,481]

DBLP数据集

加载所需要的库

In [ ]:
```
import codecs
import numpy as np
from xml.sax import handler, make_parser
```

数据预处理

1. 先将dblp.xml文件转化为作者列表 authors.txt，以关键词'attention'和'transformer'进行筛选
2. 对authors.txt进行编码，生成authors_encoded.txt和authors_index.txt

In [2]:
```
#将dblp.xml转化为author.txt
paper_tag = (
    'article',
    'inproceedings',
    'proceedings',
    'book',
    'incollection',
    'phdthesis',
    'mastersthesis',
    'www'
)
#以'attention'和'transformer'为关键词进行筛选
keywords = ('attention', 'transformer')
specific_year = 2017
split_str = ' ||| '

class mHandler(handler.ContentHandler):
    def __init__(self,result):
        self.result = result
        self.flag_author = False
        self.flag_title = False
        self.flag_year = False
        self.work_info = {}

    def startElement(self, name, attrs):
        if name == 'year':
            self.flag_year = True
        if name == 'title':
            self.flag_title = True
        if name == 'author':
            self.flag_author = True

    def endElement(self, name):
        if name == 'year':
            self.flag_year = False
        if name == 'title':
            self.flag_title = False
        if name == 'author':
            self.flag_author = False

        if name in paper_tag:
            if self.work_info.get('year', 0) >= specific_year and \
                    (keywords[0] in self.work_info.get('title', '') or \
```

```
                    keywords[1] in self.work_info.get('title', '')):
                self.result.write(str(self.work_info['year'])+split_str)
                self.result.write(str(self.work_info['title'])+split_str)
                authors = self.work_info.get('author', tuple())
                for author in authors:
                    if len(author)>3 and author[-4] == '0':
                        author = author[:-5]
                    self.result.write(author+split_str)
                self.result.write('\r\n')

            self.work_info = {}
            self.flag_write = False
            self.flag_title = False
            self.flag_year = False

    def characters(self, content):
        if self.flag_year:
            self.work_info['year'] = int(content)
        if self.flag_title:
            self.work_info['title'] = str.lower(content)
        if self.flag_author:
            self.work_info['author'] = self.work_info.get('author', tuple()) + (str.lo


def parserDblpXml(source,result):
    handler = mHandler(result)
    parser = make_parser()
    parser.setContentHandler(handler)
    parser.parse(source)
```

In [4]:

```
xml_file_name = r'C:\Users\pipi_s\Desktop\data_mining\dblp_2\dataset\dblp.xml'
source = codecs.open(xml_file_name, 'r', 'utf-8')
result = codecs.open('./authors.txt', 'w', 'utf-8')
parserDblpXml(source,result)
result.close()
source.close()
```

In [5]:

```
# 将author进行编码
split_str = ' ||| '
source = codecs.open('./authors.txt', 'r', 'utf-8')
result = codecs.open('./authors_encoded.txt', 'w', 'utf-8')
index = codecs.open('./authors_index.txt', 'w', 'utf-8')
index_dict = {}
name_id = 0

for line in source:
    words = line.split(split_str)
    length = len(words)
    result.write(words[0] + split_str + words[1] + split_str)
    for i in range(2, length-1):
        name = words[i]
        if not name in index_dict.keys():
            index_dict[name] = [name_id, 1]
            name_id += 1
        else:
            index_dict[name][1] += 1
        result.write(str(index_dict[name][0]) + split_str)
    result.write('\r\n')

for name in index_dict.keys():
    index.write(str(index_dict[name][0]) + split_str \
        + name + split_str \
        + str(index_dict[name][1]) + u'\r\n')
```

```
source.close()
result.close()
index.close()
```

关联规则生成

用FPGrowth生成关联规则

In [4]:
```
import pandas as pd
import time
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.frequent_patterns import association_rules

split_str = ' ||| '
min_sup = 5
min_con = 0.5
alpha = 1
beta = 10
def loadData():
    dataset = {i:[] for i in range(2017, 2025)}
    with open('./authors_encoded.txt', 'r') as encode:
        for line in encode:
            words = line.split(split_str)
            length = len(words) - 1
            dataLine = [int(words[i]) for i in range(2, length)]
            dataset[int(words[0])].append(dataLine)
    return dataset

def readAuthorIndex():
    author_dict = {}
    with open('./authors_index.txt', 'r') as authors_index:
        for name in authors_index:
            name = name.strip().split(split_str)
            if len(name) != 3: continue
            author_dict[int(name[0])] = [name[1], name[2]]
    return author_dict
```

In [13]:
```
print('loading the dataset...')
dataset = loadData()
author_dict = readAuthorIndex()

print('change the format to one-hot...')
df = {}
te_len = {}
co_authors = {}
teams = {}
for year in range(2017, 2025):
    data = dataset[year]
    te = TransactionEncoder()
    te_ary = te.fit(data).transform(data)
    df[year] = pd.DataFrame(te_ary, columns=te.columns_)
    te_len[year] = te_ary.shape[0]

print('get co-authors & teams...')
df_co_authors = pd.DataFrame(columns=['year', 'authors', 'papers', 'active'])
df_teams = pd.DataFrame(columns=['year', 'authors', 'papers', 'active'])

for year in range(2017, 2025):
    # get frequent itemsets
```

```
    frequent_itemsets = fpgrowth(df[year], min_support = min_sup/te_len[year], use_col
    # get rules
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=mi

    co_authors = frequent_itemsets[frequent_itemsets.itemsets.apply(lambda x: len(x))
    teams = frequent_itemsets[frequent_itemsets.itemsets.apply(lambda x: len(x)) > 2]
    co_authors_set = set([tuple(fs) for fs in co_authors['itemsets']])
    teams_set = set([tuple(fs) for fs in teams['itemsets']])

    rules_set = set([
        (tuple(fs[0]), tuple(fs[1])) for fs in
            zip(rules['antecedents'].tolist(), rules['consequents'].tolist())
    ])

    # remove not confident relationships
    for rel in co_authors_set:
        stay = True
        for i in range(2):
            ok = False
            for rule in rules_set:
                if rel[0:1] == rule[0] and rel[1:2] == rule[1]:
                    ok = True
                    break
            if not ok:
                stay = False
                break
        if not stay:
            co_authors = co_authors[co_authors.itemsets.apply(lambda x: tuple(x)) !=
    for rel in teams_set:
        stay = True
        for i in range(len(rel)):
            ok = False
            for rule in rules_set:
                if rel[:i] + rel[i+1:] == rule[0] and (rel[i],) == rule[1]:
                    ok = True
                    break
            if not ok:
                stay = False
                break
        if not stay:
            teams = teams[teams.itemsets.apply(lambda x: tuple(x)) != rel]

    # write df
    for encode_lists in co_authors['itemsets']:
        authors = []
        active = 0.0
        # get author name and active/group_papers
        for encode in encode_lists:
            authors.append(author_dict[encode][0])
            active += 1/int(author_dict[encode][1])
        active += alpha + beta / len(authors)
        authors = tuple(authors)
        # get group_papers
        for index, rows in co_authors.iterrows():
            if encode_lists == rows['itemsets']:
                papers = int(float(rows['support']) * te_len[year])
                active *= papers
                break
        df_co_authors.loc[len(df_co_authors)] = [year, authors, papers, active]
    for encode_lists in teams['itemsets']:
        authors = []
        active = 0.0
        # get author name and active/group_papers
        for encode in encode_lists:
```

```
            authors.append(author_dict[encode][0])
            active += 1/int(author_dict[encode][1])
        active += alpha + beta / len(authors)
        authors = tuple(authors)
        # get group_papers
        for index, rows in teams.iterrows():
            if encode_lists == rows['itemsets']:
                papers = int(float(rows['support']) * te_len[year])
                active *= papers
                break
        df_teams.loc[len(df_teams)] = [year, authors, papers, active]

print("Writing result into csv...")
df_co_authors.to_csv(
    r'C:\Users\pipi_s\Desktop\data_mining\dblp_2\result\result_co_authors_' + str(min_s
    index=False
)
df_teams.to_csv(
     r'C:\Users\pipi_s\Desktop\data_mining\dblp_2\result\result_teams_' + str(min_sup)
    index=False
)
```

```
loading the dataset...
change the format to one-hot...
get co-authors & teams...
Writing result into csv...
```

可视化分析

In [5]:
```
co_authors = pd.read_csv(r'C:\Users\pipi_s\Desktop\data_mining\dblp_2\result\result_co_
teams = pd.read_csv(r'C:\Users\pipi_s\Desktop\data_mining\dblp_2\result\result_teams_5_
```

In [15]:
```
co_authors.head(10)
```

Out[15]:

	year	authors	papers	active
0	2017	('hanwang zhang', 'tat-seng chua')	5	30.513158
1	2017	('zhou zhao', 'yueting zhuang')	6	36.392857
2	2017	('kang liu', 'jun zhao')	5	30.401515
3	2017	('haewoon kwak', 'jisun an')	6	37.500000
4	2017	('edison marrese-taylor', 'yutaka matsuo')	6	36.878788
5	2017	('takaaki hori', 'shinji watanabe')	5	30.250000
6	2017	('lianli gao', 'heng tao shen')	5	30.247059
7	2017	('xiaogang wang', 'wanli ouyang')	6	36.266667
8	2017	('guanbin li', 'liang lin')	5	30.261905
9	2018	('yi liu', 'yujiu yang')	6	36.390374

In [16]:
```
teams.head(10)
```

Out[16]:

	year	authors	papers	active
0	2018	('chengqi zhang', 'guodong long', 'jing jiang')	6	26.633333
1	2018	('chengqi zhang', 'tianyi zhou', 'jing jiang')	6	26.819608
2	2018	('chengqi zhang', 'tianyi zhou', 'guodong long')	6	26.852941
3	2018	('tianyi zhou', 'guodong long', 'jing jiang')	6	26.652941
4	2018	('chengqi zhang', 'tianyi zhou', 'guodong long...	6	21.986275
5	2018	('chengqi zhang', 'tao shen', 'tianyi zhou')	6	26.936275
6	2018	('tao shen', 'tianyi zhou', 'jing jiang')	6	26.736275
7	2018	('tao shen', 'tianyi zhou', 'guodong long')	6	26.769608
8	2018	('chengqi zhang', 'tao shen', 'jing jiang')	6	26.716667
9	2018	('chengqi zhang', 'tao shen', 'guodong long')	6	26.750000

In [6]:

```
co_authors_group_size = co_authors.groupby('year').size()
co_authors_group_size
```

Out[6]:

```
year
2017      9
2018     31
2019     91
2020    113
2021    122
2022    370
2023    429
dtype: int64
```

In [7]:

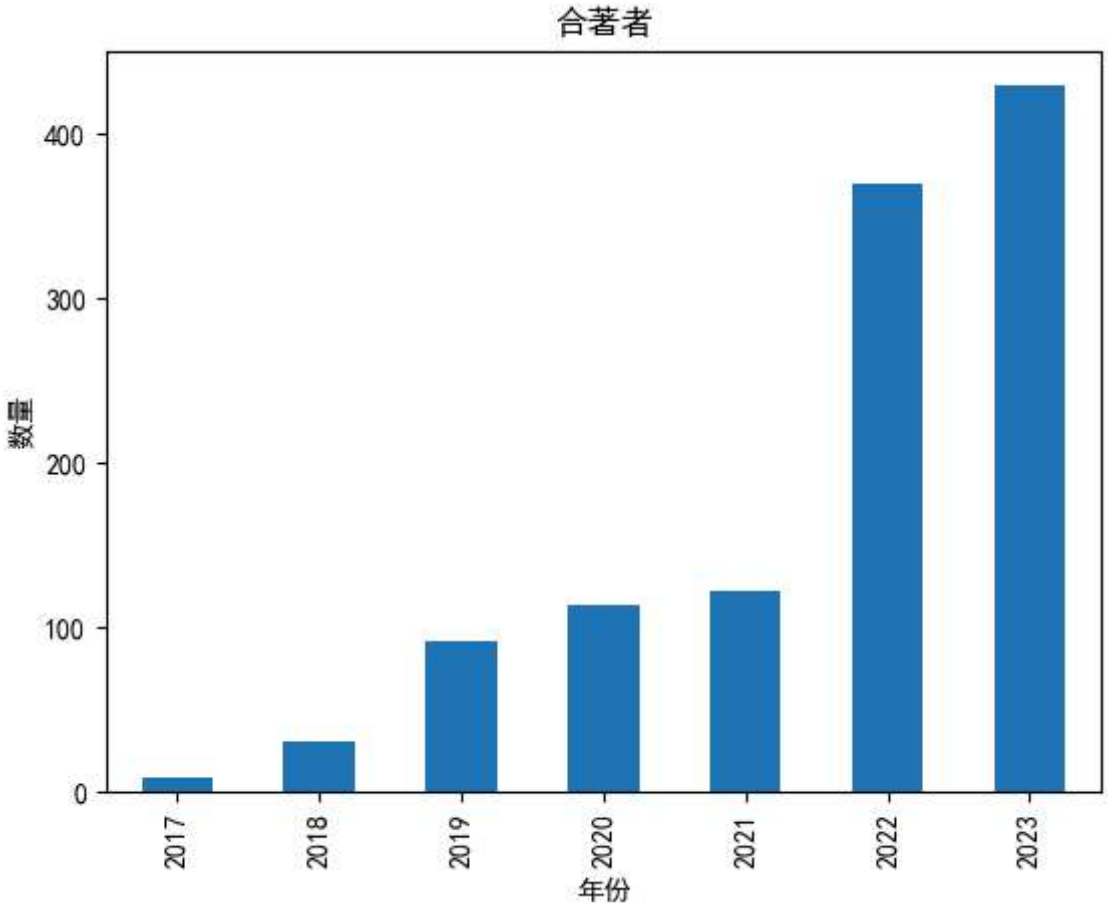
```
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置字体为黑体
plt.rcParams['axes.unicode_minus'] = False
```

In [20]:

```
# 绘制柱形图
co_authors_group_size.plot(kind='bar')

# 添加标题和轴标签
plt.title('合著者')
plt.xlabel('年份')
plt.ylabel('数量')

# 显示图形
plt.show()
```



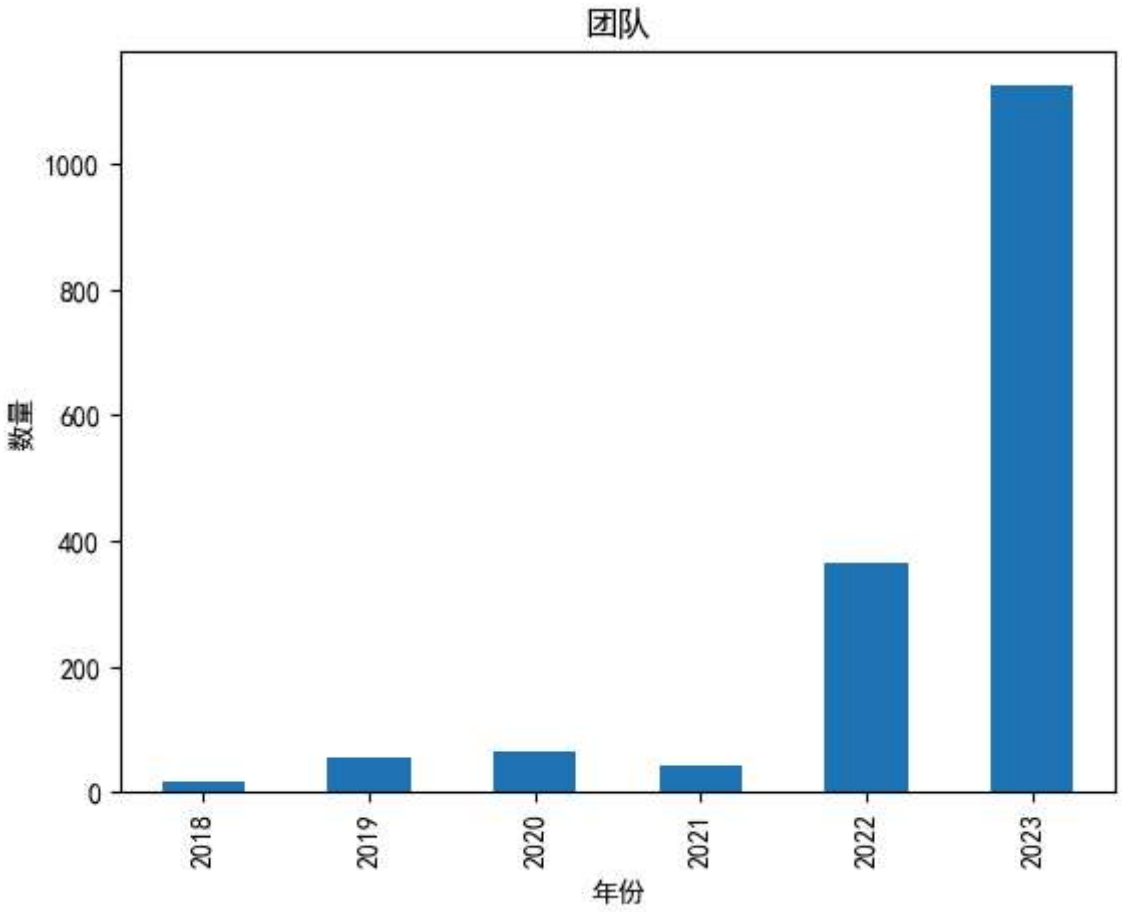
In [9]:
```
teams_group_size = teams.groupby('year').size()
teams_group_size
```

Out[9]:
```
year
2018    16
2019    55
2020    64
2021    43
2022   365
2023  1125
dtype: int64
```

In [21]:
```
# 绘制柱形图
teams_group_size.plot(kind='bar')

# 添加标题和轴标签
plt.title('团队')
plt.xlabel('年份')
plt.ylabel('数量')

# 显示图形
plt.show()
```



In [10]:
```
co_authors_group_size_df = co_authors_group_size.reset_index(name='count')
teams_group_size_df = teams_group_size.reset_index(name='count')
new_data = {'year':2017,'count':0}
teams_group_size_df.loc[-1] = new_data
teams_group_size_df = teams_group_size_df.sort_index()
teams_group_size_df = teams_group_size_df.reset_index(drop=True)
teams_group_size_df
```

Out[10]:

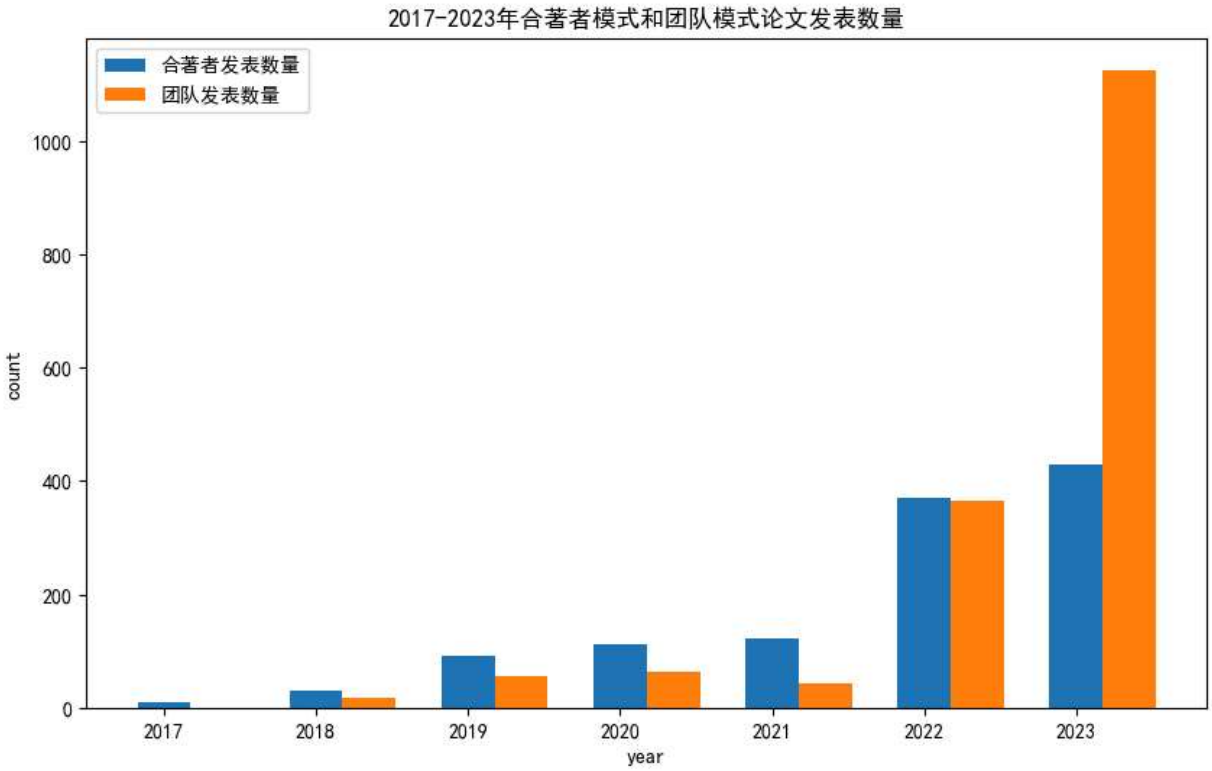
	year	count
0	2017	0
1	2018	16
2	2019	55
3	2020	64
4	2021	43
5	2022	365
6	2023	1125

In [13]:
```
plt.figure(figsize=(10, 6))
bar_width = 0.35
plt.bar(co_authors_group_size_df['year'], co_authors_group_size_df['count'], bar_width,
plt.bar(teams_group_size_df['year']+bar_width, teams_group_size_df['count'], bar_width,
# 添加图例
plt.legend()

# 添加标题和轴标签
plt.title('2017-2023年合著者模式和团队模式论文发表数量')
plt.xlabel('year')
plt.ylabel('count')
```

```
# 设置x轴的刻度位置，以便它们与柱形的中心对齐
#plt.xticks(ind + bar_width / 2, co_authors_group_size_df['year'])

# 显示图形
plt.show()
```



学者关系活跃程度

In [74]:
```
def sort_group(group):
    return group.sort_values(by='papers')
co_authors_sorted = co_authors.groupby('year', as_index=False).apply(sort_group)
teams_sorted = teams.groupby('year', as_index=False).apply(sort_group)
```

2017-2023年活跃程度前5的合著者

In [72]:
```
co_authors_sorted.groupby('year').head(5)
```

Out[72]:

		year	authors	papers	active
0	**0**	2017	('hanwang zhang', 'tat-seng chua')	5	30.513158
	2	2017	('kang liu', 'jun zhao')	5	30.401515
	5	2017	('takaaki hori', 'shinji watanabe')	5	30.250000
	6	2017	('lianli gao', 'heng tao shen')	5	30.247059
	8	2017	('guanbin li', 'liang lin')	5	30.261905
1	**24**	2018	('tianlang chen', 'jiebo luo')	5	30.506567
	36	2018	('shiyu zhou', 'bo xu')	5	30.792411
	35	2018	('shiyu zhou', 'shuang xu')	5	31.098901
	32	2018	('jinsong su', 'deyi xiong')	5	30.400641
	31	2018	('rongwu zhu', 'marco liserre')	5	30.586124
2	**74**	2019	('marco faifer', 'roberto ottoboni')	5	30.625000
	67	2019	('jaewoo kang', 'raehyun kim')	5	30.833333
	96	2019	('xiao-hu zhou', 'zeng-guang hou')	5	30.647773
	95	2019	('zhen-liang ni', 'xiao-hu zhou')	5	30.839161
	94	2019	('gui-bin bian', 'xiao-hu zhou')	5	30.634615
3	**187**	2020	('simon doclo', 'ali aroudi')	5	30.871212
	164	2020	('ankit singh rawat', 'sashank j. reddi')	5	30.884615
	198	2020	('michael l. seltzer', 'yongqiang wang')	5	30.833333
	166	2020	('chulhee yun', 'sashank j. reddi')	5	31.214286
	167	2020	('srinadh bhojanapalli', 'chulhee yun')	5	30.992063
4	**244**	2021	('ois br', 'fran')	6	36.532127
	306	2021	('yifan zhao', 'jia li')	6	36.752351
	364	2021	('hongfei lin', 'bo xu')	6	36.236607
	303	2021	('n fernandez astudillo', 'ram')	6	37.250000
	302	2021	('n fernandez astudillo', 'tahira naseem')	6	37.416667
5	**366**	2022	('yanzhi wang', 'geng yuan')	5	30.502451
	564	2022	('gr', 'goire lefebvre')	5	31.111111
	566	2022	('yang nan', 'guang yang')	5	30.816327
	567	2022	('farnoosh naderkhani', 'arash mohammadi')	5	30.931677
	568	2022	('jiahao huang', 'huanjun wu')	5	31.416667
6	**736**	2023	('shaohui mei', 'mingyang ma')	5	30.902778
	1006	2023	('heinrich dinkel', 'zhiyong yan')	5	32.000000
	1005	2023	('heinrich dinkel', 'yongqing wang')	5	31.833333
	1004	2023	('yujun wang', 'yongqing wang')	5	31.111111
	1003	2023	('junbo zhang', 'yongqing wang')	5	31.127451

2017-2023年活跃程度前5的团队

In [75]:
```
teams_sorted.groupby('year').head(5)
```

Out[75]:

		year	authors	papers	active
0	**15**	2018	('shiyu zhou', 'bo xu', 'shuang xu')	5	22.843693
	0	2018	('chengqi zhang', 'guodong long', 'jing jiang')	6	26.633333
	1	2018	('chengqi zhang', 'tianyi zhou', 'jing jiang')	6	26.819608
	2	2018	('chengqi zhang', 'tianyi zhou', 'guodong long')	6	26.852941
	3	2018	('tianyi zhou', 'guodong long', 'jing jiang')	6	26.652941
1	**43**	2019	('yongfeng huang', 'xing xie', 'fangzhao wu')	5	22.264805
	37	2019	('michele zanoni', 'marco faifer', 'christian ...	5	18.747711
	38	2019	('sergio toscani', 'marco faifer', 'christian ...	5	18.590368
	39	2019	('michele zanoni', 'sergio toscani', 'marco fa...	5	18.662483
	40	2019	('nicholas cummins', 'rn w. schuller', 'bj')	5	22.275506
2	**71**	2020	('yunsong li', 'rui song', 'jiaojiao li')	5	22.228515
	94	2020	('sanjiv kumar', 'srinadh bhojanapalli', 'chul...	5	19.210012
	95	2020	('sanjiv kumar', 'srinadh bhojanapalli', 'chul...	5	19.325397
	97	2020	('yuya moroto', 'keisuke maeda', 'takahiro oga...	5	22.729167
	98	2020	('yuya moroto', 'keisuke maeda', 'miki haseyama')	5	22.729167
3	**135**	2021	('daquan zhou', 'qibin hou', 'jiashi feng')	6	26.955087
	170	2021	('jianlong fu', 'minghao chen', 'houwen peng')	6	26.916863
	160	2021	('pengchuan zhang', 'jianwei yang', 'xiyang dai')	6	27.208791
	159	2021	('lu yuan', 'pengchuan zhang', 'xiyang dai')	6	26.929071
	158	2021	('lu yuan', 'jianwei yang', 'xiyang dai')	6	26.929071
4	**178**	2022	('mengshu sun', 'zhangyang wang', 'haoyu ma')	5	22.546485
	325	2022	('hartwig adam', 'yukun zhu', 'liang-chieh che...	5	16.455662
	326	2022	('hartwig adam', 'yukun zhu', 'liang-chieh che...	5	15.146139
	332	2022	('han zhang', 'lu jiang', 'huiwen chang')	5	22.511484
	336	2022	('sudarshan pant', 'hyung-jeong yang', 'soo-hy...	5	22.819048
5	**1667**	2023	('usman ahmed', 'jerry chun-wei lin', 'gautam ...	5	22.623177
	1556	2023	('zhiding yu', 'jos', ' m. ', 'anima anandkumar')	5	18.517665
	689	2023	('jiawen yao', 'yu shi', 'ling zhang')	5	22.486425
	1555	2023	('zhiding yu', 'lvarez', ' m. ', 'anima anandk...	5	18.626893
	1554	2023	('zhiding yu', 'lvarez', 'jos')	5	22.212324

结论

1. 活跃合著者中，有部分学者重复出现，推测这部分学者是导师，可能符合“导师+学生”、“两个导师”等二人合著模式
2. 活跃团队基本上由3名学者组成，构成三人合作模式
3. 活跃团队中，有部分成员重复；推测他们来自同一个实验室或一个学院，可能符合“两个固定导师+一个学生”、“三个相互熟悉的导师”等三人合作模式。
4. 发表相关论文数量上，合著者发文数量在2018年激增，推测是受2017年发表的论文“Attention is All You Need”影响，之后在2019年，2022年也出现激增；团队发文数量在2022年和2023年出现激增