Pandas进阶-文本信息处理基础 C09

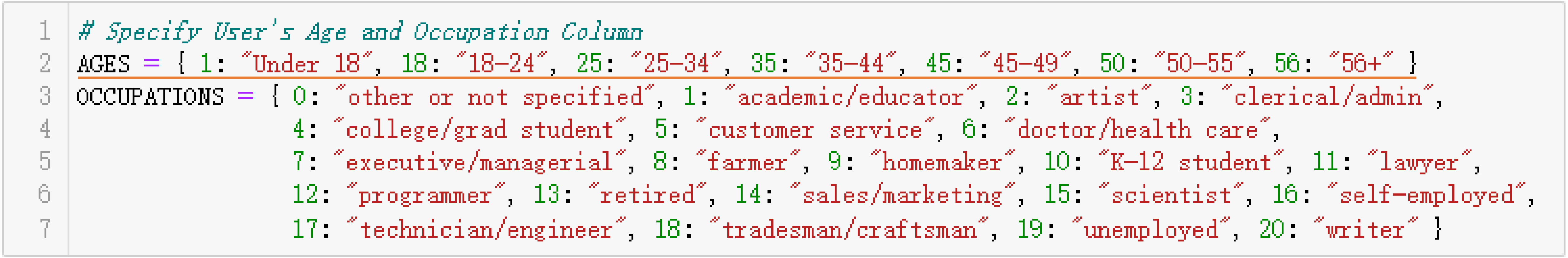
信息科学技术学院

胡俊峰

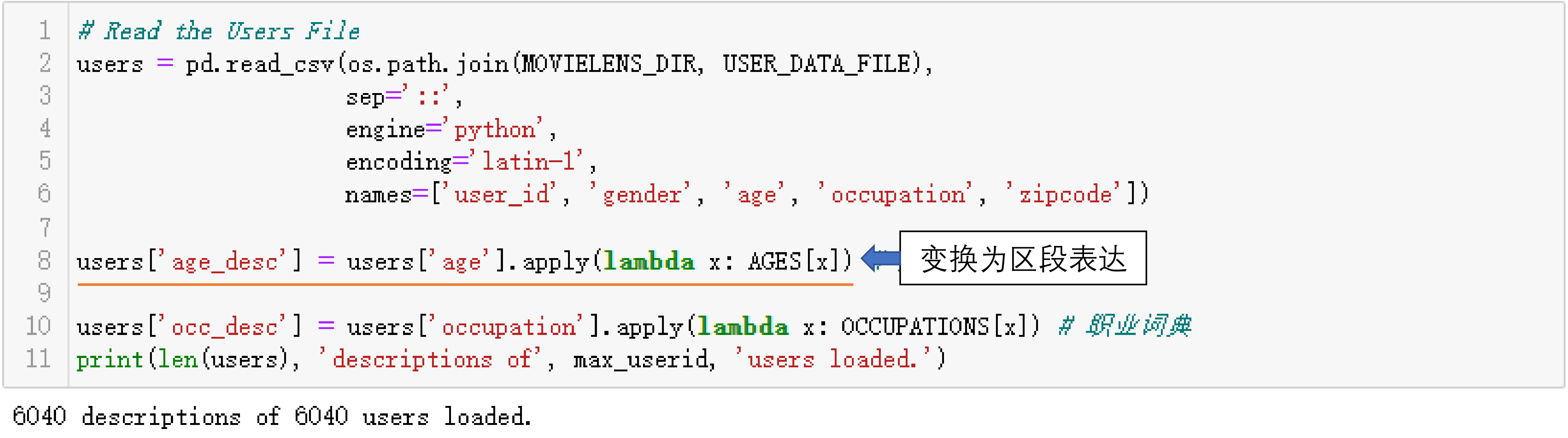
主要内容：

- 表内容统计（续）
- 表连接操作：Merge-join
- 文本处理基础

连续值属性量化



```python
# Specify User's Age and Occupation Column
AGES = { 1: "Under 18", 18: "18-24", 25: "25-34", 35: "35-44", 45: "45-49", 50: "50-55", 56: "56+" }
OCCUPATIONS = { 0: "other or not specified", 1: "academic/educator", 2: "artist", 3: "clerical/admin",
                4: "college/grad student", 5: "customer service", 6: "doctor/health care",
                7: "executive/managerial", 8: "farmer", 9: "homemaker", 10: "K-12 student", 11: "lawyer",
                12: "programmer", 13: "retired", 14: "sales/marketing", 15: "scientist", 16: "self-employed",
                17: "technician/engineer", 18: "tradesman/craftsman", 19: "unemployed", 20: "writer" }
```

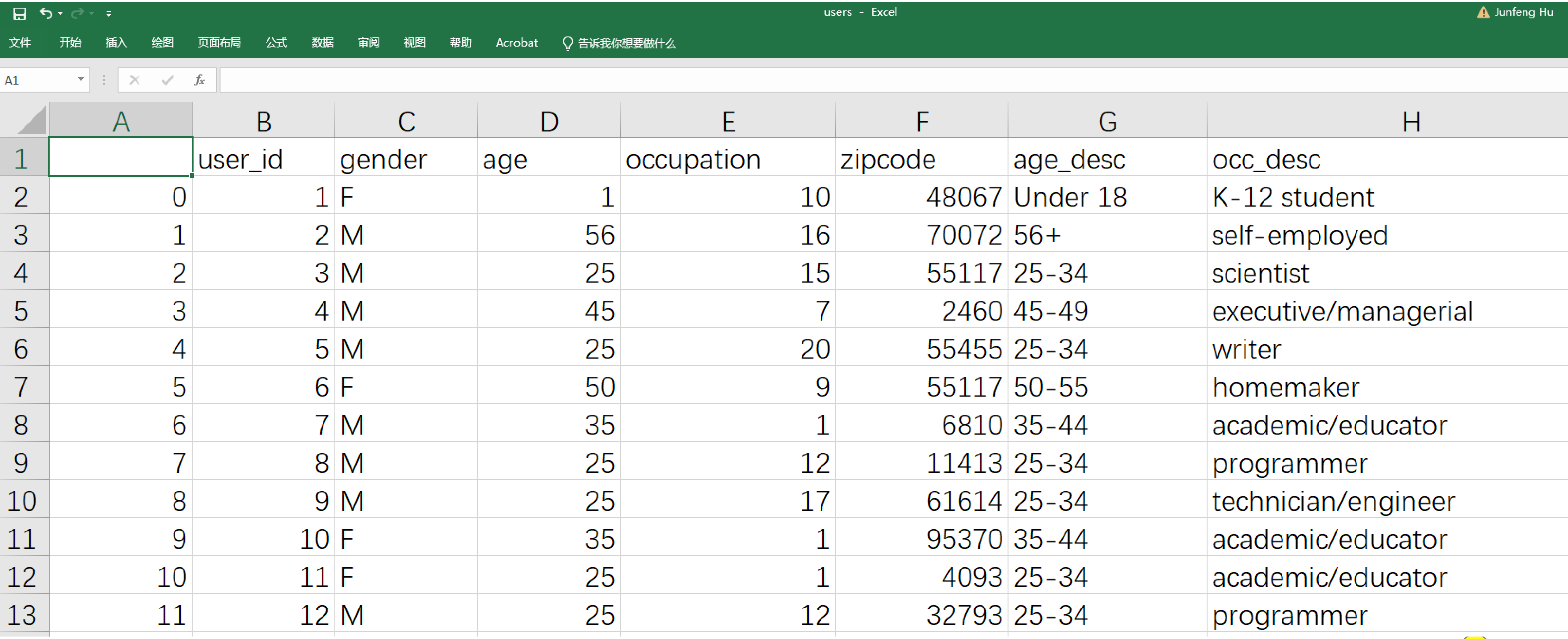


```python
# Read the Users File
users = pd.read_csv(os.path.join(MOVIELENS_DIR, USER_DATA_FILE),
                    sep='::',
                    engine='python',
                    encoding='latin-1',
                    names=['user_id', 'gender', 'age', 'occupation', 'zipcode'])

users['age_desc'] = users['age'].apply(lambda x: AGES[x])

users['occ_desc'] = users['occupation'].apply(lambda x: OCCUPATIONS[x]) # 职业词典
print(len(users), 'descriptions of', max_userid, 'users loaded.')
```

变换为区段表达

```
6040 descriptions of 6040 users loaded.
```

加工后的数据表：


users - Excel
Junfeng Hu
文件 开始 插入 绘图 页面布局 公式 数据 审阅 视图 帮助 Acrobat 告诉我你想要做什么
A1

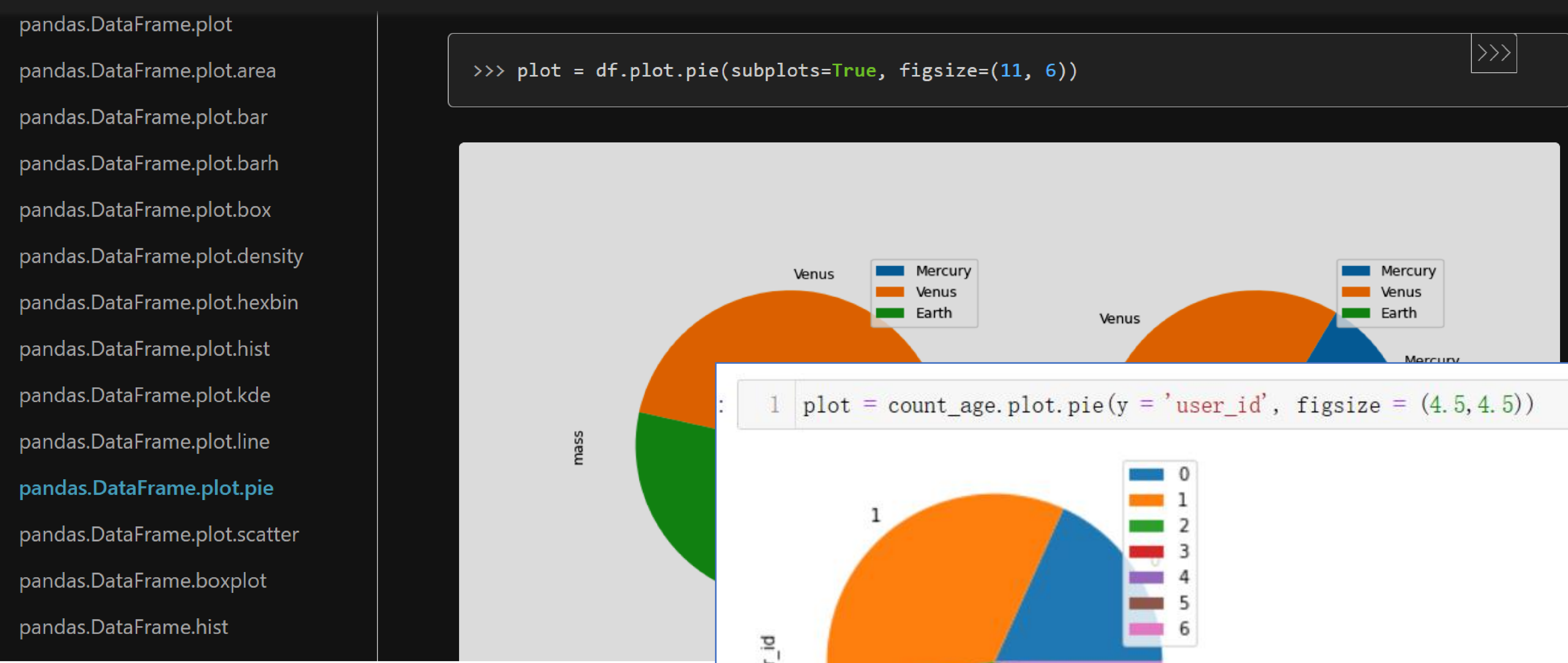
	A	B	C	D	E	F	G	H
1		user_id	gender	age	occupation	zipcode	age_desc	occ_desc
2	0	1	F	1	10	48067	Under 18	K-12 student
3	1	2	M	56	16	70072	56+	self-employed
4	2	3	M	25	15	55117	25-34	scientist
5	3	4	M	45	7	2460	45-49	executive/managerial
6	4	5	M	25	20	55455	25-34	writer
7	5	6	F	50	9	55117	50-55	homemaker
8	6	7	M	35	1	6810	35-44	academic/educator
9	7	8	M	25	12	11413	25-34	programmer
10	8	9	M	25	17	61614	25-34	technician/engineer
11	9	10	F	35	1	95370	35-44	academic/educator
12	10	11	F	25	1	4093	25-34	academic/educator
13	11	12	M	25	12	32793	25-34	programmer

```python
count_age_gender = users.groupby(by = ['age_desc','gender'])['user_id'].count().reset_index()
count_age_gender
```

	age_desc	gender	user_id
0	18-24	F	298
1	18-24	M	805
2	25-34	F	558
3	25-34	M	1538
4	35-44	F	338
5	35-44	M	855
6	45-49	F	189
7	45-49	M	361
8	50-55	F	146
9	50-55	M	350
10	56+	F	102
11	56+	M	278
12	Under 18	F	78
13	Under 18	M	144

```python
print(count_age_gender.describe())
```

```
           user_id
count    14.000000
mean    431.428571
std     399.754100
min      78.000000
25%     156.750000
50%     318.000000
75%     508.750000
max    1538.000000
```



pandas.DataFrame.plot

pandas.DataFrame.plot.area

pandas.DataFrame.plot.bar

pandas.DataFrame.plot.barh

pandas.DataFrame.plot.box

pandas.DataFrame.plot.density

pandas.DataFrame.plot.hexbin

pandas.DataFrame.plot.hist

pandas.DataFrame.plot.kde

pandas.DataFrame.plot.line

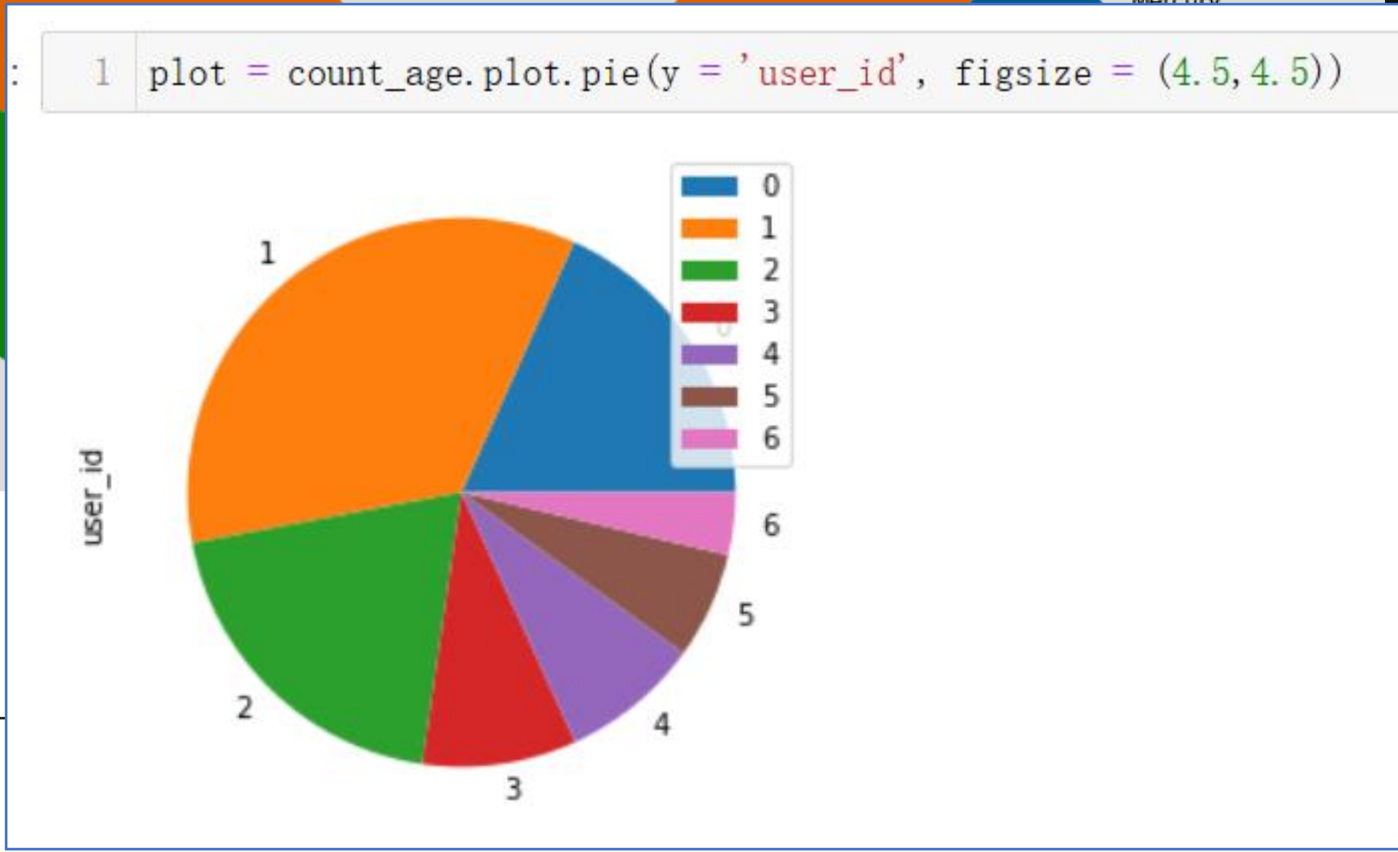
pandas.DataFrame.plot.pie

pandas.DataFrame.plot.scatter

pandas.DataFrame.boxplot

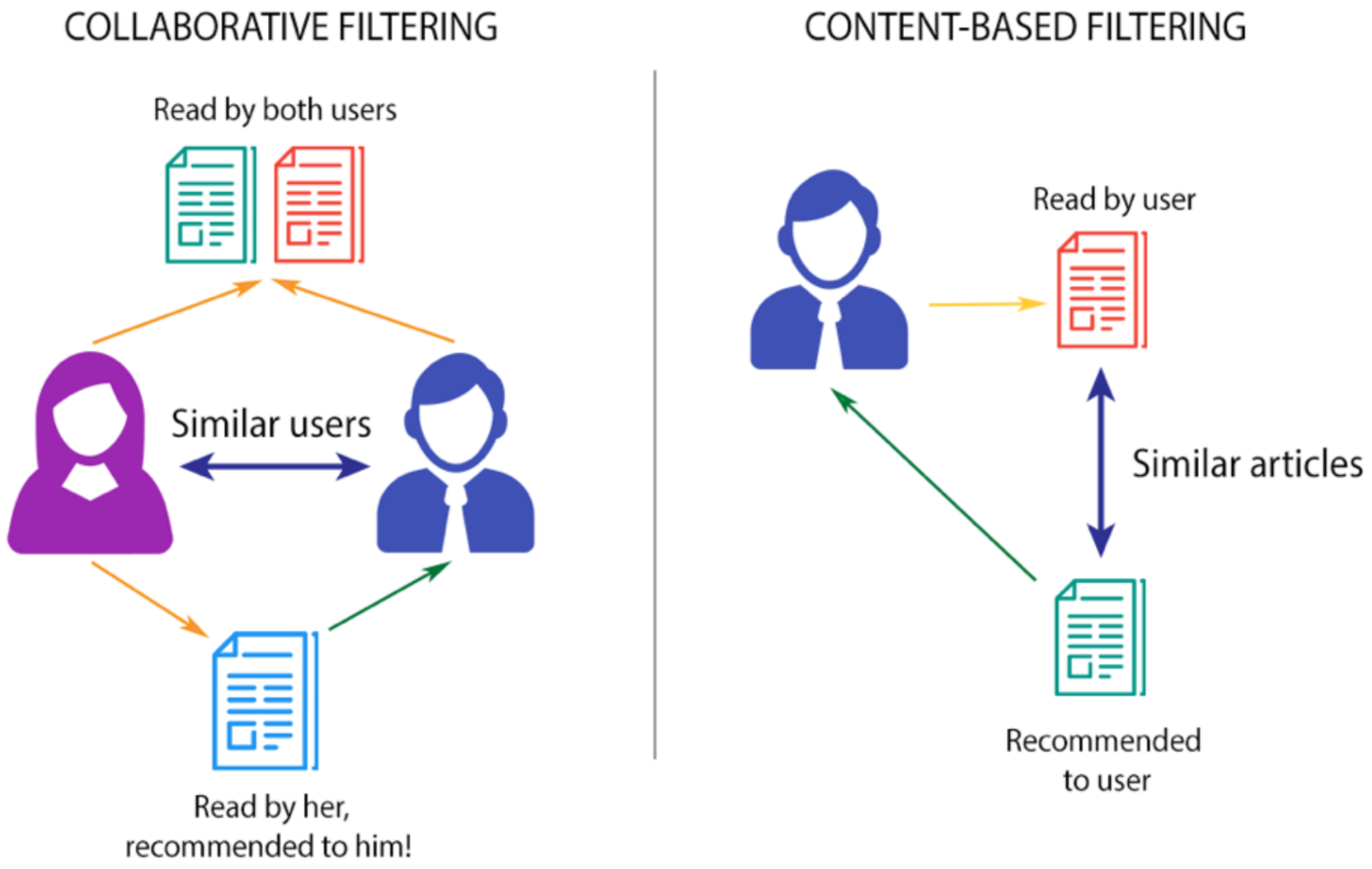
pandas.DataFrame.hist

```
>>> plot = df.plot.pie(subplots=True, figsize=(11, 6))
```

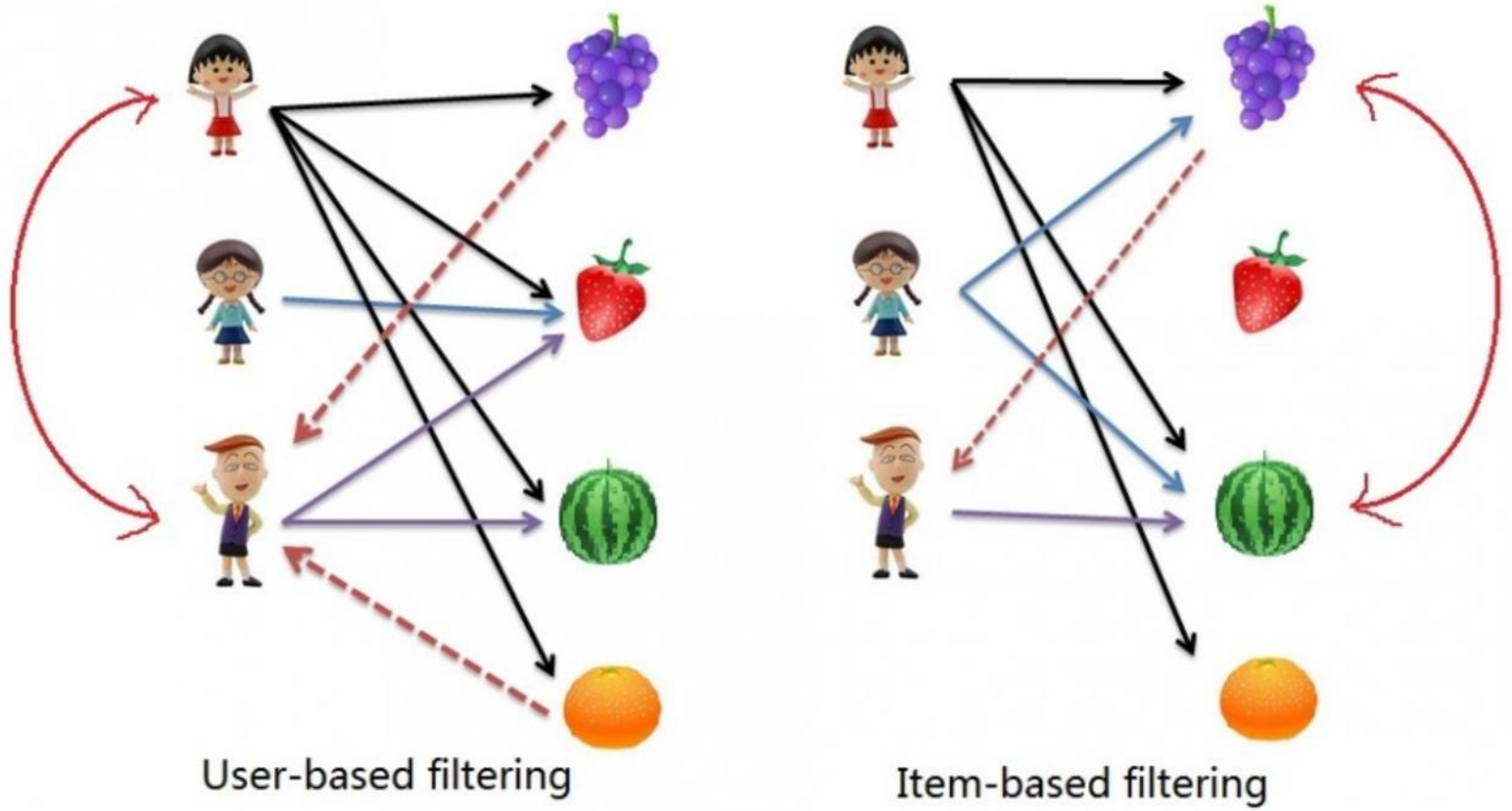


```
plot = count_age.plot.pie(y = 'user_id', figsize = (4.5,4.5))
```

协同过滤推荐：


COLLABORATIVE FILTERING
Read by both users
Similar users
Read by her,
recommended to him!
CONTENT-BASED FILTERING
Read by user
Similar articles
Recommended
to user

1. **User-User Collaborative Filtering**: Here we find look alike users based on similarity and recommend movies which first user’s look-alike has chosen in past. This algorithm is very effective but takes a lot of time and resources. It requires to compute every user pair information which takes time. Therefore, for big base platforms, this algorithm is hard to implement without a very strong parallelizable system.
2. **Item-Item Collaborative Filtering**: It is quite similar to previous algorithm, but instead of finding user's look-alike, we try finding movie's look-alike. Once we have movie's look-alike matrix, we can easily recommend alike movies to user who have rated any movie from the dataset. This algorithm is far less resource consuming than user-user collaborative filtering. Hence, for a new user, the algorithm takes far lesser time than user-user collaborate as we don’t need all similarity scores between users. And with fixed number of movies, movie-movie look alike matrix is fixed over time.



多字段聚合函数:

- Pandas has built-in data aggregation methods,
- such as mean(), sum(), and max().

- Group by certain Key

agg（）

聚合函数为每个组返回单个聚合值。 通过分组系列，还可以传递函数的列表或字典来进行聚合，并生成DataFrame

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

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
         'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
         'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
         'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
         'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

grouped = df.groupby('Team')
agg = grouped['Points'].agg([np.sum, np.mean, np.std])
print (agg)


```

```
         sum        mean         std
Team
Devils  1536  768.000000  134.350288
Kings   2285  761.666667   24.006943
Riders  3049  762.250000   88.567771
Royals  1505  752.500000   72.831998
kings    812  812.000000         NaN
```

https://www.yiibai.com/pandas/python_pandas_groupby.html

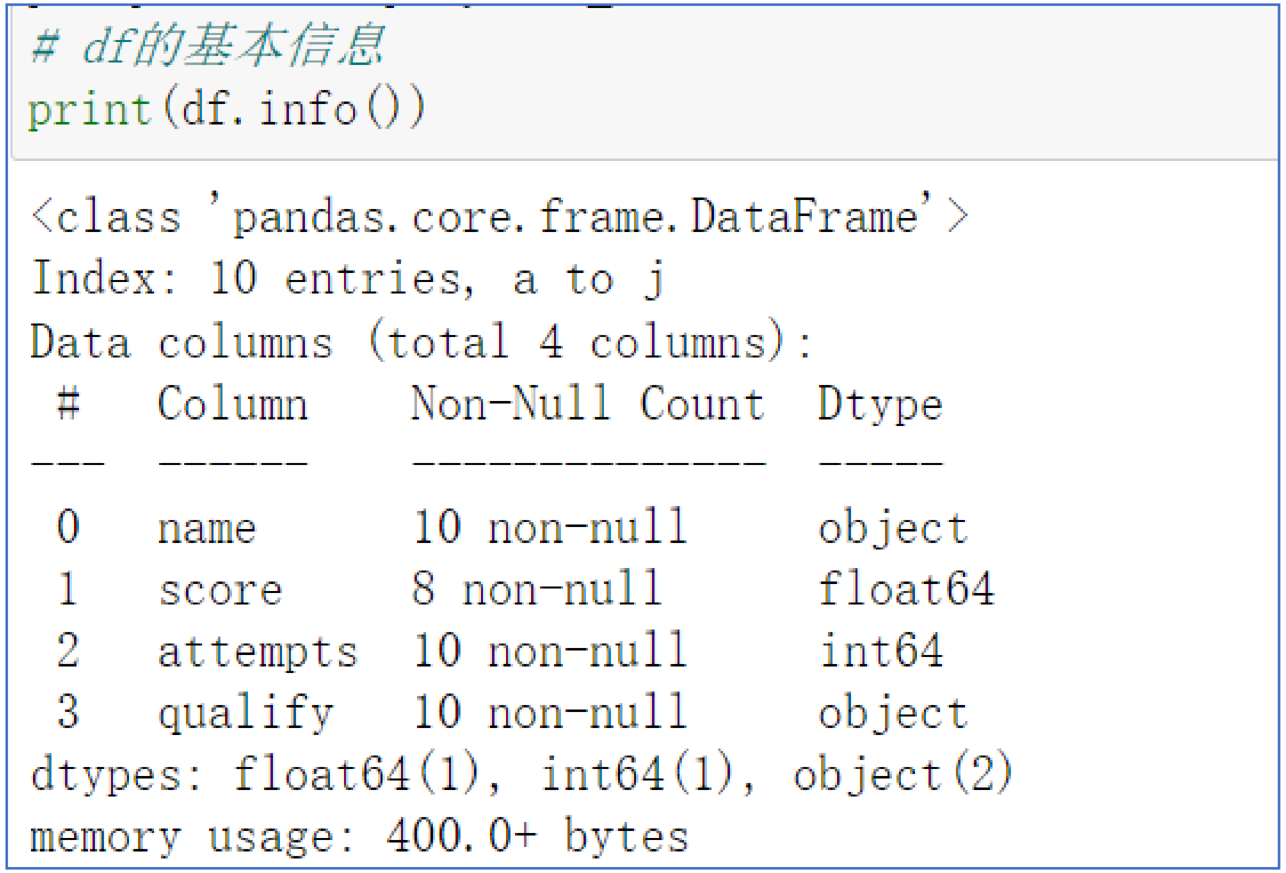
```
pd.options.display.max_rows = 6 # 最多显示6行

exam_data = {
    'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew',
    'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
    'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
    'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']
}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

# 以labels为行索引，生成一个exam_data的DataFrame
df = pd.DataFrame(exam_data, index = labels)
df
```

	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Dima	9.0	3	no
c	Katherine	16.5	2	yes
...	...	...	...	...
h	Laura	NaN	1	no
i	Kevin	8.0	2	no
j	Jonas	19.0	1	yes

10 rows × 4 columns

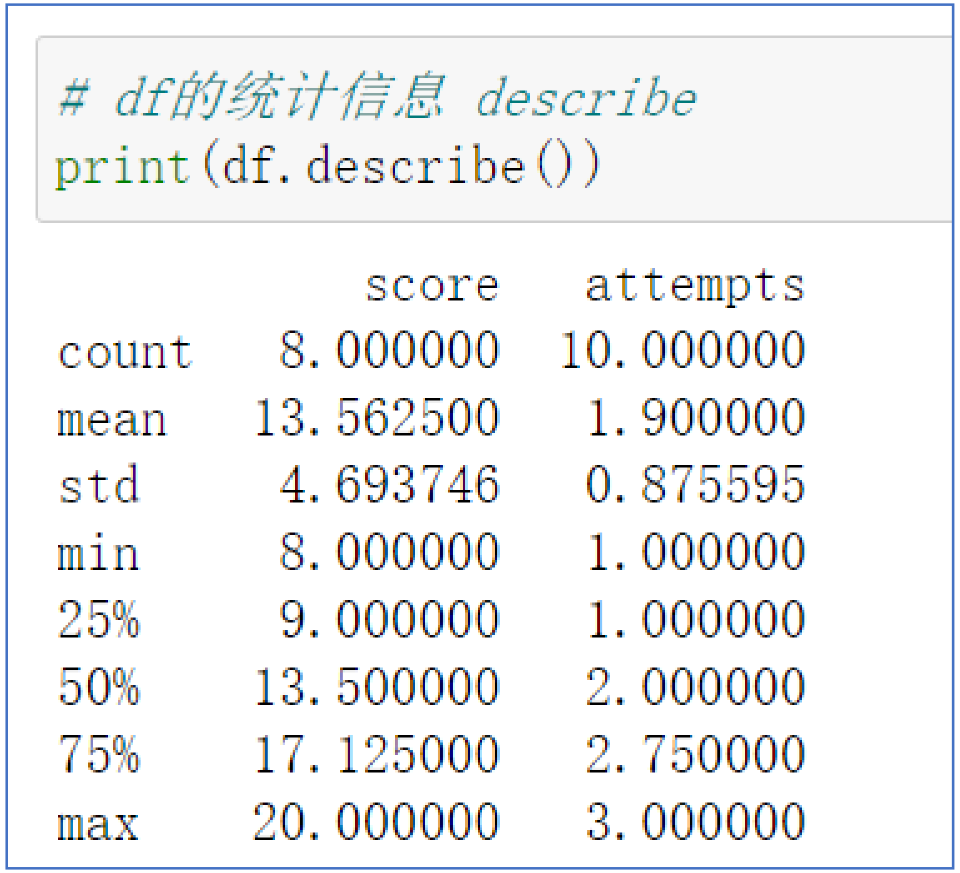
以labels为索引生成一个成绩表

输出数据表的结构信息和基本统计信息



```
# df的基本信息
print(df.info())
```

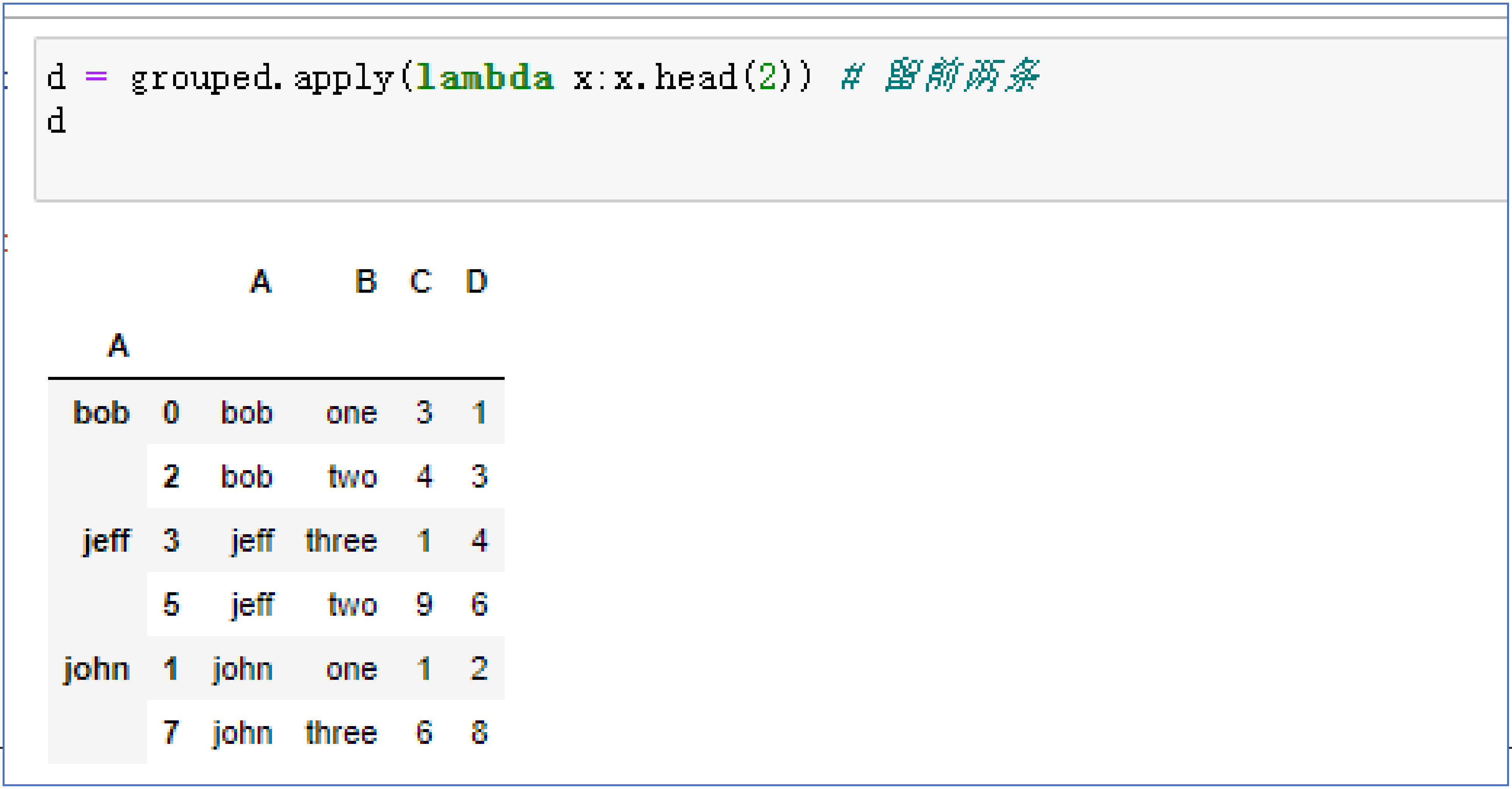
```
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, a to j
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   name      10 non-null     object
 1   score     8 non-null      float64
 2   attempts  10 non-null     int64
 3   qualify   10 non-null     object
dtypes: float64(1), int64(1), object(2)
memory usage: 400.0+ bytes
```



```
# df的统计信息 describe
print(df.describe())
```

```
           score   attempts
count   8.000000  10.000000
mean   13.562500   1.900000
std     4.693746   0.875595
min     8.000000   1.000000
25%     9.000000   1.000000
50%    13.500000   2.000000
75%    17.125000   2.750000
max    20.000000   3.000000
```

Pandas apply方法用于分组数据



```
d = grouped.apply(lambda x:x.head(2)) # 留前两条
d
```

		A	B	C	D
A					
bob	0	bob	one	3	1
	2	bob	two	4	3
jeff	3	jeff	three	1	4
	5	jeff	two	9	6
john	1	john	one	1	2
	7	john	three	6	8

表连接：合并-关联视图
Combining Datasets: Merge and Join

- **Combining**
- **Merge-Join**

Index计算 - 类ordered set

```
indA = pd.Index([1, 3, 5, 7, 9])
indB = pd.Index([2, 3, 5, 7, 11])
```

```
indA & indB  # intersection
```

```
Int64Index([3, 5, 7], dtype='int64')
```

```
indA | indB  # union
```

```
Int64Index([1, 2, 3, 5, 7, 9, 11], dtype='int64')
```

```
indA ^ indB  # symmetric difference
```

```
Int64Index([1, 2, 9, 11], dtype='int64')
```

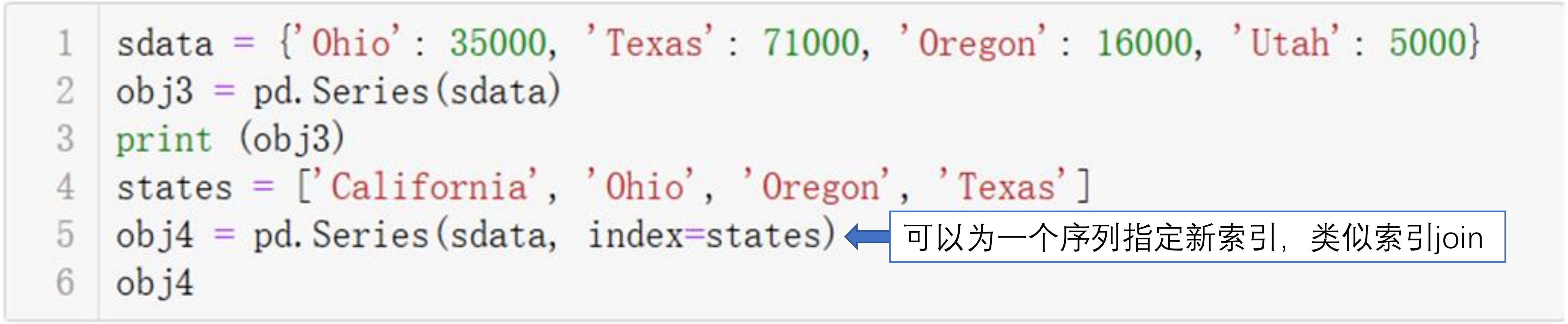
```python
# Series最重要的一个功能是，它会根据运算的索引标签自动对齐数据
# 关于数据对齐功能如果你使用过数据库，可以认为是类似join的操作
obj3+obj4
```

```
California         NaN
Ohio           70000.0
Oregon         32000.0
Texas         142000.0
Utah               NaN
dtype: float64
```

```python
obj3 - obj4
```

```
California    NaN
Ohio          0.0
Oregon        0.0
Texas         0.0
Utah          NaN
dtype: float64
```

对应元素element wise运算，非对映元素NaN



```python
sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
obj3 = pd.Series(sdata)
print (obj3)
states = ['California', 'Ohio', 'Oregon', 'Texas']
obj4 = pd.Series(sdata, index=states)
obj4
```

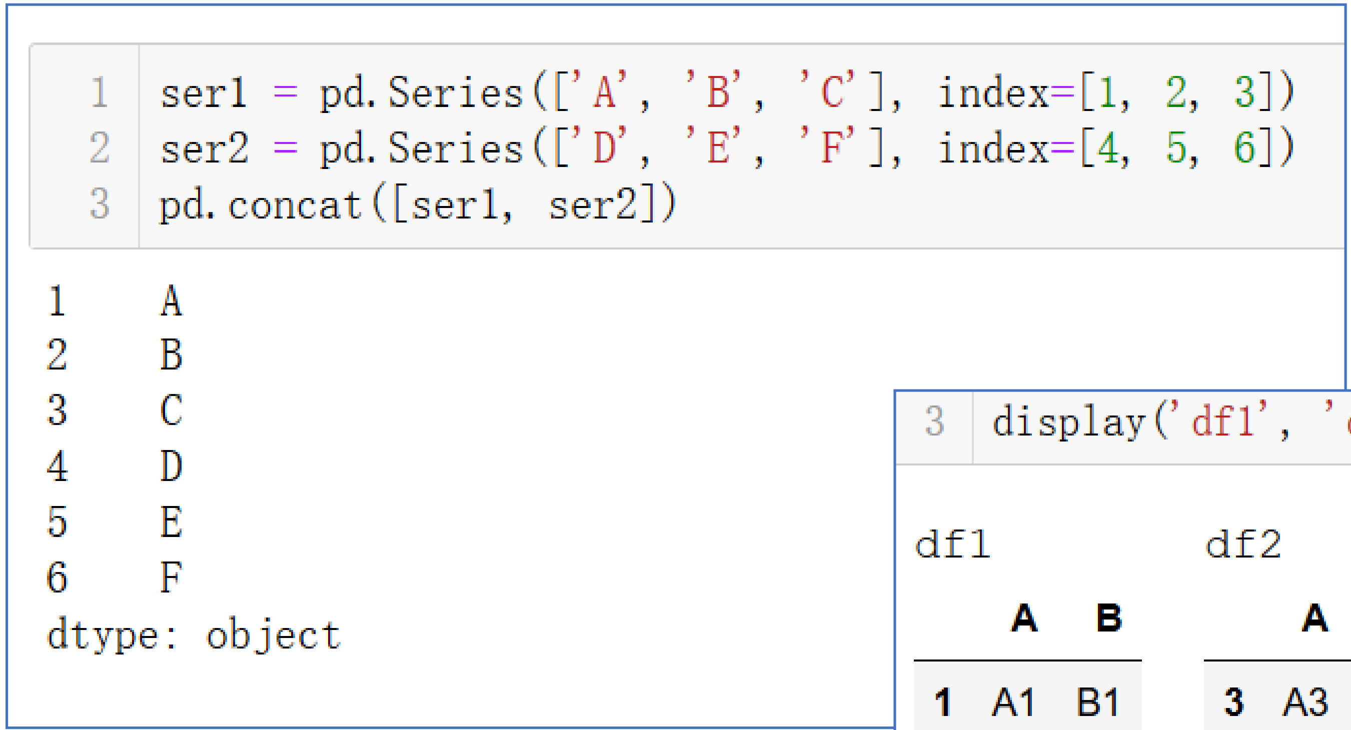
可以为一个序列指定新索引，类似索引join

```
Ohio      35000
Texas     71000
Oregon    16000
Utah       5000
dtype: int64
```

```
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

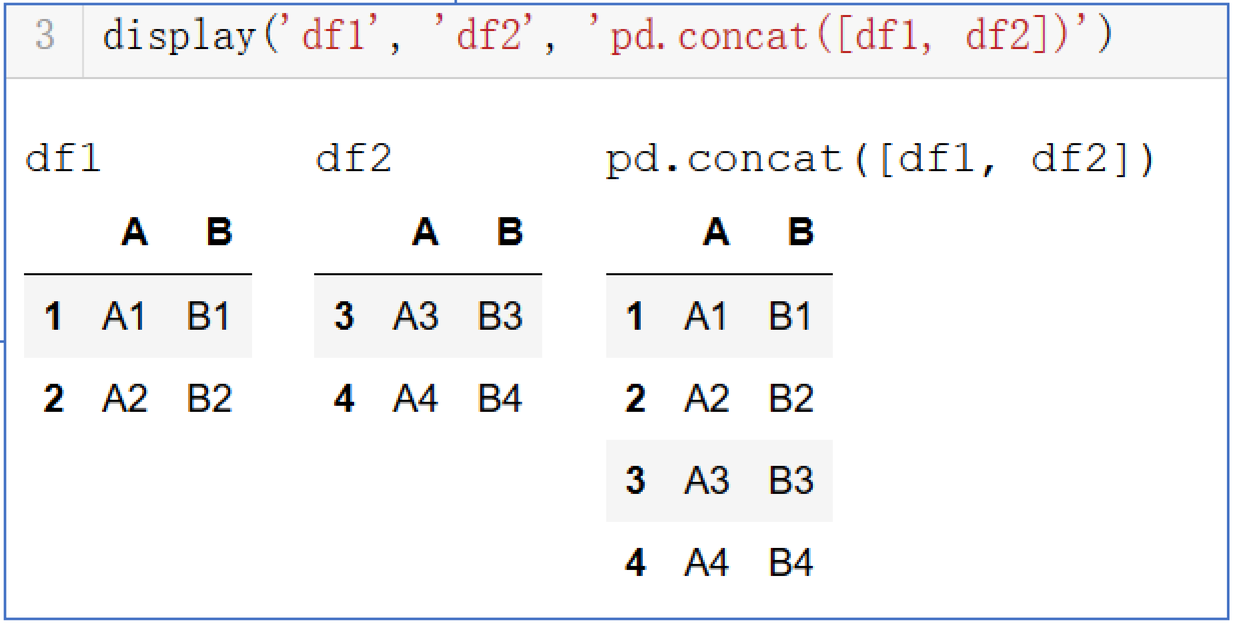
由于"California"所对应的sdata值找不到，所以其结果就为NaN（即“非数字”（not a number），在pandas中，它用于表示缺失或NA值）。因为‘Utah’不在states中，它被从结果中除去。

Combining Datasets: Concat and Append



```python
ser1 = pd.Series(['A', 'B', 'C'], index=[1, 2, 3])
ser2 = pd.Series(['D', 'E', 'F'], index=[4, 5, 6])
pd.concat([ser1, ser2])
```

```
1    A
2    B
3    C
4    D
5    E
6    F
dtype: object
```



```python
display('df1', 'df2', 'pd.concat([df1, df2])')
```

df1

	A	B
1	A1	B1
2	A2	B2

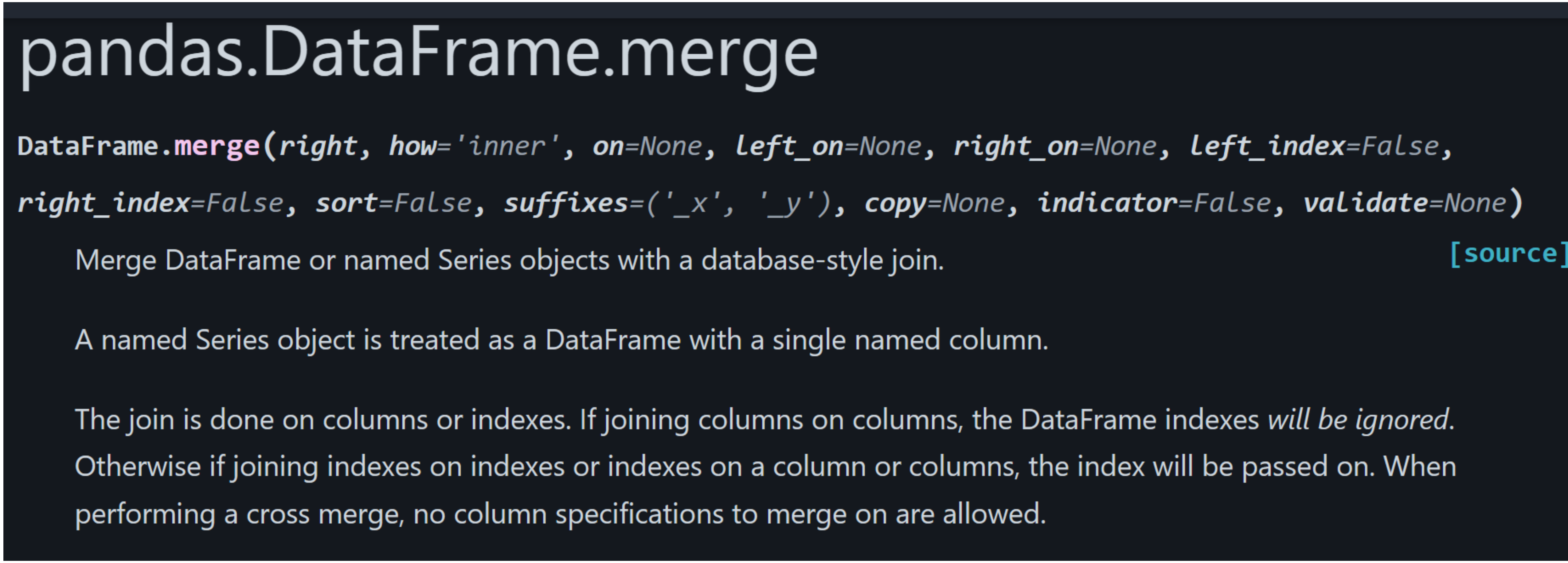
df2

	A	B
3	A3	B3
4	A4	B4

pd.concat([df1, df2])

	A	B
1	A1	B1
2	A2	B2
3	A3	B3
4	A4	B4

Merge函数：可以在非索引字段间进行



pandas.DataFrame.merge

```
DataFrame.merge(right, how='inner', on=None, left_on=None, right_on=None, left_index=False,
right_index=False, sort=False, suffixes=('_x', '_y'), copy=None, indicator=False, validate=None)
```

[source

Merge DataFrame or named Series objects with a database-style join.

A named Series object is treated as a DataFrame with a single named column.

The join is done on columns or indexes. If joining columns on columns, the DataFrame indexes *will be ignored*. Otherwise if joining indexes on indexes or indexes on a column or columns, the index will be passed on. When performing a cross merge, no column specifications to merge on are allowed.

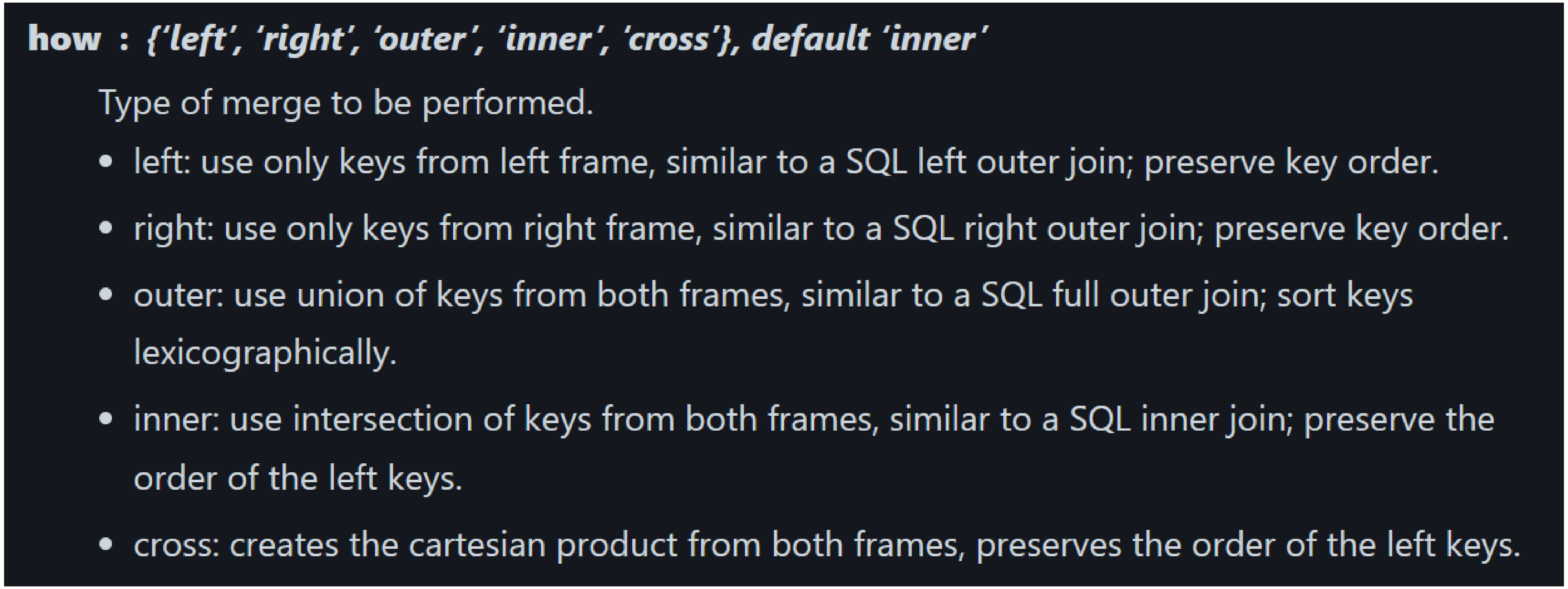
表连接的多种形式

同属性值之间:

- *one-to-one*
- *many-to-one*
- *many-to-many*

表连接的多种形式

- 属性值之间:



how : *{'left', 'right', 'outer', 'inner', 'cross'}, default 'inner'*

Type of merge to be performed.

- left: use only keys from left frame, similar to a SQL left outer join; preserve key order.
- right: use only keys from right frame, similar to a SQL right outer join; preserve key order.
- outer: use union of keys from both frames, similar to a SQL full outer join; sort keys lexicographically.
- inner: use intersection of keys from both frames, similar to a SQL inner join; preserve the order of the left keys.
- cross: creates the cartesian product from both frames, preserves the order of the left keys.

One-to-one joins:

```python
df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})
df2 = pd.DataFrame({'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
                    'hire_date': [2004, 2008, 2012, 2014]})
display('df1', 'df2')
```

df1

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

df2

	employee	hire_date
0	Lisa	2004
1	Bob	2008
2	Jake	2012
3	Sue	2014

```python
df3 = pd.merge(df1, df2)
df3
```

两个表有一个同名字段

自动按公共字段为轴进行合并

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

Many-to-one joins 有重键值与对应的唯一键值进行join，单值对应的记录展开为多个:

```python
df4 = pd.DataFrame({'group': ['Accounting', 'Engineering', 'HR'],
                    'supervisor': ['Carly', 'Guido', 'Steve']})
display('df3', 'df4', 'pd.merge(df3, df4)')
```

df3

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

df4

	group	supervisor
0	Accounting	Carly
1	Engineering	Guido
2	HR	Steve

pd.merge(df3, df4)

	employee	group	hire_date	supervisor
0	Bob	Accounting	2008	Carly
1	Jake	Engineering	2012	Guido
2	Lisa	Engineering	2004	Guido
3	Sue	HR	2014	Steve

Many-to-many joins：
多对多关联操作可以认为是一对多操作的复用

```
df5 = pd.DataFrame({'group': ['Accounting', 'Accounting',
                              'Engineering', 'Engineering', 'HR', 'HR'],
                    'skills': ['math', 'spreadsheets', 'coding', 'linux',
                               'spreadsheets', 'organization']})
display('df1', 'df5', "pd.merge(df1, df5)")
```

df1

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

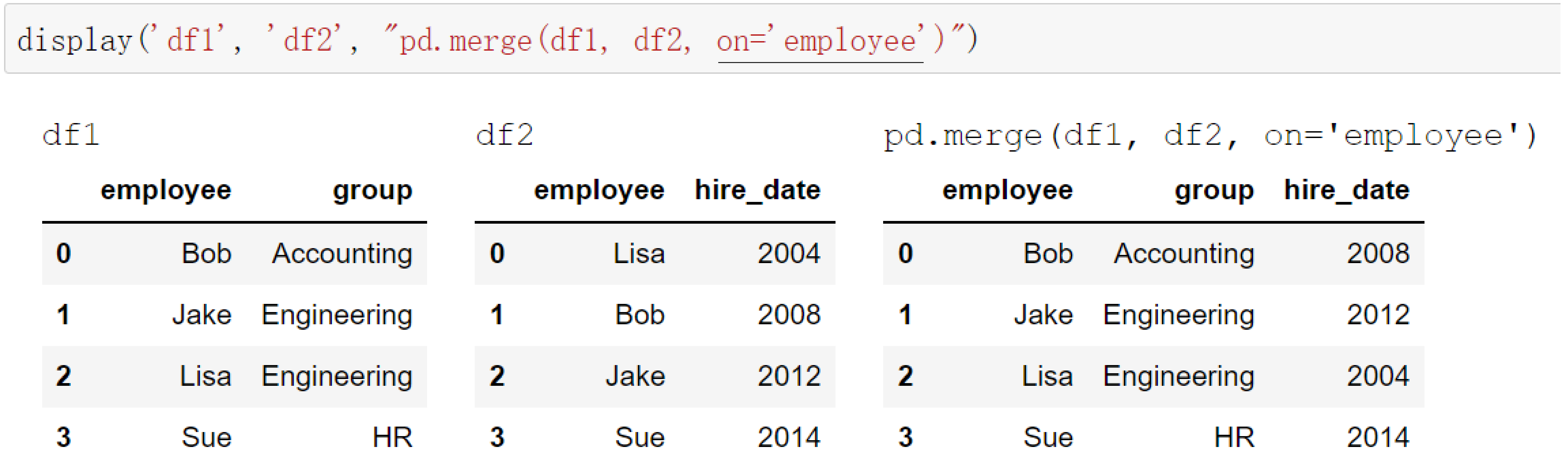
df5

	group	skills
0	Accounting	math
1	Accounting	spreadsheets
2	Engineering	coding
3	Engineering	linux
4	HR	spreadsheets
5	HR	organization

pd.merge(df1, df5)

	employee	group	skills
0	Bob	Accounting	math
1	Bob	Accounting	spreadsheets
2	Jake	Engineering	coding
3	Jake	Engineering	linux
4	Lisa	Engineering	coding
5	Lisa	Engineering	linux
6	Sue	HR	spreadsheets
7	Sue	HR	organization

指定特定字段进行关联 merge-on



```
display(’df1’, ’df2’, ”pd.merge(df1, df2, on=’employee’)”)
```

df1

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

df2

	employee	hire_date
0	Lisa	2004
1	Bob	2008
2	Jake	2012
3	Sue	2014

pd.merge(df1, df2, on='employee')

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

left_on,right_on可以指定不同名字段关联

```
df3 = pd.DataFrame({'name': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'salary': [70000, 80000, 120000, 90000]})
display('df1', 'df3', 'pd.merge(df1, df3, left_on="employee", right_on="name")')
```

df1

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

df3

	name	salary
0	Bob	70000
1	Jake	80000
2	Lisa	120000
3	Sue	90000

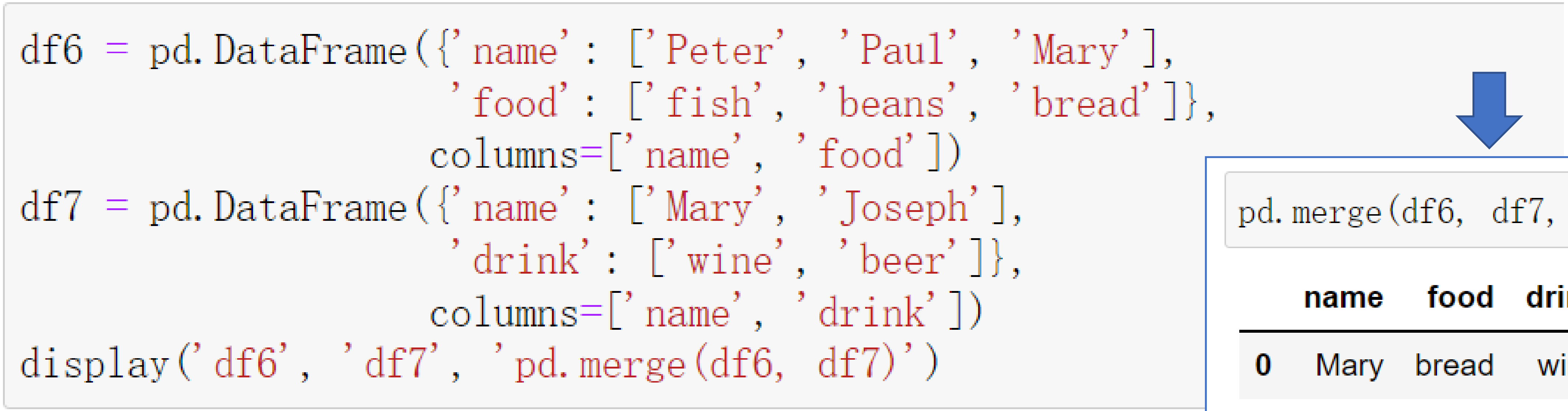
pd.merge(df1, df3, left_on="employee", right_on="name")

	employee	group	name	salary
0	Bob	Accounting	Bob	70000
1	Jake	Engineering	Jake	80000
2	Lisa	Engineering	Lisa	120000
3	Sue	HR	Sue	90000

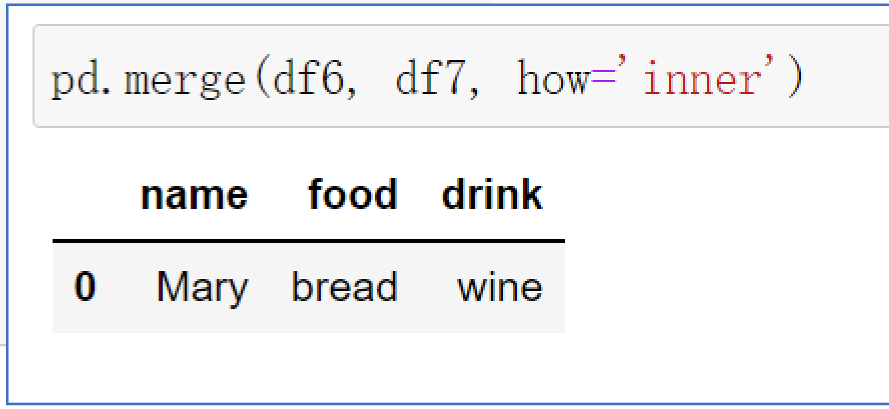
The result has a redundant column that we can drop if desired–for example, by using the `drop()` method of `DataFrame`s:

```
pd.merge(df1, df3, left_on="employee", right_on="name").drop('name', axis=1)
```

Inner-join 求索引交集

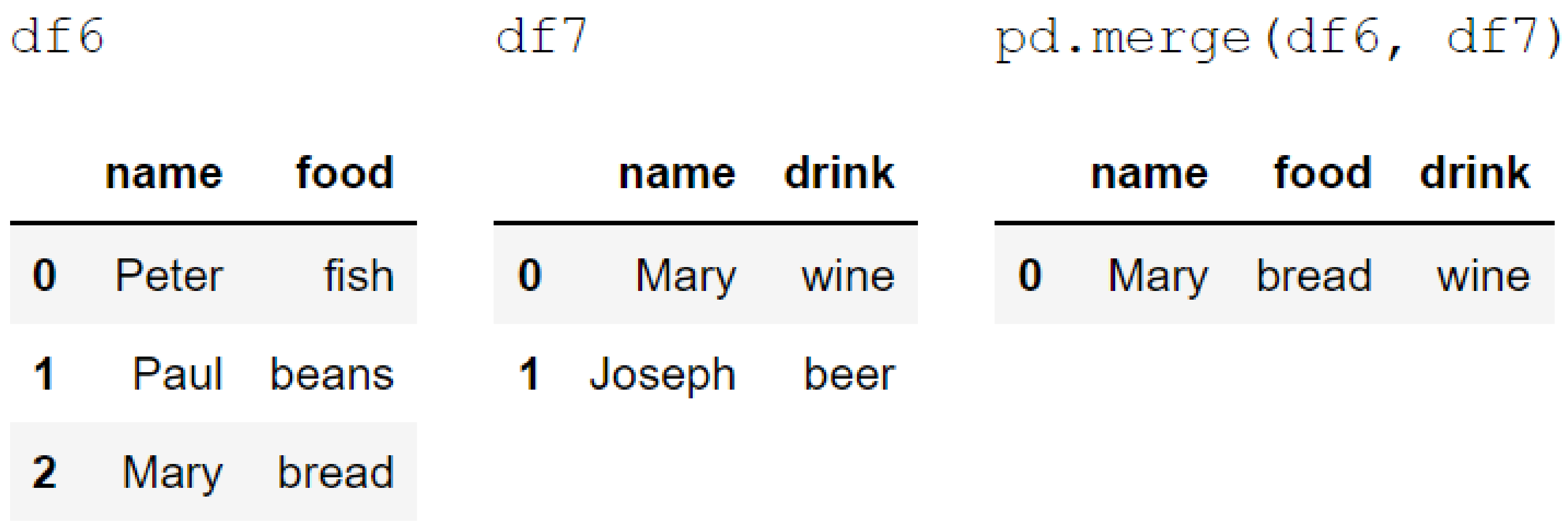


```
df6 = pd.DataFrame({'name': ['Peter', 'Paul', 'Mary'],
                    'food': ['fish', 'beans', 'bread']},
                   columns=['name', 'food'])
df7 = pd.DataFrame({'name': ['Mary', 'Joseph'],
                    'drink': ['wine', 'beer']},
                   columns=['name', 'drink'])
display('df6', 'df7', 'pd.merge(df6, df7)')
```



```
pd.merge(df6, df7, how='inner')
```

	name	food	drink
0	Mary	bread	wine



df6

	name	food
0	Peter	fish
1	Paul	beans
2	Mary	bread

df7

	name	drink
0	Mary	wine
1	Joseph	beer

pd.merge(df6, df7)

	name	food	drink
0	Mary	bread	wine

outer-join是关联字段项的并集（left join， right join…）

```
display('df6', 'df7', "pd.merge(df6, df7, how='outer')")
```

df6

	name	food
0	Peter	fish
1	Paul	beans
2	Mary	bread

df7

	name	drink
0	Mary	wine
1	Joseph	beer

```
pd.merge(df6, df7, how='outer')
```

	name	food	drink
0	Peter	fish	NaN
1	Paul	beans	NaN
2	Mary	bread	wine
3	Joseph	NaN	beer

文本信息处理基础

- 中文分词与词频统计
- 向量空间模型与文档特征
- 话题模型

中文文本分词与词频统计

- Jieba分词
- 新词与短语发现

例子：分类关键词发现

```
import pandas as pd
fruit_all=pd.read_csv('fruit.csv')
fruit_all.head(6)
```

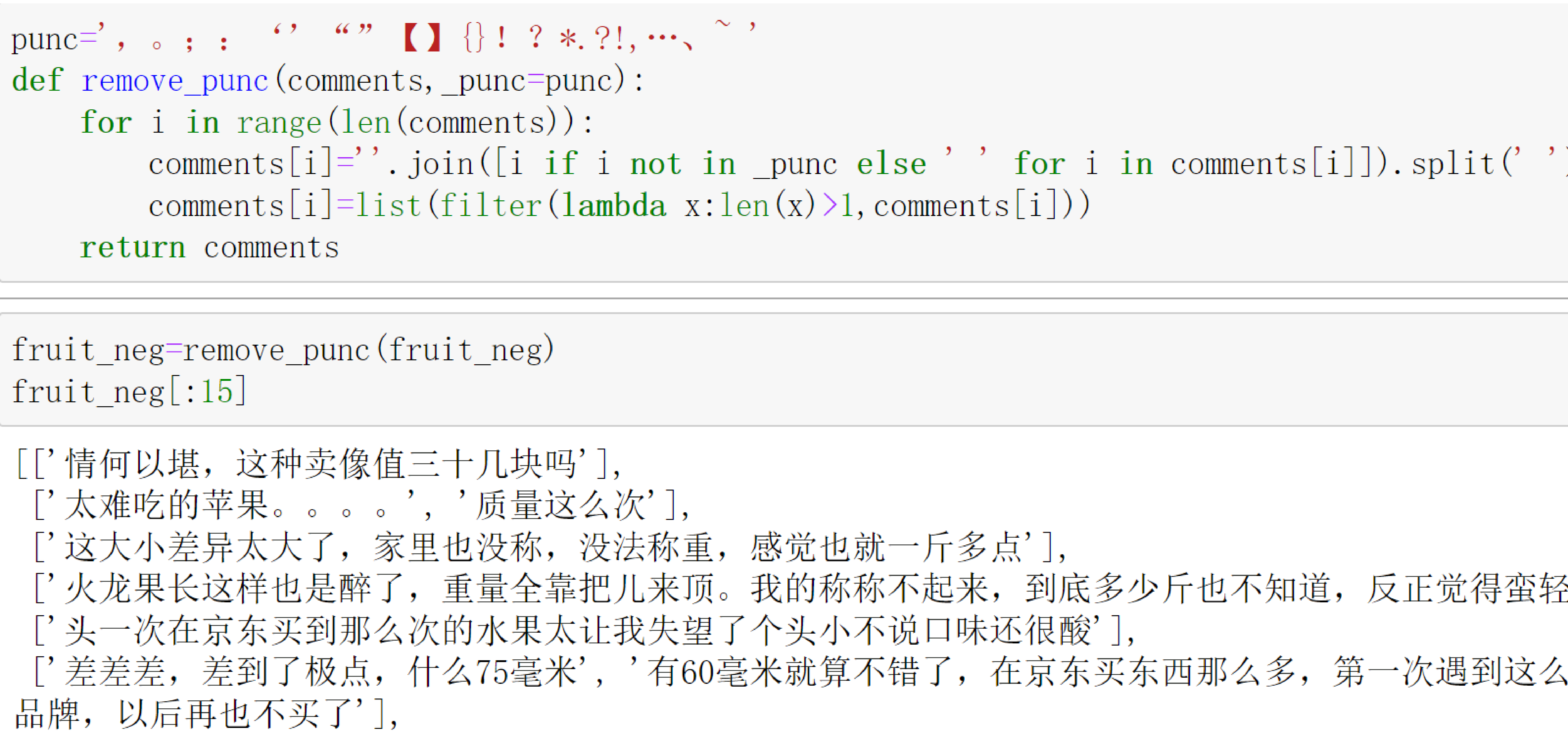
	cat	label	review
0	水果	1	闻着很重苹果味，很香，吃起来不是很脆，但苹果是好吃的。
1	水果	0	情何以堪，这种卖像值三十几块吗?
2	水果	0	太难吃的苹果。。。。**啊 质量这么次
3	水果	1	味道还不错，但个头真的是太小了，快递小哥辛苦啦，送的很快
4	水果	1	甜，口感好，京东送货快。支持京东的自营，
5	水果	0	这大小差异太大了，家里也没称，没法称重，感觉也就一斤多点

分别提取褒贬类数据:

```
fruit_neg=fruit_all[fruit_all['label']==0].review.to_list()
fruit_pos=fruit_all[fruit_all['label']==1].review.to_list()
fruit_pos[:5]
```

```
['闻着很重苹果味，很香，吃起来不是很脆，但苹果是好吃的。',
 '味道还不错，但个头真的是太小了，快递小哥辛苦啦，送的很快',
 '甜，口感好，京东送货快。支持京东的自营，',
 '甜甜脆脆的，很好吃，店里一收到就吃了5个，都觉得很不错，物廉价美值得购买哈，春
 '第一次在京东上买水果，特别感谢快递帅哥，我的地址只写到楼层，没想到快递员来之
好。苹果没有损坏，每个都有两层泡沫包裹。还没吃，看纹路，应该口感不会差，很新鲜，
的。']
```

去除标点（根据流程可选）：



```
punc='，。；：‘’“”【】{}！？*.?!,…、～'
def remove_punc(comments,_punc=punc):
    for i in range(len(comments)):
        comments[i]=''.join([i if i not in _punc else ' ' for i in comments[i]]).split('
        comments[i]=list(filter(lambda x:len(x)>1,comments[i]))
    return comments
```

```
fruit_neg=remove_punc(fruit_neg)
fruit_neg[:15]
```

```
[['情何以堪，这种卖像值三十几块吗'],
 ['太难吃的苹果。。。。', '质量这么次'],
 ['这大小差异太大了，家里也没称，没法称重，感觉也就一斤多点'],
 ['火龙果长这样也是醉了，重量全靠把儿来顶。我的称称不起来，到底多少斤也不知道，反正觉得蛮轻
 ['头一次在京东买到那么次的水果太让我失望了个头小不说口味还很酸'],
 ['差差差，差到了极点，什么75毫米', '有60毫米就算不错了，在京东买东西那么多，第一次遇到这么
品牌，以后再也不买了'],
```

分类高频词提取与词云显示：

很好吃
很甜
很新鲜
很不错
还不错
比较
很快
喜欢
快递小哥
下单
没有坏
方便
非常好
脆甜
很脆
好评
满意
吃完
个头不大
包装很好
推荐
味道不错
果子
发货
多次
放心
速度
送到
水分
送货快
实惠
可口
多次购买
非常满意
产品
快递员
划算
苹果不错
值得
辛苦
有点小
这个苹果
第二天
给力
态度
喜欢吃
还好
活动
谢谢
很大
挺新鲜
上午
下午

高频短语合并与更新jieba用户词典

```
def word_available(word_pos):    # 去停用词
    #remove words with pos 'u' or 'c'
    if 'u' in word_pos:
        return False
    if 'c' in word_pos:
        return False
    return True
def neighbor_words(article,theta=5):  # 统计近邻词 freq >= 5
    #catenate neighboring words and count the frequency of catenated words
    count_neighbor_words={}
    for i in range(len(article)):
        for ii in range(len(article[i])):
            sentence=[tuple(t) for t in posseg.cut(article[i][ii])] #分词
            for j in range(len(sentence)-1):
                if word_available(sentence[j][1]) and word_available(sentence[j+1][1]): #近邻词非pos 'u' or 'c'
                    cat_words = sentence[j][0]+sentence[j+1][0]                       # 合并为短语词
                    count_neighbor_words[cat_words]=count_neighbor_words.get(cat_words,0) + 1
    count_neighbor_words=list(filter(lambda x:x[1]>theta,count_neighbor_words.items())) # 邻接超过5次
    return sorted(count_neighbor_words,key=lambda x:x[1],reverse=True)

def add_cat_words(article,theta=5,epochs=3): # 将短语词加入jieba用户词典
    #add catenated words to jieba's dictionary
    for i in range(epochs):                                      # 支持多轮合并
        candidate_neighbor_words = neighbor_words(article,theta)  # 获取短语词表
        #print(candidate_neighbor_words)
        if len(candidate_neighbor_words)==0:
            break
        for word,fre in candidate_neighbor_words:
            added_words.append((word,fre))
            jieba.add_word(word,fre)
```

```
with open('stopwords.txt','r',encoding='utf-8') as f:
    stopwords=[line.strip() for line in f]
```

```
from collections import Counter
def count_word(comments):
    sentences=[]
    for comment in comments:
        for sentence in comment:
            sentence_cut=list(jieba.cut(sentence))
            sentences+=sentence_cut
    word_fre=dict(Counter(sentences))
    word_fre_sorted=list(sorted(word_fre.items(),key=lambda x:x[1],reverse=True))
    word_fre_sorted=list(filter(lambda x:x[0] not in stopwords,word_fre_sorted))
    return word_fre_sorted
```

```
pos_words=count_word(fruit_pos)
pos_words[:20]
```

```
[('苹果', 268),
 ('好', 246),
 ('很', 204),
 ('吃', 197),
 ('买', 189),
 ('还', 155),
 ('不错', 140),
 ('都', 136),
 ('好吃', 120),
neg_keywords=neg_words_set-pos_words_set
print(neg_keywords)

neg_dict=dict(list(filter(lambda x:x[0] in neg_keywords,neg_words)))
create_wordcloud(neg_dict,'neg_wordcloud.png')  #调用词云
```

```
{'难吃', '4个', '大果', '不买', '大小不一', '现在', '全部', '不脆', '大家', '是烂', '完全', '不
'很小', '很差', '一半', '售后', '是不是', '2个', '就算', '不满意', '不会', '一次购物', '真是',
'看到', '发霉', '之前', '不能', '苹果都', '一股', '不行', '一共', '不甜', '问题', '收到货', '再
'里面', '这家', '出来', '第一次买', '图片', '卖家', '失望', '差评', '差劲', '根本', '这么小', '
箱', '外面', '最差', '退货', '不想', '上当', '这苹果', '大家不要买', '3个', '箱子', '不新鲜', '
'估计', '竟然', '水果店', '超市买', '一斤', '客服', '真心', '这次买', '上面', '不好吃'}
```

失望
太差
三个
不甜
里面
差评
客服
不新鲜
不好
不满意
很失望
很小
一箱
不要
根本
真是
超市买
外面
退货
最差
一次购物
3个
无语
这种
完全
估计
箱子
2个
4个
看到
是不是

给定文档集，统计文档的TF*IDF向量

- 按关键词集合实现文档检索
- 利用文档的词向量表达计算文档相似度或实现话题聚类?
- 利用词的文档分布计算词义相似度或实现词义聚类?

词袋子方案下的文档词向量 Incidence Vector

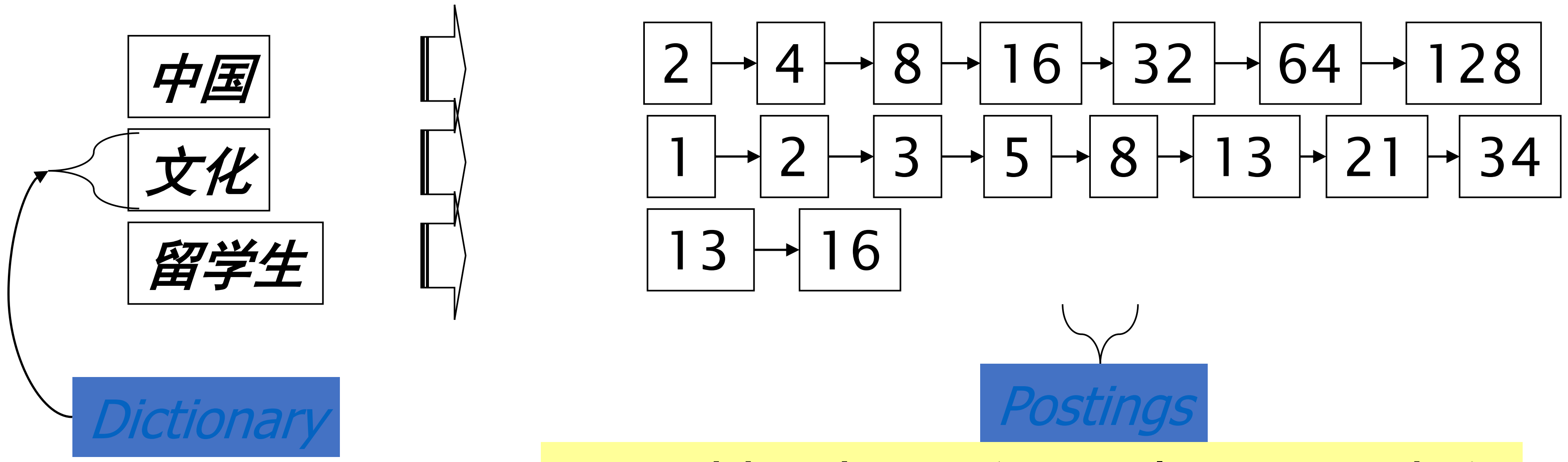
- term-document 关联矩阵
- 每个term对应一个0/1或词频 向量, incidence vector

	D_1	D_2	D_3	D_4	D_5	D_6	…
中国	1	0	1	1	1	0	
文化	1	1	0	0	1	0	
日本	0	1	1	0	1	1	
留学生	0	1	1	1	0	0	
教育	1	0	0	1	0	0	
北京	1	0	0	0	1	1	
…							

1 or TF if page contains word, otherwise 0

倒排索引 Inverted index

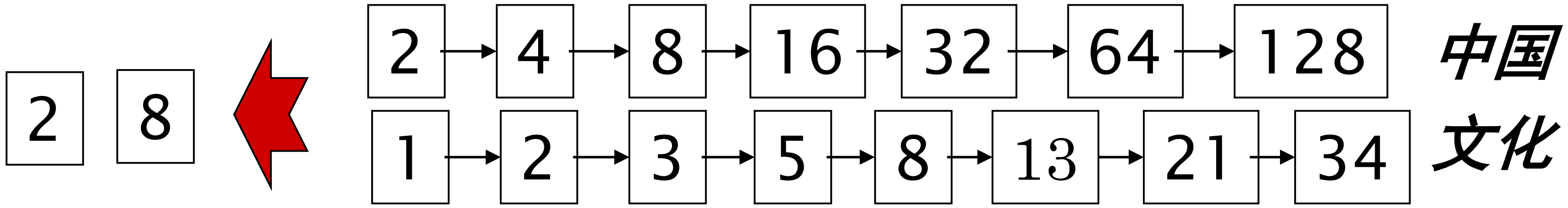
- 对每个 term *T*: 保存包含T的文档(编号)列表



Sorted by docID (more later on why).

Boolean Query processing

- 查询: ***中国*** *AND* ***文化***
  - 查找Dictionary，定位***中国***;
    - 读取对应的postings.
  - 查找Dictionary，定位***文化***;
    - 读取对应的postings.
  - “Merge” 合并(AND)两个postings:



ranking: density-based

- 按query，给文档打分scoring，根据score排序
- Idea
  - 如果一个文档 talks about a topic *more*, then it is a *better* match

    ➔如果包含很多次query term的出现，文档是relevant(相关的)
  - ➔ **term weighting?**

tf ∗ idf term weights

- tf x idf 权值计算公式:
  - term frequency (*tf*)
    - or *wf*, some measure of term density in a doc
  - inverse document frequency (*idf*)
    - 表达term的重要度(稀有度)
    - 原始值$idf_t = 1/df_t$
      - 通常会作平滑

$$idf_t = \log\left(\frac{N}{df_t}\right)$$

- 实际文档词向量中计算其tf∗idf权重来表达与query的相关性:

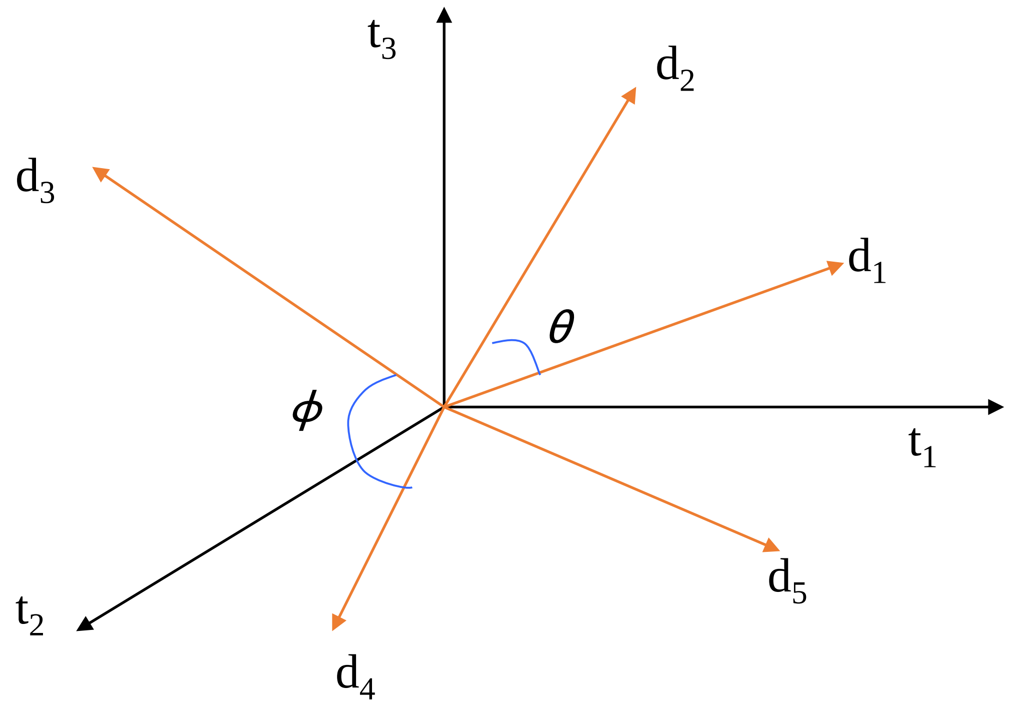
$$w_{t,d} = tf_{t,d} \times \log(N/df_t)$$

Documents to vectors

	D_1	D_2	D_3	D_4	D_5	D_6	…
中国	4.1	0.0	3.7	5.9	3.1	0.0	
文化	4.5	4.5	0	0	11.6	0	
日本	0	3.5	2.9	0	2.1	3.9	
留学生	0	3.1	5.1	12.8	0	0	
教育	2.9	0	0	2.2	0	0	
北京	7.1	0	0	0	4.4	3.8	
…							

- 每一个文档 *j* 能够被看作一个向量，每个term 是一个维度，取值为tf.idf
- So we have a vector space
  - terms are axes
  - docs live in this space
  - 高维空间：即使作stemming, may have 20,000+ dimensions

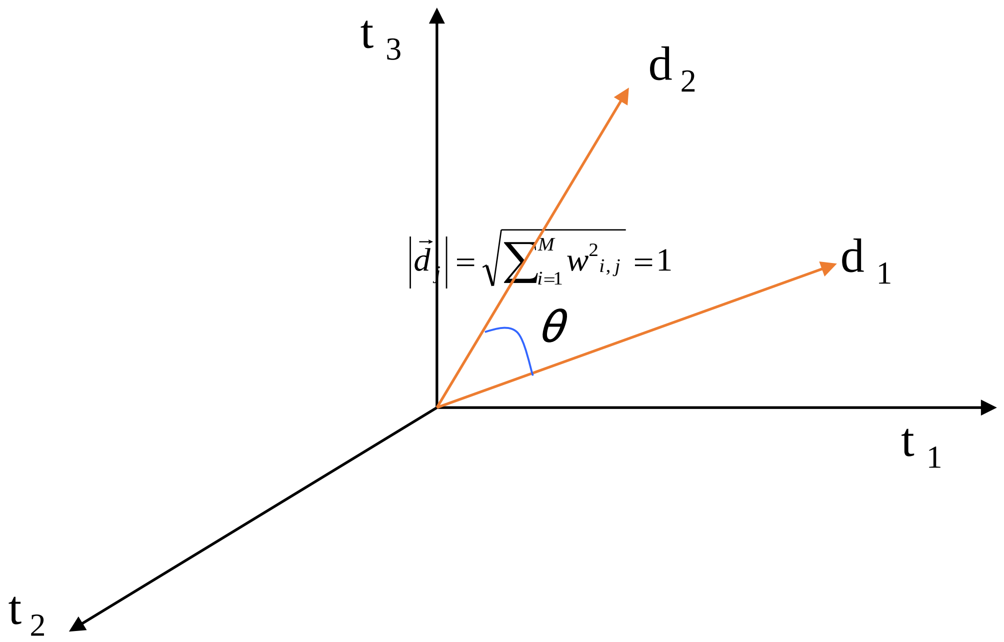
文档词向量空间模型（简称向量空间模型）



一个符合直觉的假定：在vector space中“close together” 的文档会talk about the same things.

用例：Query-by-example，Free Text query as vector

Cosine similarity



t_3
d_2
d_1
$\left|\vec{d}_j\right| = \sqrt{\sum_{i=1}^{M} w_{i,j}^2} = 1$
θ
t_1
t_2

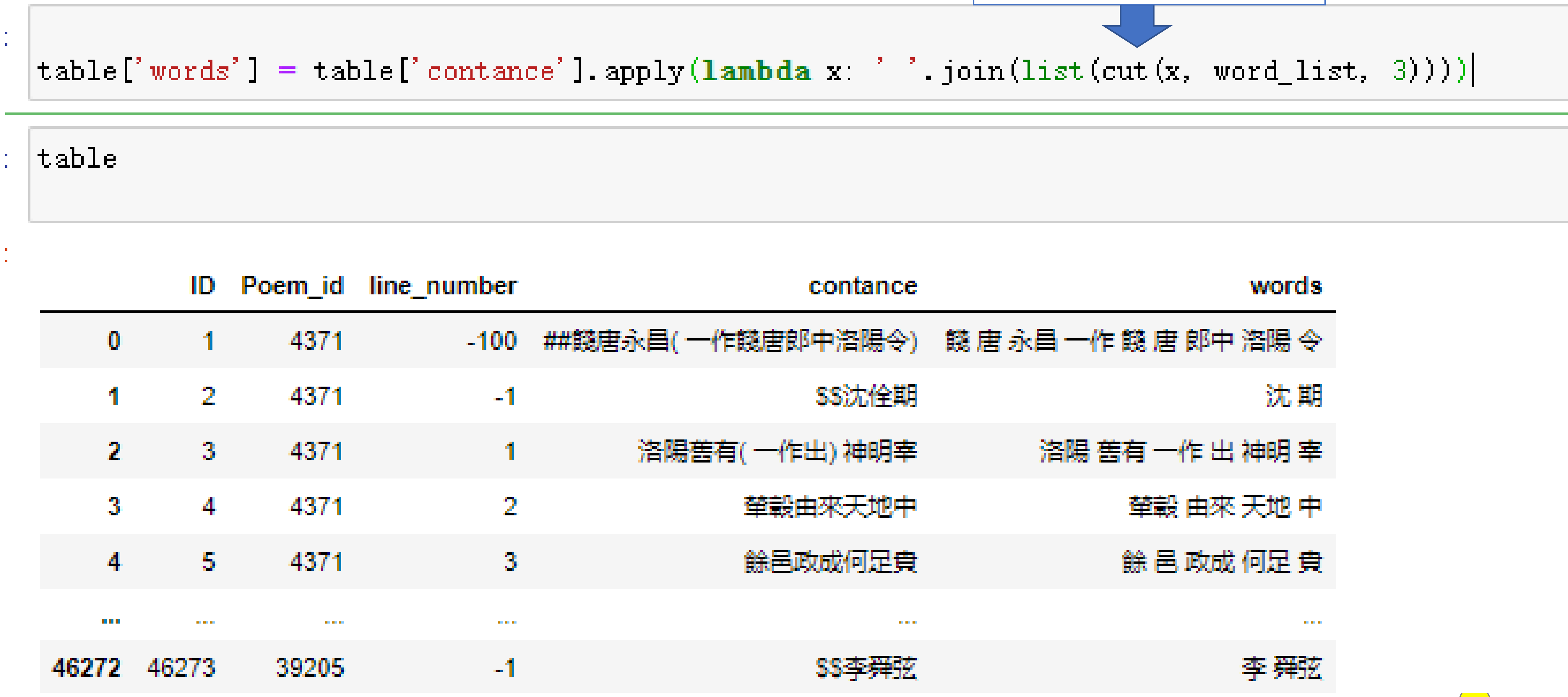
- 向量d_1和d_2的 “closeness”可以用它们之间的夹角大小来度量
- 具体的，可用cosine of the angle *x*来*计算向量相似度*.
- 向量按长度归一化Normalization

$$sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right| \left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{M} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{M} w_{i,j}^2} \sqrt{\sum_{i=1}^{M} w_{i,k}^2}}$$

计算文档的TF*IDF向量

- 统计文档总数
- 统计每篇文档的TF向量
- 统计每个词出现的文档数
- 计算每个词的IDF
- 在TF向量中乘上对应词的IDF（可以按词表广播？）

自定义最大向前匹配函数

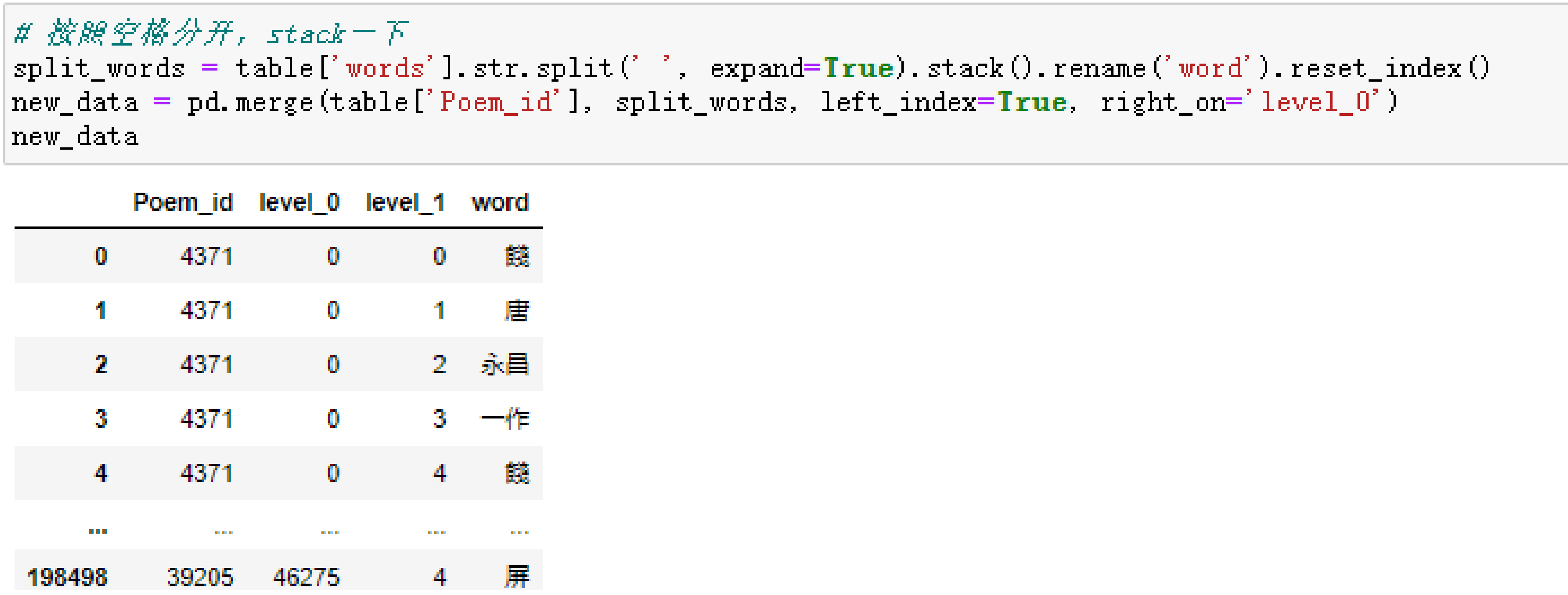


```
table['words'] = table['contance'].apply(lambda x: ' '.join(list(cut(x, word_list, 3))))
```

```
table
```

	ID	Poem_id	line_number	contance	words
0	1	4371	-100	##餞唐永昌(一作餞唐郎中洛陽令)	餞 唐 永昌 一作 餞 唐 郎中 洛陽 令
1	2	4371	-1	SS沈佺期	沈 期
2	3	4371	1	洛陽舊有(一作出)神明宰	洛陽 舊有 一作 出 神明 宰
3	4	4371	2	輦轂由來天地中	輦轂 由來 天地 中
4	5	4371	3	餘邑政成何足貴	餘 邑 政成 何足 貴
...	...	...	...	...	...
46272	46273	39205	-1	SS李舜弦	李 舜弦

1.2 统计每个词的TF-IDF值



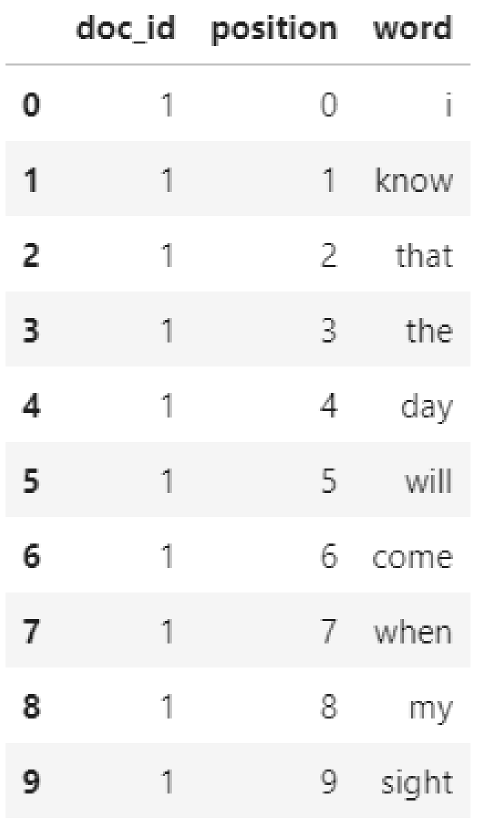
```
# 按照空格分开，stack一下
split_words = table['words'].str.split(' ', expand=True).stack().rename('word').reset_index()
new_data = pd.merge(table['Poem_id'], split_words, left_index=True, right_on='level_0')
new_data
```

	Poem_id	level_0	level_1	word
0	4371	0	0	饒
1	4371	0	1	唐
2	4371	0	2	永昌
3	4371	0	3	一作
4	4371	0	4	饒
...	...	...	...	...
198498	39205	46275	4	屏

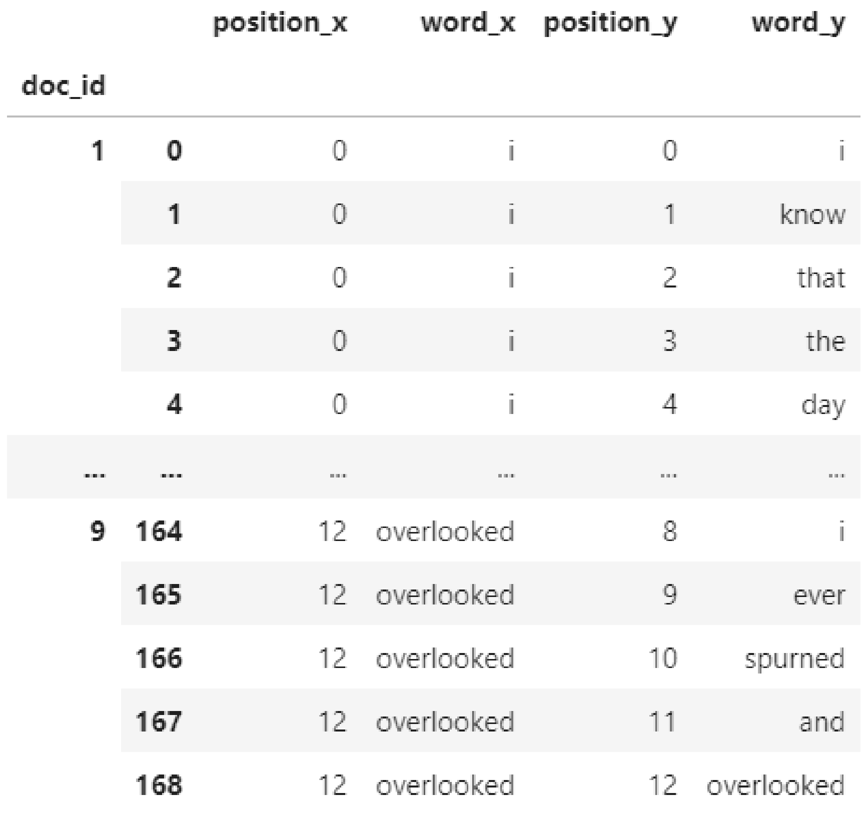
https://www.oreilly.com/library/view/python-data-science/9781491912126/

1 生成“词-上下文词”的二维索引序列表

- grouby + merge(, how = ‘cross’):

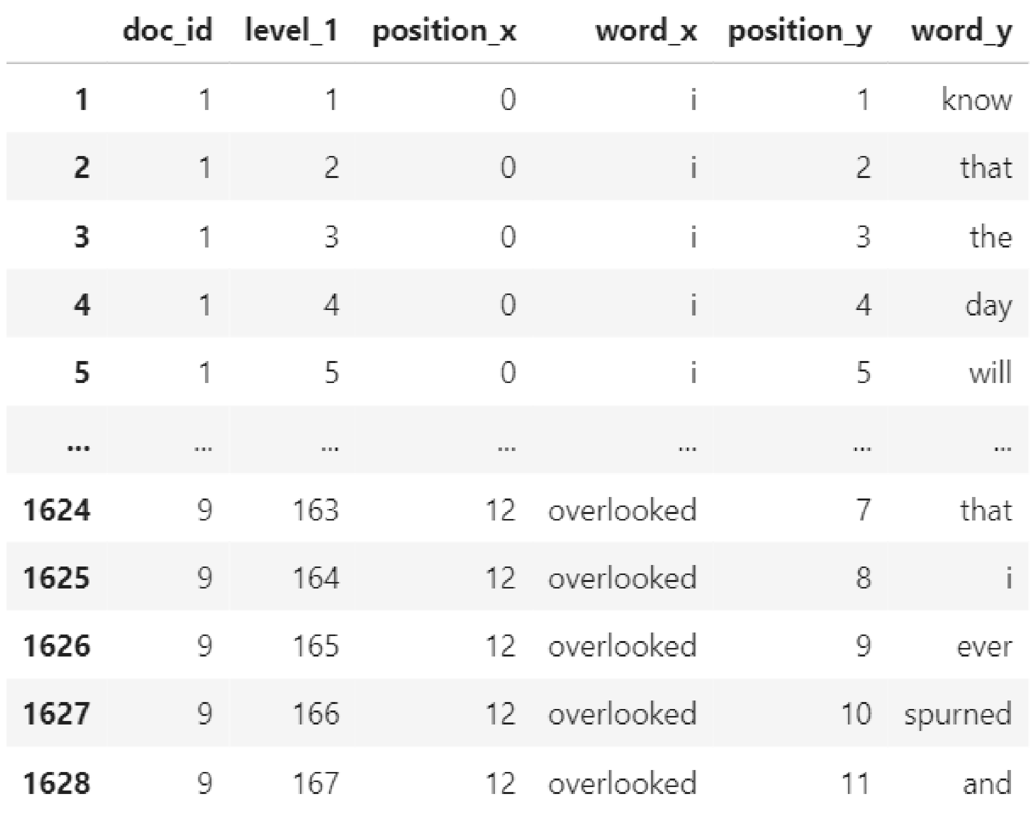


	doc_id	position	word
0	1	0	i
1	1	1	know
2	1	2	that
3	1	3	the
4	1	4	day
5	1	5	will
6	1	6	come
7	1	7	when
8	1	8	my
9	1	9	sight



		position_x	word_x	position_y	word_y
doc_id					
1	**0**	0	i	0	i
	1	0	i	1	know
	2	0	i	2	that
	3	0	i	3	the
	4	0	i	4	day
...	**...**	...	...	...	...
9	**164**	12	overlooked	8	i
	165	12	overlooked	9	ever
	166	12	overlooked	10	spurned
	167	12	overlooked	11	and
	168	12	overlooked	12	overlooked

merge



	doc_id	level_1	position_x	word_x	position_y	word_y
1	1	1	0	i	1	know
2	1	2	0	i	2	that
3	1	3	0	i	3	the
4	1	4	0	i	4	day
5	1	5	0	i	5	will
...	...	...	...	...	...	...
1624	9	163	12	overlooked	7	that
1625	9	164	12	overlooked	8	i
1626	9	165	12	overlooked	9	ever
1627	9	166	12	overlooked	10	spurned
1628	9	167	12	overlooked	11	and

reset_index()之后

主题模型（LSI-LDA）

- SVD-LSI
- 非负矩阵分解（NMF）与主题合成思想
- 多项分布迪利克雷过程与文档-主题生成模型

SVD到LSI（隐含语义索引）

3.LSI

- 利用矩阵奇异值分解SVD，对单词-文本矩阵进行分解得到主题向量空间的话题-文本矩阵

$$X \approx U_k \underset{\Uparrow}{\Sigma_k} V_k^{\mathrm{T}} = \begin{bmatrix} u_1 & u_2 & \cdots & u_k \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sigma_k \end{bmatrix} \begin{bmatrix} v_1^{\mathrm{T}} \\ v_2^{\mathrm{T}} \\ \vdots \\ v_k^{\mathrm{T}} \end{bmatrix}$$

  分解得到的矩阵U为话题向量空间，对角矩阵和矩阵V的乘积作为文本在话题向量空间的表示
- 非负矩阵分解 NMF

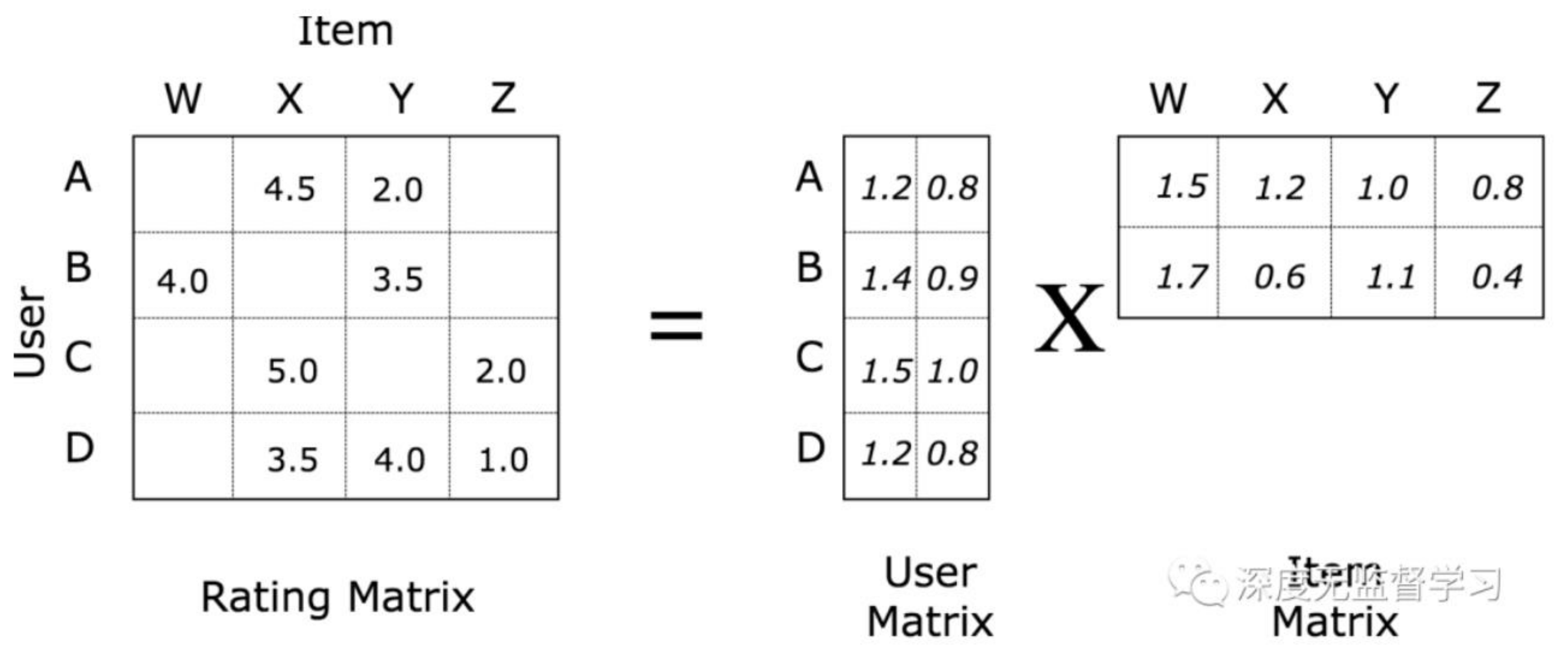
  给定一个非负矩阵$X \geq 0$，找到两个非负矩阵$W_{m\times k}, H_{k\times n}$，使得$X = WH$

```python
    with open(path,"r",encoding='utf-8') as f:
        for line in f.readlines()[:5]:
            corpus.append(json.loads(line)["sentence"])

    # 计算单词-文本矩阵
    W, words = compute_word_document_matrix(corpus)

    topics = 3      # 设定话题数
    lsa = TruncatedSVD(n_components=topics)    # 使用SVD，降到指定的维数，可选，lsa = NMF(n_components=topics) 非负矩阵分解
    U = lsa.fit_transform(W)  # 投影到新的基底

    # 每个话题选出3个关键词
    keynum = 3
    # lsa.components_是一个大小为(T,V)，每一行为主题在每个单词上的分布。我们可以通过这个矩阵得到哪些词对主题t贡献最大
    topic_keywdid = [lsa.components_[t].argsort()[:-(keynum + 1):-1] for t in range(topics)]
    print('LSI奇异值:')
    print(lsa.singular_values_)
    print('10个话题向量空间下的表示')
    print(U)  # 降维后的文档表达
    for t in range(topics):
        print(f'话题{t+1}:')
        print(f'\t 关键词 {", ".join(words[topic_keywdid[t][j]] for j in range(keynum))}')
```

```
LSI奇异值:
[1.06925492 1.         1.        ]
10个话题向量空间下的表示
[[ 4.33164668e-18 -2.78375305e-16 -1.64198744e-17]
 [-3.01429735e-15  4.47213595e-01  8.94427191e-01]
 [ 7.56077407e-01  1.03574990e-15 -3.95748223e-15]
 [ 1.10791503e-15  8.94427191e-01 -4.47213595e-01]
 [ 7.56077407e-01  1.27651404e-16  8.14005188e-15]]
话题1:
	 关键词 改革, 开放, 坚持
话题2:
	 关键词 １９８７年, 儿女, 征程
话题3:
	 关键词 小雪, 风向, 风力
```

非负矩阵分解与主题合成方案

- 例1：在推荐系统中，将用户给与不同商品的评分拆分为用户矩阵与商品条目矩阵。通过这种拆分能够发现一些用户的相似喜好，进行内容推荐。

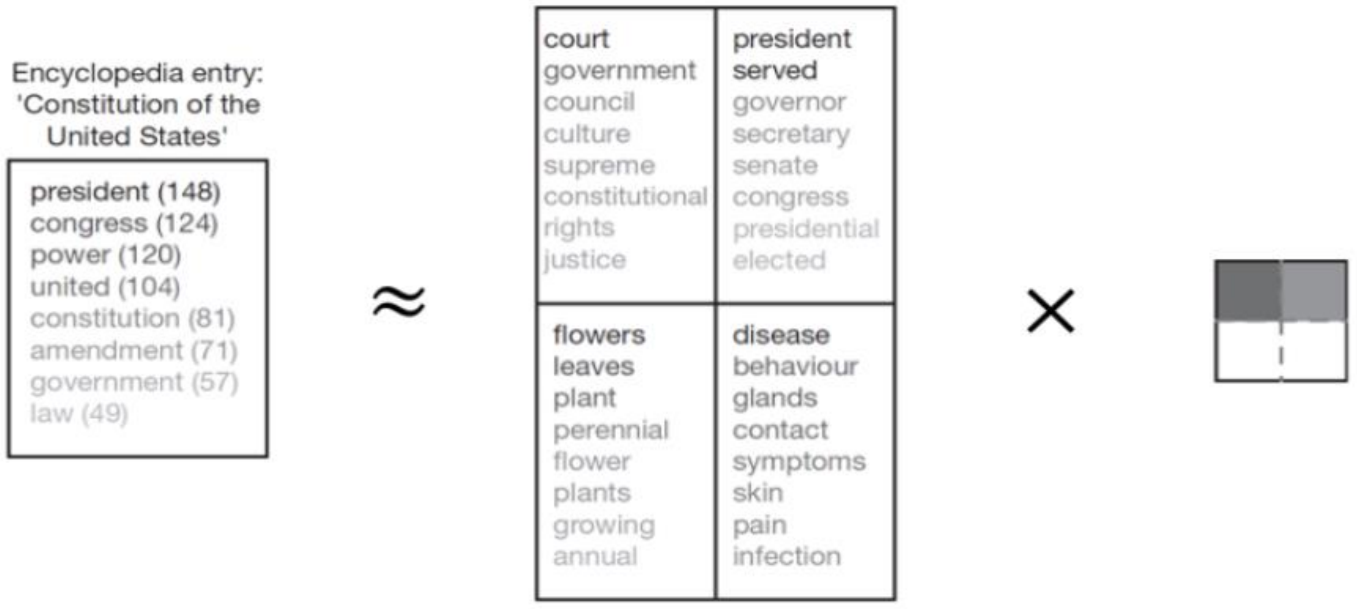


$$\begin{array}{c|cccc} & W & X & Y & Z \\ \hline A & & 4.5 & 2.0 & \\ B & 4.0 & & 3.5 & \\ C & & 5.0 & & 2.0 \\ D & & 3.5 & 4.0 & 1.0 \end{array} = \begin{array}{c|cc} A & 1.2 & 0.8 \\ B & 1.4 & 0.9 \\ C & 1.5 & 1.0 \\ D & 1.2 & 0.8 \end{array} \times \begin{array}{cccc} W & X & Y & Z \\ \hline 1.5 & 1.2 & 1.0 & 0.8 \\ 1.7 & 0.6 & 1.1 & 0.4 \end{array}$$

https://www.zhihu.com/question/34801598/answer/2255092112

非负矩阵分解与主题合成方案(续)

- 例3：文件拆分：将文件拆分为一些主题词汇与他们的比例权重组合



X input documents

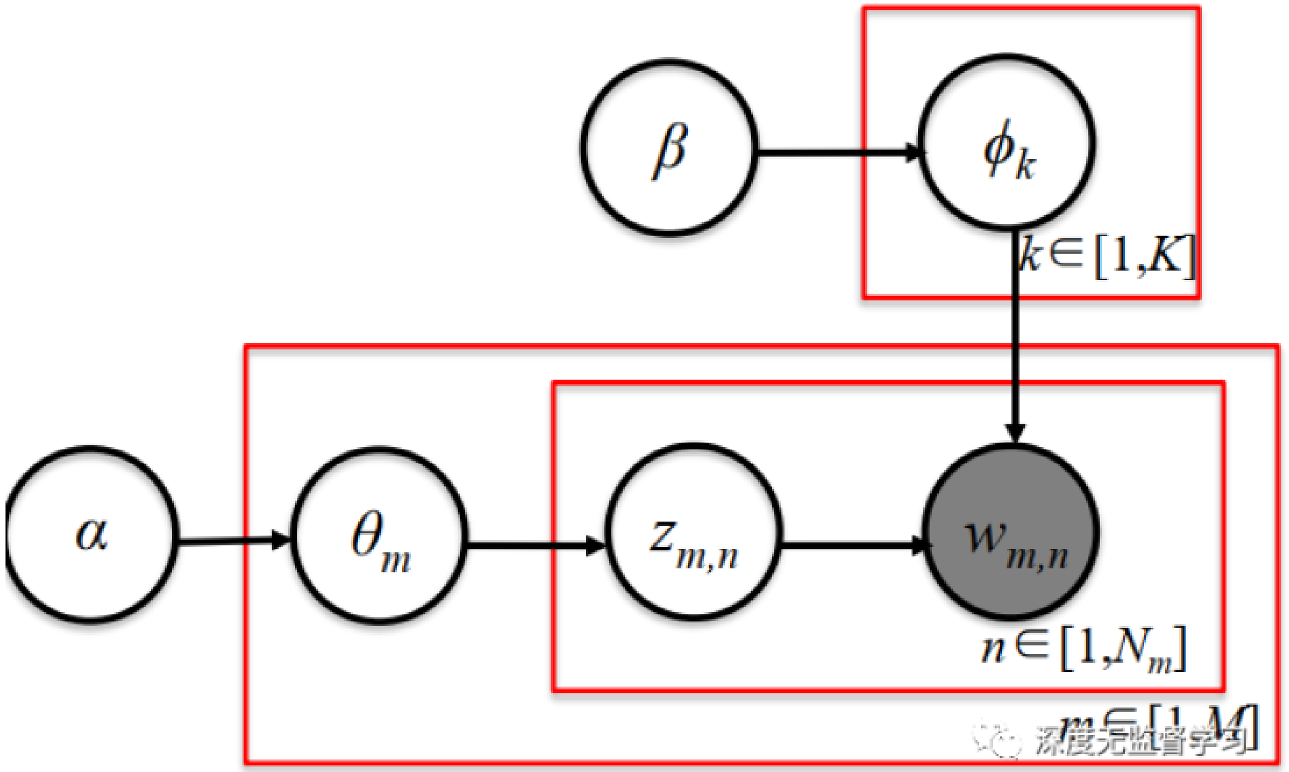
U latent topics

$\mathbf{V}^T$ docs represented with topics

文档-主题生成模型

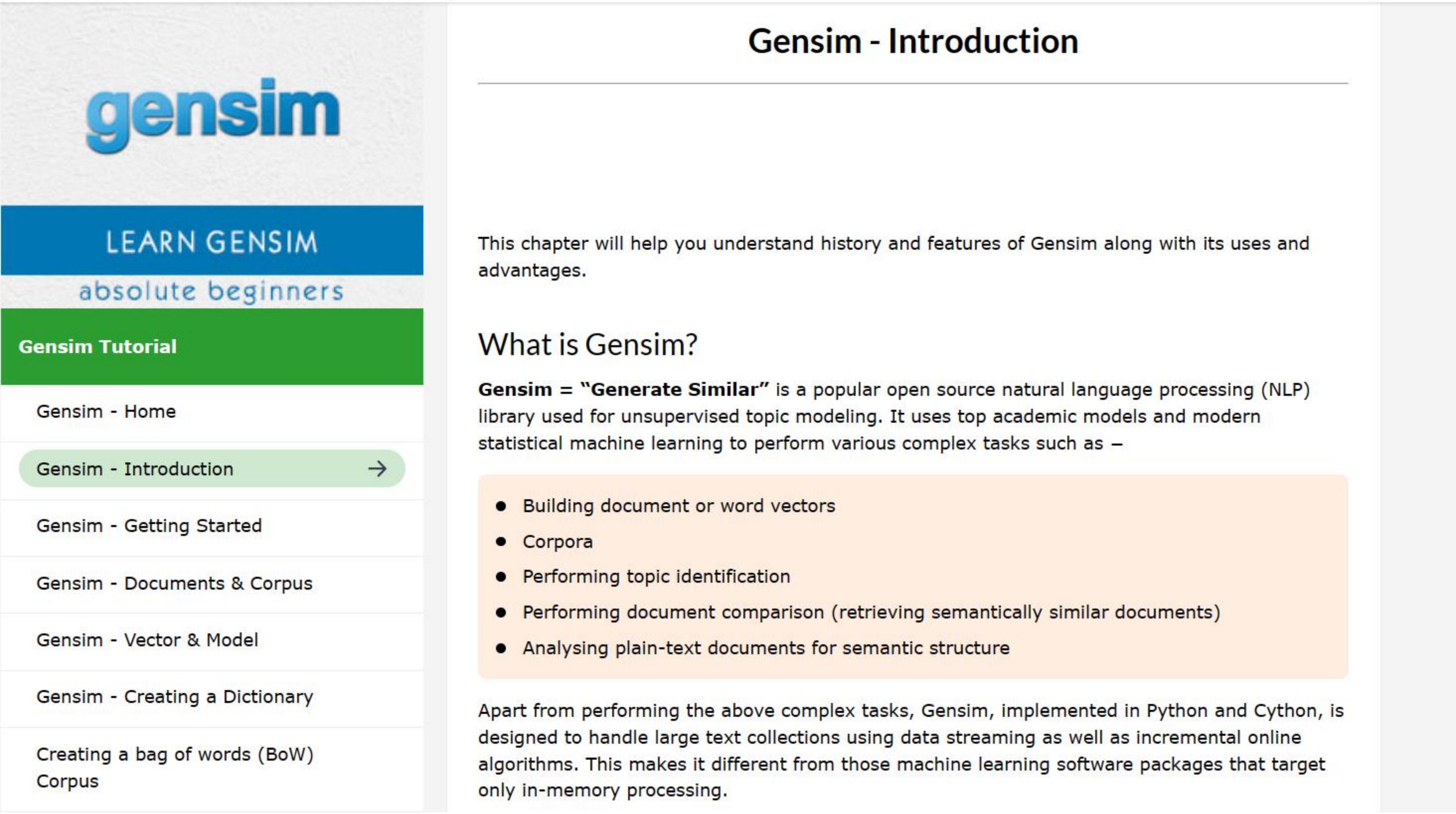
3.2 LDA模型

- LDA图模型如下，



图模型的表示

- 对于每个主题$z_k, k \in [1, K]$，假设主题的分布参数服从狄利克雷分布$\vec{\phi}_k \sim \mathrm{Dir}(\vec{\beta})$
- 对于每个文档$d_m, m \in [1, M]$，假设文档的分布参数服从狄利克雷分布$\vec{\theta}_m \sim \mathrm{Dir}(\vec{\alpha})$
- 以上两个分布都是关于分布参数的概率分布，$\vec{\alpha}, \vec{\beta}$都是超参数
- 假设文档长度服从泊松分布$N_m \sim \mathrm{Poiss}(\xi)$
- 对于每个词$w_{m,n}, n \in [1, N_m]$，假设其主题服从多项分布$z_{m,n} \sim \mathrm{Mult}(\vec{\theta}_m)$
- 词汇由主题生成，单个词语服从多项式分布$w_{m,n} \sim \mathrm{Mult}(\vec{\phi}_{z_{m,n}})$



LEARN GENSIM

absolute beginners

Gensim - Introduction

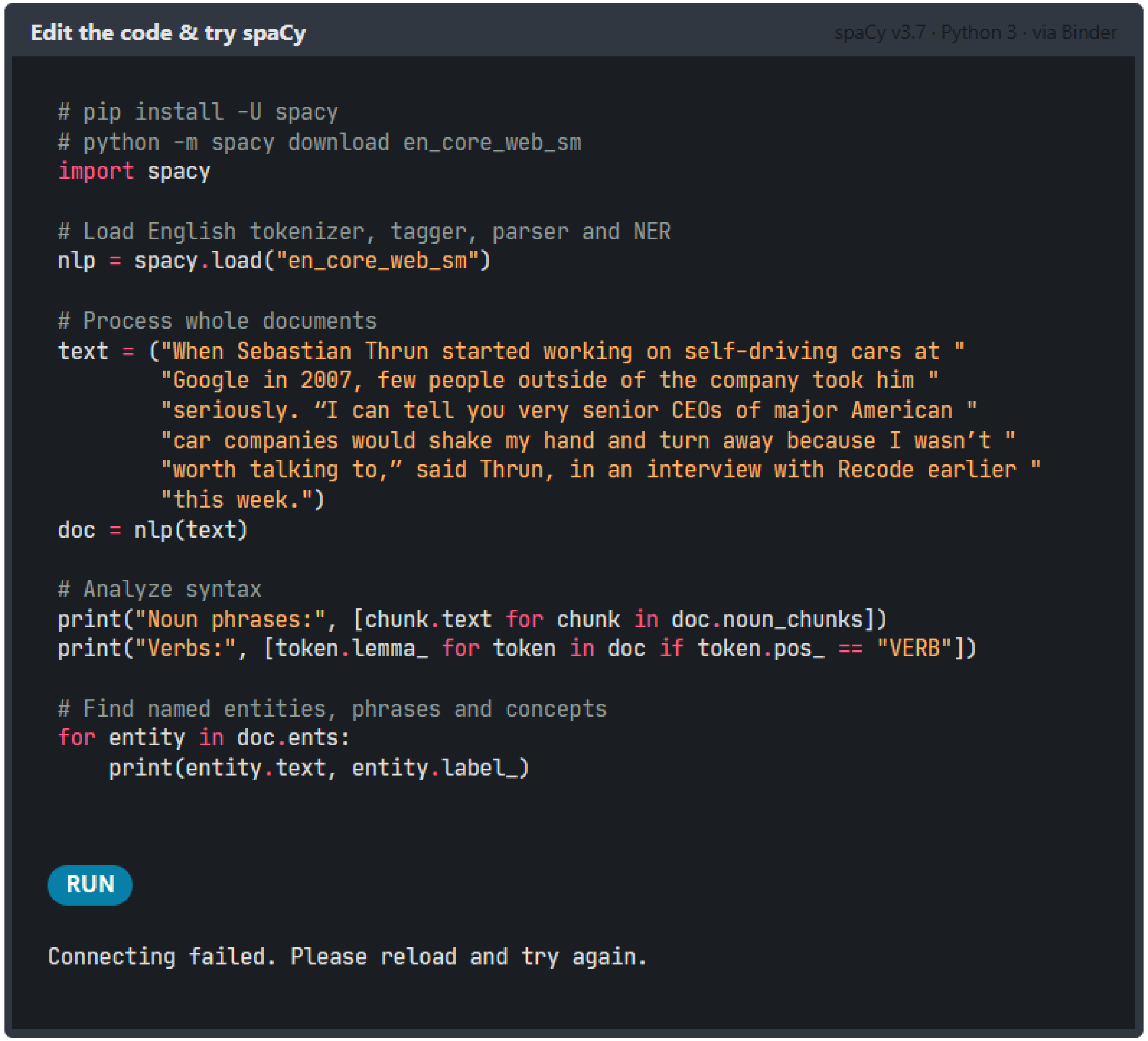
This chapter will help you understand history and features of Gensim along with its uses and advantages.

What is Gensim?

Gensim = "Generate Similar" is a popular open source natural language processing (NLP) library used for unsupervised topic modeling. It uses top academic models and modern statistical machine learning to perform various complex tasks such as –

- Building document or word vectors
- Corpora
- Performing topic identification
- Performing document comparison (retrieving semantically similar documents)
- Analysing plain-text documents for semantic structure

Apart from performing the above complex tasks, Gensim, implemented in Python and Cython, is designed to handle large text collections using data streaming as well as incremental online algorithms. This makes it different from those machine learning software packages that target only in-memory processing.

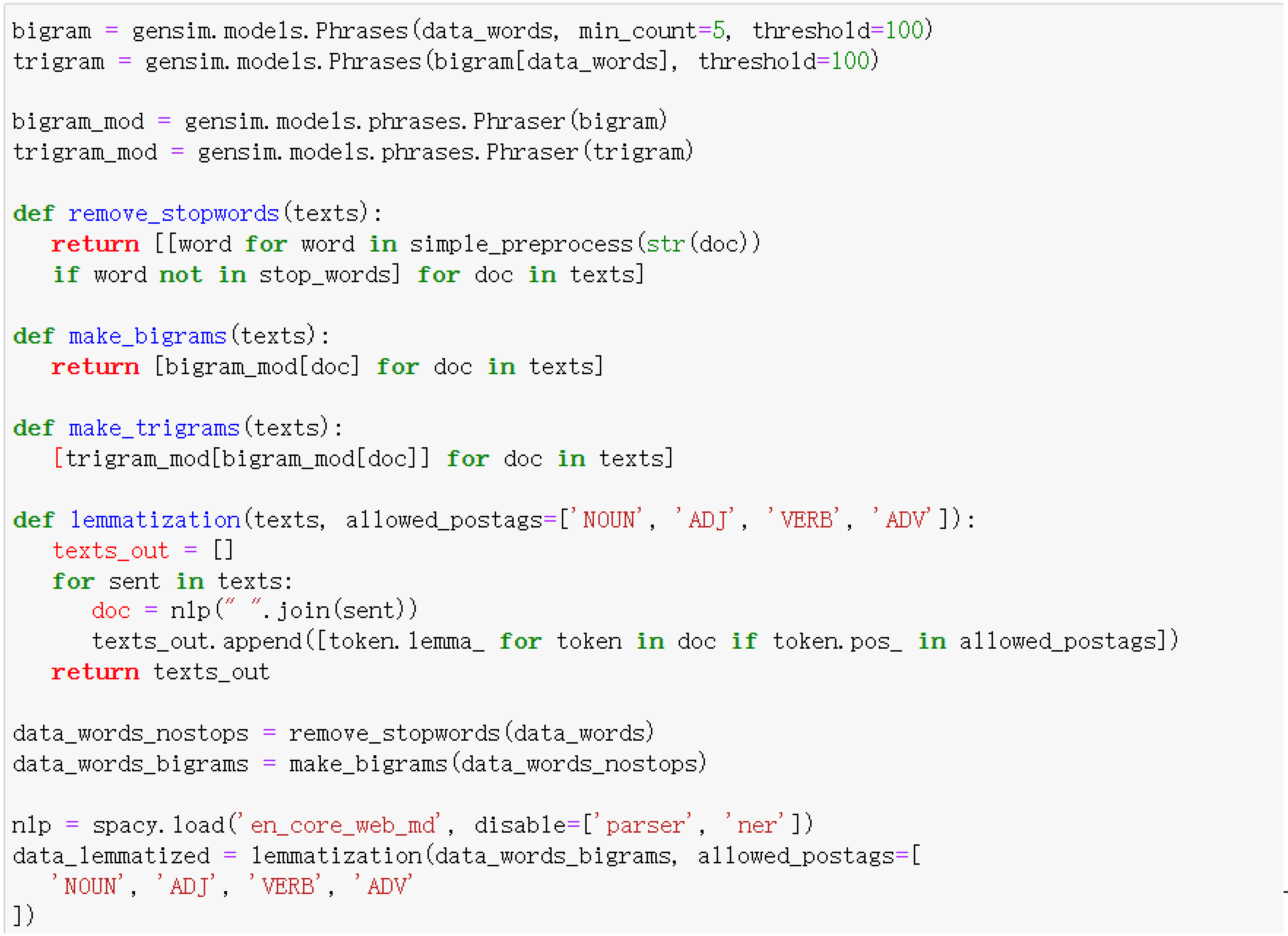


```
# pip install -U spacy
# python -m spacy download en_core_web_sm
import spacy

# Load English tokenizer, tagger, parser and NER
nlp = spacy.load("en_core_web_sm")

# Process whole documents
text = ("When Sebastian Thrun started working on self-driving cars at "
        "Google in 2007, few people outside of the company took him "
        "seriously. “I can tell you very senior CEOs of major American "
        "car companies would shake my hand and turn away because I wasn’t "
        "worth talking to,” said Thrun, in an interview with Recode earlier "
        "this week.")
doc = nlp(text)

# Analyze syntax
print("Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
print("Verbs:", [token.lemma_ for token in doc if token.pos_ == "VERB"])

# Find named entities, phrases and concepts
for entity in doc.ents:
    print(entity.text, entity.label_)
```

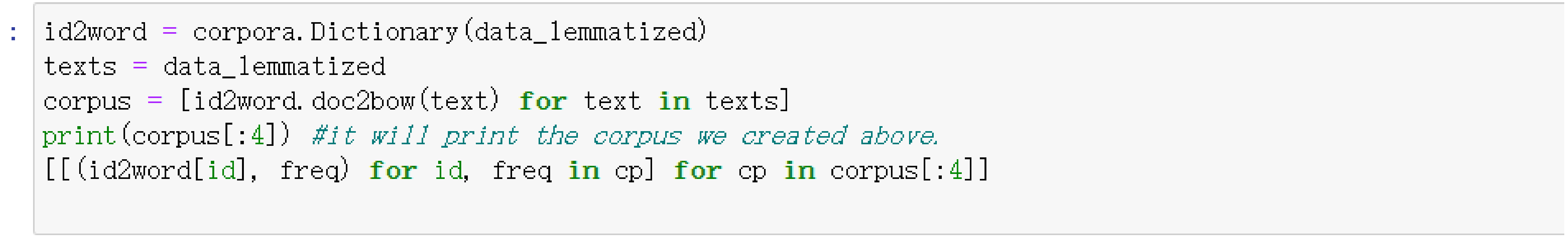
```
Connecting failed. Please reload and try again.
```

Features

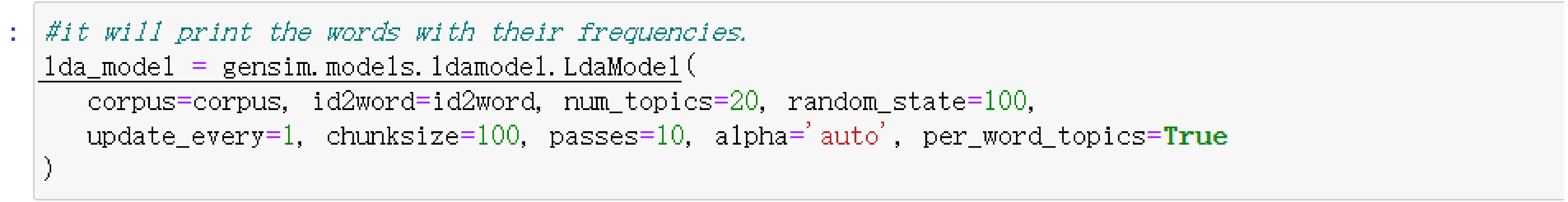
- ✓ Support for **73+ languages**
- ✓ **84 trained pipelines** for 25 languages
- ✓ Multi-task learning with pretrained **transformers** like BERT
- ✓ Pretrained **word vectors**
- ✓ State-of-the-art speed
- ✓ Production-ready **training system**
- ✓ Linguistically-motivated **tokenization**
- ✓ Components for **named entity** recognition, part-of-speech tagging, dependency parsing, sentence segmentation, **text classification**, lemmatization, morphological analysis, entity linking and more
- ✓ Easily extensible with **custom components** and attributes
- ✓ Support for custom models in **PyTorch**, **TensorFlow** and other frameworks
- ✓ Built in **visualizers** for syntax and NER
- ✓ Easy **model packaging**, deployment and workflow management
- ✓ Robust, rigorously evaluated accuracy



```
bigram = gensim.models.Phrases(data_words, min_count=5, threshold=100)
trigram = gensim.models.Phrases(bigram[data_words], threshold=100)

bigram_mod = gensim.models.phrases.Phraser(bigram)
trigram_mod = gensim.models.phrases.Phraser(trigram)

def remove_stopwords(texts):
   return [[word for word in simple_preprocess(str(doc))
   if word not in stop_words] for doc in texts]

def make_bigrams(texts):
   return [bigram_mod[doc] for doc in texts]

def make_trigrams(texts):
   [trigram_mod[bigram_mod[doc]] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
   texts_out = []
   for sent in texts:
      doc = nlp(" ".join(sent))
      texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
   return texts_out

data_words_nostops = remove_stopwords(data_words)
data_words_bigrams = make_bigrams(data_words_nostops)

nlp = spacy.load('en_core_web_md', disable=['parser', 'ner'])
data_lemmatized = lemmatization(data_words_bigrams, allowed_postags=[
   'NOUN', 'ADJ', 'VERB', 'ADV'
])
```



```
:  id2word = corpora.Dictionary(data_lemmatized)
   texts = data_lemmatized
   corpus = [id2word.doc2bow(text) for text in texts]
   print(corpus[:4]) #it will print the corpus we created above.
   [[(id2word[id], freq) for id, freq in cp] for cp in corpus[:4]]
```



```
:  #it will print the words with their frequencies.
   lda_model = gensim.models.ldamodel.LdaModel(
      corpus=corpus, id2word=id2word, num_topics=20, random_state=100,
      update_every=1, chunksize=100, passes=10, alpha='auto', per_word_topics=True
   )
```

进一步参考: [Gensim Topic Modeling - A Guide to Building Best LDA models (machinelearningplus.com)]()