实习三：非关系数据

成员：蔡鑫豪2100013113 马啸宇2100013083 章毅2100013107

递归查询

In [ ]:
```
%load_ext sql
```

In [ ]:
```
import pymysql
pymysql.install_as_MySQLdb()
%sql mysql://stu2100013113:stu2100013113@162.105.146.37:43306
```

In [ ]:
```
%sql use stu2100013113;
```

```
 * mysql://stu2100013113:***@162.105.146.37:43306
0 rows affected.
```

Out[ ]:
```
[]
```

In [ ]:
```
%%sql
drop table if exists family;
create table  family( father char(10), son char(10) );
insert into family(father, son) values ('司马防','司马懿');
insert into family(father, son) values ('司马防','司马孚');
insert into family(father, son) values ('司马防','司马馗');
insert into family(father, son) values ('司马懿','司马师');
insert into family(father, son) values ('司马懿','司马昭');
insert into family(father, son) values ('司马懿','司马亮');
insert into family(father, son) values ('司马懿','司马伦');
insert into family(father, son) values ('司马孚','司马瑰');
insert into family(father, son) values ('司马馗','司马泰');
insert into family(father, son) values ('司马师','司马攸');
insert into family(father, son) values ('司马昭','司马炎');
insert into family(father, son) values ('司马泰','司马越');
insert into family(father, son) values ('司马攸','司马囧');
insert into family(father, son) values ('司马炎','司马衷');
insert into family(father, son) values ('司马炎','司马玮');
insert into family(father, son) values ('司马炎','司马乂');
insert into family(father, son) values ('司马炎','司马颖');
insert into family(father, son) values ('司马炎','司马炽');
```

```
 * mysql://stu2100013113:***@162.105.146.37:43306
0 rows affected.
0 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

Out[ ]: `[]`

找出兄弟关系，需要两个儿子有同样的父亲。

In [ ]:

```
%%sql
SELECT f1.son AS brother1, f2.son AS brother2
FROM family f1
JOIN family f2 ON f1.father = f2.father
WHERE f1.son < f2.son;
```

```
 * mysql://stu2100013113:***@162.105.146.37:43306
19 rows affected.
```

Out[ ]:

brother1	brother2
司马孚	司马懿
司马孚	司马馗
司马懿	司马馗
司马伦	司马师
司马亮	司马师
司马伦	司马昭
司马亮	司马昭
司马师	司马昭
司马亮	司马伦
司马炽	司马衷
司马乂	司马衷
司马玮	司马衷
司马炽	司马玮
司马乂	司马玮
司马炽	司马颖
司马乂	司马颖
司马玮	司马颖
司马衷	司马颖
司马乂	司马炽

找出祖先关系，递归查询。祖先的父亲也是祖先。

In [ ]:

```
%%sql
WITH RECURSIVE ancestor_relation AS (
  SELECT father, son FROM family
  UNION ALL
  SELECT ar.father, f.son
  FROM ancestor_relation ar JOIN family f ON ar.son = f.father
)
SELECT * FROM ancestor_relation;
```

```
 * mysql://stu2100013113:***@162.105.146.37:43306
48 rows affected.
```

Out[ ]:

father	son
司马防	司马懿
司马防	司马孚
司马防	司马馗
司马懿	司马师
司马懿	司马昭
司马懿	司马亮
司马懿	司马伦
司马孚	司马瑰
司马馗	司马泰
司马师	司马攸
司马昭	司马炎
司马泰	司马越
司马攸	司马冏
司马炎	司马衷
司马炎	司马玮
司马炎	司马乂
司马炎	司马颖
司马炎	司马炽
司马防	司马师
司马防	司马昭
司马防	司马亮
司马防	司马伦
司马防	司马瑰
司马防	司马泰
司马懿	司马攸
司马懿	司马炎
司马馗	司马越
司马师	司马冏
司马昭	司马衷
司马昭	司马玮
司马昭	司马乂
司马昭	司马颖
司马昭	司马炽
司马防	司马攸

司马防	司马炎
司马防	司马越
司马懿	司马冏
司马懿	司马衷
司马懿	司马玮
司马懿	司马乂
司马懿	司马颖
司马懿	司马炽
司马防	司马冏
司马防	司马衷
司马防	司马玮
司马防	司马乂
司马防	司马颖
司马防	司马炽

找出堂兄弟关系，需要两个儿子的父亲有相同的父亲。即：cousin(X,Y):father(A,X), father(B,Y),brother(A,B)。

```
In [ ]: %%sql
SELECT f1.son AS cousin1, f2.son AS cousin2
FROM family f1
JOIN family f2 ON f1.father < f2.father
JOIN family f3 ON f1.father = f3.son
JOIN family f4 ON f2.father = f4.son
WHERE f3.father = f4.father;
```

```
 * mysql://stu2100013113:***@162.105.146.37:43306
10 rows affected.
```

Out[ ]:

cousin1	cousin2
司马瑰	司马师
司马瑰	司马昭
司马瑰	司马亮
司马瑰	司马伦
司马伦	司马泰
司马亮	司马泰
司马昭	司马泰
司马师	司马泰
司马瑰	司马泰
司马攸	司马炎

找出叔侄关系，需要儿子的父亲与他是兄弟。即uncle(X,Y):-father(Z,Y),brother(X,Z)。

In [ ]:
```
%%sql
SELECT f3.son AS uncle, f2.son AS nephew
FROM family f1
JOIN family f2 ON f1.son = f2.father
JOIN family f3 ON f1.father = f3.father
WHERE f1.son <> f3.son;
```

```
 * mysql://stu2100013113:***@162.105.146.37:43306
18 rows affected.
```

Out[ ]:

uncle	nephew
司马馗	司马师
司马孚	司马师
司马馗	司马昭
司马孚	司马昭
司马馗	司马亮
司马孚	司马亮
司马馗	司马伦
司马孚	司马伦
司马馗	司马瑰
司马懿	司马瑰
司马孚	司马泰
司马懿	司马泰
司马伦	司马攸
司马亮	司马攸
司马昭	司马攸
司马伦	司马炎
司马亮	司马炎
司马师	司马炎

In [ ]:

窗口查询

In [ ]:
```
%load_ext sql
```

In [ ]:
```
import pymysql
import pandas as pd
```

In [ ]:
```
conn = pymysql.connect(host='162.105.146.37',user='stu2100013113', password='stu
cursor = conn.cursor()
cursor.execute("drop table if exists my_stock_table")
print('1')
```

```
1
```

```
In [ ]: create = "CREATE TABLE IF NOT EXISTS my_stock_table( \
            id INT PRIMARY KEY AUTO_INCREMENT,\
            ts_code VARCHAR(10),\
            trade_date DATE,\
            open FLOAT,\
            high FLOAT,\
            low FLOAT,\
            close FLOAT,\
            pre_close FLOAT,\
            pct_chg FLOAT,\
            vol BIGINT,\
            amount FLOAT);"
        cursor.execute(create)

Out[ ]: 0
```

```
In [ ]: data = pd.read_csv("csi_300.csv", header=0,
                                      sep=",").fillna(0)
        headers = data.columns.values.tolist()
        data = data.to_numpy().tolist()
        tablename = "my_stock_table"
        sql = "insert into {tablename} ({columns}) values({data})".format(
                    tablename=tablename, columns=",".join(headers), data=('%s,'*len(data
        )
        cursor.executemany(sql, data)

Out[ ]: 247437
```

```
In [ ]: cursor.execute("select * from my_stock_table LIMIT 1")
        print(cursor.fetchall())

((1, '000001.SZ', datetime.date(2020, 1, 2), 16.65, 16.95, 16.55, 16.87, 16.45,
2.5532, 1530230, 2571200.0),)
```

```
In [ ]: conn.commit()
```

```
In [ ]: pymysql.install_as_MySQLdb()
        %sql mysql://stu2100013113:stu2100013113@162.105.146.37:43306
        %sql use stu2100013113;

 * mysql://stu2100013113:***@162.105.146.37:43306
0 rows affected.

Out[ ]: []
```

计算Alpha#12，33，54，101.为了展示结果，这里设置了LIMIT 20.

```
In [ ]: %%sql
        SELECT
          ts_code,
          trade_date,
          SIGN(close - LAG(close, 1) OVER (PARTITION BY ts_code ORDER BY trade_date)) *
        FROM
          my_stock_table LIMIT 20;

 * mysql://stu2100013113:***@162.105.146.37:43306
20 rows affected.
```

Out[ ]:

ts_code	trade_date	Alpha12
000001.SZ	2020-01-02	None
000001.SZ	2020-01-03	-0.3099994659423828
000001.SZ	2020-01-06	-0.1100006103515625
000001.SZ	2020-01-07	-0.07999992370605469
000001.SZ	2020-01-08	-0.48999977111816406
000001.SZ	2020-01-09	-0.13000106811523438
000001.SZ	2020-01-10	-0.10000038146972656
000001.SZ	2020-01-13	-0.2999992370605469
000001.SZ	2020-01-14	-0.22999954223632812
000001.SZ	2020-01-15	-0.23999977111816406
000001.SZ	2020-01-16	-0.1900005340576172
000001.SZ	2020-01-17	-0.05999946594238281
000001.SZ	2020-01-20	-0.060001373291015625
000001.SZ	2020-01-21	-0.4500007629394531
000001.SZ	2020-01-22	-0.09000015258789062
000001.SZ	2020-01-23	-0.5500001907348633
000001.SZ	2020-02-03	-1.5500001907348633
000001.SZ	2020-02-04	-0.6100006103515625
000001.SZ	2020-02-05	-0.029999732971191406
000001.SZ	2020-02-06	-0.1400003433227539

In [ ]:

```
%%sql
SELECT
  ts_code,
  trade_date,
  RANK() OVER (ORDER BY -1 * POW(1 - open / close, 1)) / COUNT(*) OVER () AS Alp
FROM
 my_stock_table LIMIT 20;
```

```
 * mysql://stu2100013113:***@162.105.146.37:43306
20 rows affected.
```

Out[ ]:

ts_code	trade_date	Alpha33
300979.SZ	2021-04-26	0.0000
601728.SH	2021-08-20	0.0000
688126.SH	2020-07-29	0.0000
688396.SH	2020-02-28	0.0000
300763.SZ	2020-09-11	0.0000
000708.SZ	2020-02-04	0.0000
688005.SH	2020-02-10	0.0000
688126.SH	2020-07-13	0.0000
688008.SH	2020-02-07	0.0000
600905.SH	2021-06-10	0.0000
300223.SZ	2021-05-27	0.0000
601868.SH	2021-09-28	0.0000
603290.SH	2020-02-04	0.0001
605499.SH	2021-05-27	0.0001
605117.SH	2021-04-20	0.0001
603195.SH	2020-02-06	0.0001
300274.SZ	2020-09-11	0.0001
600918.SH	2020-06-03	0.0001
300769.SZ	2021-08-12	0.0001
688012.SH	2020-01-22	0.0001

In [ ]:

```
%%sql
SELECT
  ts_code,
  trade_date,
  (-1 * ((low - close) * POW(open, 5))) / ((low - high) * POW(close, 5)) AS Alph
FROM
  my_stock_table LIMIT 20;
```

```
 * mysql://stu2100013113:***@162.105.146.37:43306
20 rows affected.
```

Out[ ]:

ts_code	trade_date	Alpha54
000001.SZ	2020-01-02	-0.7491799106493531
000001.SZ	2020-01-03	-0.6213854292036899
000001.SZ	2020-01-06	-0.3655988619759651
000001.SZ	2020-01-07	-0.6025315686411671
000001.SZ	2020-01-08	-0.07902262769854222
000001.SZ	2020-01-09	-0.6538815030491126
000001.SZ	2020-01-10	-0.6039826314505853
000001.SZ	2020-01-13	-0.8426368738343487
000001.SZ	2020-01-14	0.0
000001.SZ	2020-01-15	-0.18514661466523072
000001.SZ	2020-01-16	-0.3722712005522921
000001.SZ	2020-01-17	-0.19938679888262384
000001.SZ	2020-01-20	-0.38228404419834405
000001.SZ	2020-01-21	-0.18965872189155888
000001.SZ	2020-01-22	-0.8007675839175155
000001.SZ	2020-01-23	-0.31935576778041175
000001.SZ	2020-02-03	0.0
000001.SZ	2020-02-04	-0.7479381600796723
000001.SZ	2020-02-05	-0.5364654613173272
000001.SZ	2020-02-06	-0.7320562519400782

In [ ]:

```
%%sql
SELECT
  ts_code,
  trade_date,
  (close - open) / (high - low + .001) AS Alpha101
FROM
  my_stock_table LIMIT 20;
```

```
 * mysql://stu2100013113:***@162.105.146.37:43306
20 rows affected.
```

Out[ ]:

ts_code	trade_date	Alpha101
000001.SZ	2020-01-02	0.548629385439198
000001.SZ	2020-01-03	0.6138111144724728
000001.SZ	2020-01-06	0.1392097992086394
000001.SZ	2020-01-07	0.0604243576244245
000001.SZ	2020-01-08	-0.8076011662062029
000001.SZ	2020-01-09	-0.049871744238403244
000001.SZ	2020-01-10	-0.3436450938290258
000001.SZ	2020-01-13	0.5700706119364085
000001.SZ	2020-01-14	-0.45009674993612064
000001.SZ	2020-01-15	-0.6569356642435502
000001.SZ	2020-01-16	-0.5121322939997909
000001.SZ	2020-01-17	0.049752665767379876
000001.SZ	2020-01-20	0.07663003917412958
000001.SZ	2020-01-21	-0.8272512866579549
000001.SZ	2020-01-22	0.3769404616178109
000001.SZ	2020-01-23	-0.7156314605181436
000001.SZ	2020-02-03	0.0
000001.SZ	2020-02-04	0.858035435940924
000001.SZ	2020-02-05	0.07005239303704566
000001.SZ	2020-02-06	-0.11080332380700725

JSON操作

In [ ]:
```
%load_ext sql
```

In [ ]:
```
import pymysql
pymysql.install_as_MySQLdb()
%sql mysql://stu2100013107:stu2100013107@162.105.146.37:43306
```

In [ ]:
```
%sql use stu2100013107;
```

```
 * mysql://stu2100013107:***@162.105.146.37:43306
0 rows affected.
```

Out[ ]:
```
[]
```

手动处理文件，删除所有注释并且修改建表指令为drop table if exists ***

In [ ]:
```
file = open("建表.txt",'r')
content = file.read()
sql_commands = content.split(';')
```

```python
db = pymysql.connect(host='162.105.146.37',user='stu2100013107', password='stu21
cursor = db.cursor()

for line in sql_commands:
    try:
        cursor.execute(line)
        db.commit()
    except:
        db.rollback()
```

In [ ]:
```python
file = open("数据.txt",'r')
content = file.read()
sql_commands = content.split(';')

db = pymysql.connect(host='162.105.146.37',user='stu2100013107', password='stu21
cursor = db.cursor()

for line in sql_commands:
    try:
        cursor.execute(line)
        db.commit()
    except:
        db.rollback()
```

使用一个WITH子句来创建一个包含产品编号和名称的临时表RankedProducts，然后在外部查询中过滤出前5个产品,查询将返回一个JSON对象，其中包含每个客户的ID、名称、地址信息以及他们购买的前5种产品信息数组。

In [ ]:
```sql
%%sql
WITH RankedProducts AS (
    SELECT
        o.custid,
        p.productid,
        p.productname,
        od.unitprice * od.qty * (1 - od.discount) AS totalAmount,
        ROW_NUMBER() OVER (PARTITION BY o.custid ORDER BY o.orderid) AS product_
    FROM Orders o
    LEFT JOIN OrderDetails od ON o.orderid = od.orderid
    JOIN Products p ON od.productid = p.productid
)
SELECT
    JSON_OBJECT(
        'customerId', c.custid,
        'customerName', c.contactname,
        'addr', JSON_OBJECT(
            'country', c.country,
            'city', c.city
        ),
        'products', JSON_ARRAYAGG(
            concat(rp.productname,':',rp.totalAmount)
        )
    ) AS customer_json
FROM Customers c
LEFT JOIN RankedProducts rp ON c.custid = rp.custid
WHERE rp.product_rank <= 5 OR rp.product_rank IS NULL
GROUP BY c.custid
LIMIT 10
```

```
 * mysql://stu2100013107:***@162.105.146.37:43306
10 rows affected.
```

Out[ ]:

customer_json
{"addr": {"city": "Berli", "country": "Germany"}, "products": ["Product OFBNT:513.000000000", "Product LSOFL:283.500000000", "Product CBRRL:18.000000000", "Product ICKNK:878.000000000", "Product IMEHJ:60.000000000"], "customerId": 1, "customerName": "Allen, Michael"}
{"addr": {"city": "México D.F.", "country": "Mexico"}, "products": ["Product COAXA:28.800000000", "Product TOONT:60.000000000", "Product PWCJB:69.750000000", "Product RJVNM:70.000000000", "Product WHBYK:340.000000000"], "customerId": 2, "customerName": "Hassall, Mark"}
{"addr": {"city": "México D.F.", "country": "Mexico"}, "products": ["Product QMVU:403.200000000", "Product ZZZHR:586.500000000", "Product MYNX:162.562500000", "Product QMVU:945.000000000", "Product YZIXQ:165.600000000"], "customerId": 3, "customerName": "Peoples, Joh"}
{"addr": {"city": "Londo", "country": "UK"}, "products": ["Product QOGNU:90.000000000", "Product OVLQI:390.000000000", "Product POXFU:96.000000000", "Product BIUDV:195.000000000", "Product VKCMF:608.000000000"], "customerId": 4, "customerName": "Arndt, Torste"}
{"addr": {"city": "Luleå", "country": "Swede"}, "products": ["Product VJIEO:248.000000000", "Product UKXRI:660.000000000", "Product ICKNK:280.800000000", "Product WEUJZ:300.000000000", "Product QOGNU:43.200000000"], "customerId": 5, "customerName": "Higginbotham, Tom"}
{"addr": {"city": "Mannheim", "country": "Germany"}, "products": ["Product QAQRL:149.000000000", "Product OFBNT:136.800000000", "Product OVLQI:78.000000000", "Product JYGFE:252.000000000", "Product QMVU:294.000000000"], "customerId": 6, "customerName": "Poland, Carole"}
{"addr": {"city": "Strasbourg", "country": "France"}, "products": ["Product BLCAX:936.000000000", "Product TOONT:240.000000000", "Product LSOFL:864.000000000", "Product GEEOO:556.000000000", "Product OFBNT:1092.000000000"], "customerId": 7, "customerName": "Bansal, Dushyant"}
{"addr": {"city": "Madrid", "country": "Spai"}, "products": ["Product KSBRM:422.400000000", "Product OVLQI:249.600000000", "Product BWRLG:310.000000000", "Product BLCAX:1170.000000000", "Product VJXY:1856.850000000"], "customerId": 8, "customerName": "Ilyina, Julia"}
{"addr": {"city": "Marseille", "country": "France"}, "products": ["Product QAQRL:88.500000000", "Product CKEDC:950.000000000", "Product TTEEX:87.780000000", "Product ZZZHR:1398.400000000", "Product LYLNI:560.000000000"], "customerId": 9, "customerName": "Raghav, Amritansh"}
{"addr": {"city": "Tsawasse", "country": "Canada"}, "products": ["Product YHXGE:396.800000000", "Product YYWRT:288.000000000", "Product WUXYK:788.000000000", "Product TOONT:360.000000000", "Product ASTM:98.000000000"], "customerId": 10, "customerName": "Bassols, Pilar Colome"}

In [ ]:

```
%%sql
drop table if exists custInfo;
CREATE TABLE custInfo (
    id INT AUTO_INCREMENT PRIMARY KEY,
    Info JSON
);

INSERT INTO custInfo (Info)
WITH RankedProducts AS (
    SELECT
```

```
        o.custid,
        p.productid,
        p.productname,
        od.unitprice * od.qty * (1 - od.discount) AS totalAmount,
        ROW_NUMBER() OVER (PARTITION BY o.custid ORDER BY o.orderid) AS product_
    FROM Orders o
    LEFT JOIN OrderDetails od ON o.orderid = od.orderid
    JOIN Products p ON od.productid = p.productid
)
SELECT
    JSON_OBJECT(
                'customerId', c.custid,
        'customerName', c.contactname,
        'addr', JSON_OBJECT(
            'country', c.country,
            'city', c.city
        ),
        'products', JSON_ARRAYAGG(
                concat(rp.productname, ':', rp.totalAmount)
        )
    ) AS customer_json
FROM Customers c
LEFT JOIN RankedProducts rp ON c.custid = rp.custid
WHERE rp.product_rank <= 5 OR rp.product_rank IS NULL
GROUP BY c.custid
```

```
 * mysql://stu2100013107:***@162.105.146.37:43306
0 rows affected.
0 rows affected.
91 rows affected.
```

Out[ ]: []

In [ ]:
```
%%sql
SELECT id,Info
FROM custInfo
WHERE JSON_EXTRACT(Info, '$.addr.country') = 'USA'
```

```
 * mysql://stu2100013107:***@162.105.146.37:43306
13 rows affected.
```

Out[ ]:

id	Info
32	{"addr": {"city": "Eugene", "country": "USA"}, "products": ["Product QMVU:63.000000000", "Product ASTM:16.000000000", "Product GEEOO:313.200000000", "Product NEVTJ:72.000000000", "Product QDOMO:3754.875000000"], "customerId": 32, "customerName": "Krishnan, Venky"}
36	{"addr": {"city": "Elgi", "country": "USA"}, "products": ["Product PWCJB:279.000000000", "Product QAQRL:59.000000000", "Product POXFU:48.000000000", "Product WUXYK:394.000000000", "Product BLCAX:62.400000000"], "customerId": 36, "customerName": "Smith, Denise"}
43	{"addr": {"city": "Walla Walla", "country": "USA"}, "products": ["Product YZIXQ:147.000000000", "Product QMVU:210.000000000"], "customerId": 43, "customerName": "Deshpande, Anu"}
45	{"addr": {"city": "San Francisco", "country": "USA"}, "products": ["Product KSZOI:155.000000000", "Product BWRLG:162.750000000", "Product CKEDC:562.500000000", "Product LYERX:58.252500000", "Product QAQRL:223.500000000"], "customerId": 45, "customerName": "Sunkammurali, Krishna"}
48	{"addr": {"city": "Portland", "country": "USA"}, "products": ["Product WUXYK:394.000000000", "Product TBTBL:30.000000000", "Product HHYDP:288.000000000", "Product OFBNT:319.200000000", "Product XLXQF:98.000000000"], "customerId": 48, "customerName": "Szymczak, Radosław"}
55	{"addr": {"city": "Anchorage", "country": "USA"}, "products": ["Product CKEDC:1125.000000000", "Product VJXY:2227.500000000", "Product LSOFL:388.800000000", "Product BLCAX:624.000000000", "Product LYERX:310.500000000"], "customerId": 55, "customerName": "Egelund-Muller, Anja"}
65	{"addr": {"city": "Albuquerque", "country": "USA"}, "products": ["Product EPEIM:163.200000000", "Product HMLNI:360.000000000", "Product VKCMF:60.800000000", "Product QHFFP:388.800000000", "Product XWOXC:400.000000000"], "customerId": 65, "customerName": "Moore, Michael"}
71	{"addr": {"city": "Boise", "country": "USA"}, "products": ["Product PAFRH:248.115000000", "Product NEVTJ:856.800000000", "Product CBRRL:288.000000000", "Product UKXRI:1496.000000000", "Product ICKNK:2386.800000000"], "customerId": 71, "customerName": "Navarro, Tomás"}
75	{"addr": {"city": "Lander", "country": "USA"}, "products": ["Product ASTM:48.000000000", "Product XKXDO:69.350000000", "Product LYERX:157.320000000", "Product QDOMO:4005.200000000", "Product VKCMF:346.560000000"], "customerId": 75, "customerName": "Wojciechowska, Agnieszka"}
77	{"addr": {"city": "Portland", "country": "USA"}, "products": ["Product PAFRH:139.000000000", "Product WUXYK:197.000000000", "Product EPEIM:85.400000000", "Product GMKIJ:95.000000000", "Product SWNJY:140.000000000"], "customerId": 77, "customerName": "Osorio, Cristia"}
78	{"addr": {"city": "Butte", "country": "USA"}, "products": ["Product OFBNT:456.000000000", "Product VJXY:742.740000000", "Product VJIEO:194.500000000", "Product YHXGE:186.000000000", "Product XLXQF:42.000000000"], "customerId": 78, "customerName": "Young, Robi"}
82	{"addr": {"city": "Kirkland", "country": "USA"}, "products": ["Product ASTM:35.000000000", "Product YZIXQ:36.800000000", "Product WUXYK:493.000000000", "Product HCQDE:199.500000000", "Product LSOFL:180.000000000"], "customerId": 82, "customerName": "Veninga, Tjeerd"}
89	{"addr": {"city": "Seattle", "country": "USA"}, "products": ["Product ASTM:114.000000000", "Product GEEOO:528.200000000", "Product

```
KSBRM:616.000000000", "Product WVJFP:1680.000000000", "Product
RECZE:516.800000000"], "customerId": 89, "customerName": "Smith Jr., Ronaldo"}
```

In [ ]:
```
%%sql
SELECT
    SUBSTRING_INDEX(SUBSTRING_INDEX(product_info, ':', 1), '"', -1) AS product_n
    ROUND(SUM(SUBSTRING_INDEX(SUBSTRING_INDEX(product_info, ':', -1), '"', 1)),2
FROM
    custInfo,
    JSON_TABLE(
        Info,
        '$.products[*]' COLUMNS (
            product_info VARCHAR(100) PATH '$'
        )
    ) AS jt
GROUP BY
    product_name
LIMIT 10
```

```
 * mysql://stu2100013107:***@162.105.146.37:43306
10 rows affected.
```

Out[ ]:

product_name	total_amount
Product OFBNT	7251.0
Product LSOFL	3321.9
Product CBRRL	876.72
Product ICKNK	3984.5
Product IMEHJ	844.0
Product COAXA	3828.96
Product TOONT	2333.4
Product PWCJB	1799.55
Product RJVNM	688.8
Product WHBYK	8343.6

向量数据库实习设计

In [ ]:
```
import pandas as pd
import numpy as np
import psycopg2
from sqlalchemy import create_engine
import matplotlib.pyplot as plt
from wordcloud import WordCloud
```

In [ ]:
```
%load_ext sql
%sql postgresql://postgres:20020912@localhost:5432/hw
```

1、建立小说表 yttlj 和人物表 person 。

In [ ]:
```
%%sql
CREATE TABLE IF NOT EXISTS yttlj (
    "phaseId" INT PRIMARY KEY,
    "phaseText" TEXT NOT NULL
);
CREATE TABLE IF NOT EXISTS person (
    "personId" INT PRIMARY KEY,
    "personName" TEXT NOT NULL
);
```

```
 * postgresql://postgres:***@localhost:5432/hw
Done.
Done.
```

Out[ ]:
```
[]
```

In [ ]:
```
yttlj_df = pd.read_csv('yttlj.csv')
# person_df = pd.read_csv('Person.csv') # 没法读入于是在下面手动输入
data = {
    'personId': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 1
    'personName': ['张三丰', '张翠山', '殷素素', '俞岱岩', '俞莲舟', '殷梨亭', '
                   '殷离', '小昭', '黛绮丝', '谢逊', '朱长龄', '朱九真', '杨逍',
}
person_df = pd.DataFrame(data)
```

In [ ]:
```
engine = create_engine('postgresql://postgres:20020912@localhost:5432/hw')
# 插入DataFrame到数据库表
yttlj_df.to_sql('yttlj', engine, if_exists='append', index=False)
person_df.to_sql('person', engine, if_exists='append', index=False)
```

Out[ ]:
```
21
```

2、为人物表添加一个字段，统计每个人物的词频，也即他在多少段落中出现过。

In [ ]:
```
# 统计每个人物的词频
def count_occurrences(text, names):
    counts = {name: 0 for name in names}
    for name in names:
        counts[name] = text.count(name)
    return counts


paragraphs = yttlj_df['phaseText'].tolist()
names = person_df['personName'].tolist()
# 初始化词频统计字典
total_counts = {name: 0 for name in names}
# 统计每个人物在段落中的出现频率
for paragraph in paragraphs:
    counts = count_occurrences(paragraph, names)
    for name, count in counts.items():
        if count > 0:
            total_counts[name] += 1

# 更新person DataFrame中的词频
person_df['wordCount'] = person_df['personName'].map(total_counts)

# 更新person表中的wordCount字段
person_df.to_sql('person', engine, if_exists='replace', index=False)

%sql select * from person
```

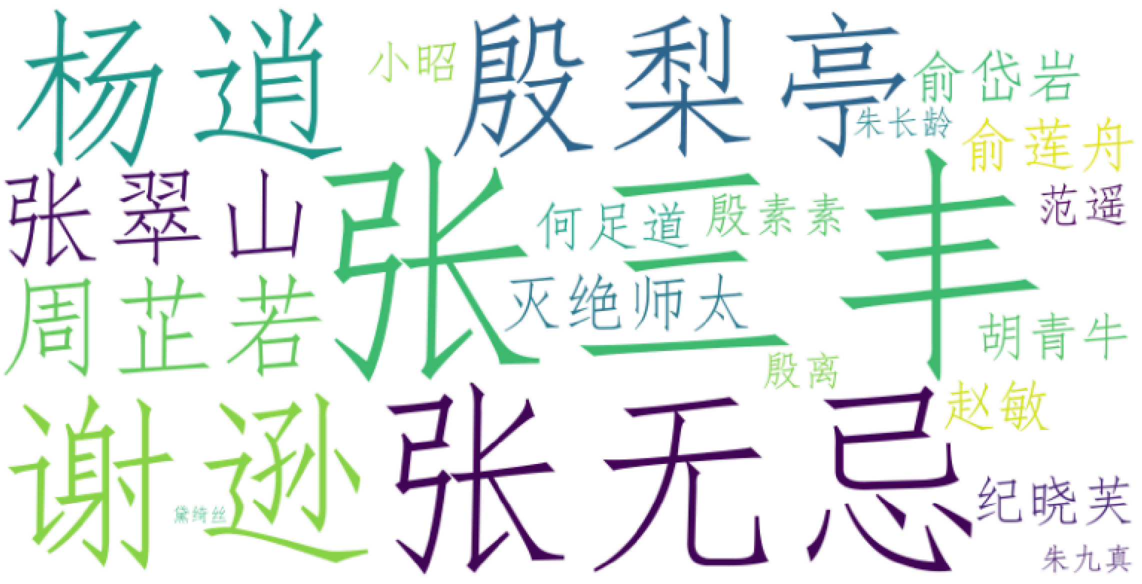
```
 * postgresql://postgres:***@localhost:5432/hw
21 rows affected.
```

Out[ ]:

personId	personName	wordCount
0	张三丰	34
1	张翠山	23
2	殷素素	13
3	俞岱岩	20
4	俞莲舟	21
5	殷梨亭	24
6	纪晓芙	17
7	张无忌	32
8	周芷若	24
9	赵敏	18
10	殷离	9
11	小昭	12
12	黛绮丝	1
13	谢逊	32
14	朱长龄	6
15	朱九真	5
16	杨逍	26
17	范遥	12
18	灭绝师太	23
19	何足道	16
20	胡青牛	16

用Python中的词云工具来显示一下。

In [ ]:
```python
# 生成词云
word_counts = dict(zip(person_df['personName'], person_df['wordCount']))
wordcloud = WordCloud(width=800, height=400, background_color='white', font_path

# 显示词云
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')  # 关闭坐标轴
plt.show()
```



3、建立共现表 cocurrence

```
In [ ]:  # 增加一个类似位向量的字段phaseContains
         %sql ALTER TABLE person ADD COLUMN phasecontains VARBIT(40);
         # 每个位对应一个段落，如果某段落包含了该person，则将该位设置为1
```

```
 * postgresql://postgres:***@localhost:5432/hw
Done.
```

```
Out[ ]:  []
```

```
In [ ]:  %%sql
         DO $$
         DECLARE
             paragraph_count INT;
             person_record RECORD;
             paragraph_record RECORD;
             phase_contains BIT VARYING := '';
         BEGIN
             -- 获取段落总数
             SELECT COUNT(*) INTO paragraph_count FROM yttlj;

             -- 遍历每个人物
             FOR person_record IN SELECT "personId", "personName" FROM person LOOP
                 phase_contains := ''; -- 初始化为空位向量

                 -- 遍历每个段落
                 FOR paragraph_record IN SELECT "phaseText" FROM yttlj LOOP
                     IF POSITION(person_record."personName" IN paragraph_record."phaseTex
                         phase_contains := phase_contains || B'1';
                     ELSE
                         phase_contains := phase_contains || B'0';
                     END IF;
                 END LOOP;

                 -- 更新person表中的phaseContains字段
                 UPDATE person
                 SET phasecontains = phase_contains
                 WHERE "personId" = person_record."personId";
             END LOOP;
         END $$;
```

```
 * postgresql://postgres:***@localhost:5432/hw
Done.
```

Out[ ]: `[]`

In [ ]:
```
%sql SELECT * FROM person
```

```
 * postgresql://postgres:***@localhost:5432/hw
21 rows affected.
```

Out[ ]:

personId	personName	wordCount	phasecontains
0	张三丰	34	0111111111111111110101111110111101011111
1	张翠山	23	00111111111111110010101010001001100001101
2	殷素素	13	0000111111110000010110010000000000001000
3	俞岱岩	20	00111101111110000000100111110001101000101
4	俞莲舟	21	0011000111100110001101011110001101001101
5	殷梨亭	24	0011000011110100011101111111101101001101
6	纪晓芙	17	0000000001111110111001101010101010000001
7	张无忌	32	0000000101111111111111111111111111111111
8	周芷若	24	0000000000110000111111110111111111101111
9	赵敏	18	0000000000000000000000111111111111111111
10	殷离	9	00000000000000000000000000001111101100011
11	小昭	12	00000000000000000000111111001111000100001
12	黛绮丝	1	00000000000000000000000000000010000000000
13	谢逊	32	00001111111111111101111010011111111111111
14	朱长龄	6	00000000000000110100010000010010000000000
15	朱九真	5	00000000000000110010001000010000000000000
16	杨逍	26	000000000000111011111111111111111011111111
17	范遥	12	00000000000000000000000000111110001111111
18	灭绝师太	23	0000000011101101111001101111111101001101
19	何足道	16	11100000010000101001011010000010000011111
20	胡青牛	16	0000000000111101111011011001011000000001

In [ ]:
```
%%sql
CREATE TABLE IF NOT EXISTS cocurrence (
    "personId1" INT,
    "personId2" INT,
    "counts" INT,
    PRIMARY KEY ("personId1", "personId2")
);
```

```
 * postgresql://postgres:***@localhost:5432/hw
Done.
```

```
Out[ ]:  []
```

```
In [ ]: %%sql
        INSERT INTO cocurrence ("personId1", "personId2", "counts")
        SELECT p1."personId" AS personId1, p2."personId" AS personId2,
               BIT_COUNT(p1.phasecontains & p2.phasecontains) AS counts
        FROM person p1
        JOIN person p2 ON p1."personId" <= p2."personId"
```

```
 * postgresql://postgres:***@localhost:5432/hw
231 rows affected.
```

```
Out[ ]:  []
```

```
In [ ]: %sql select * from cocurrence LIMIT 10
```

```
 * postgresql://postgres:***@localhost:5432/hw
10 rows affected.
```

Out[ ]:

personId1	personId2	counts
0	0	34
0	1	22
0	2	12
0	3	20
0	4	20
0	5	22
0	6	15
0	7	27
0	8	19
0	9	15

4、通过矩阵分解，得到词向量表示

```
In [ ]: def fetch_cocurrence_data():
            # 连接到你的数据库
            conn = psycopg2.connect(dbname='hw', user='postgres', password='20020912', h
            cursor = conn.cursor()

            # 获取 cocurrence 表的数据
            cursor.execute("SELECT \"personId1\", \"personId2\", \"counts\" FROM cocurre
            data = cursor.fetchall()

            cursor.close()
            conn.close()

            return data

        def build_co_occurrence_matrix(vocab_size, cocurrence_data):
            # 初始化共现矩阵
            co_occurrence_matrix = np.zeros((vocab_size, vocab_size), dtype=int)

            # 填充共现矩阵
            for row in cocurrence_data:
```

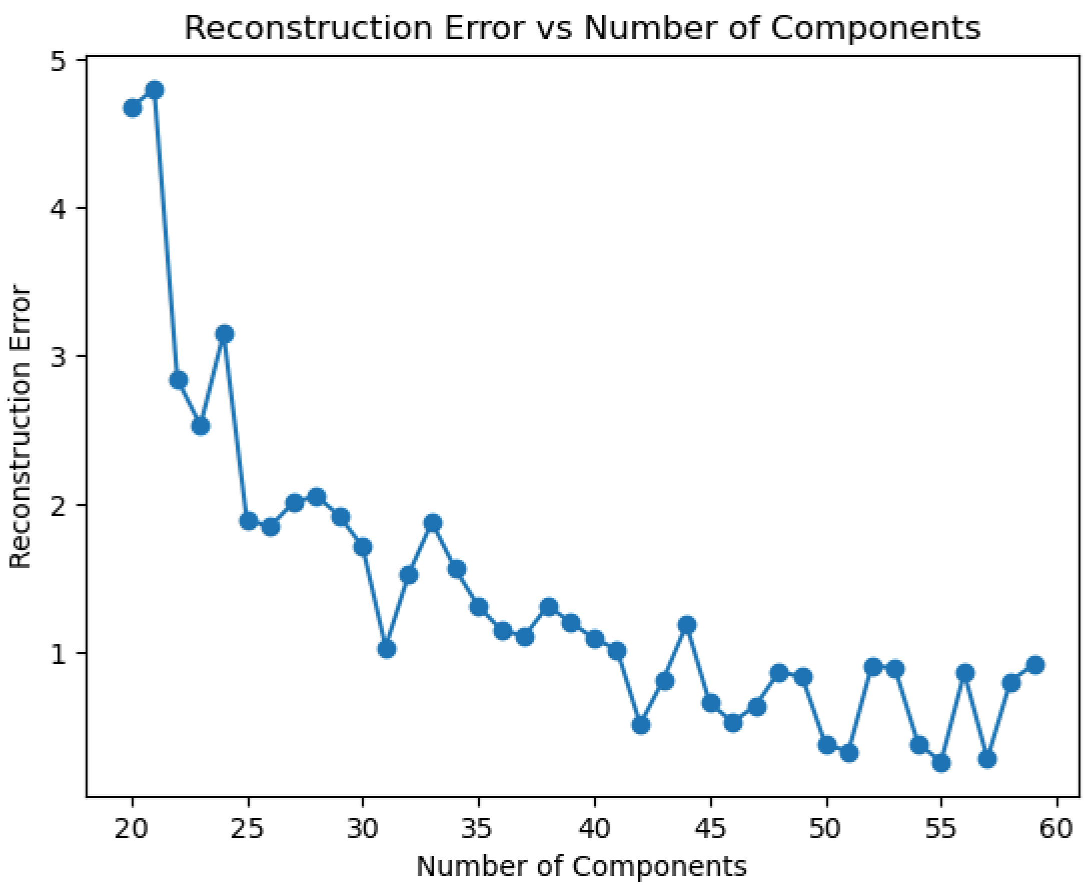
```
        personId1, personId2, counts = row
        co_occurrence_matrix[personId1, personId2] = counts
        co_occurrence_matrix[personId2, personId1] = counts  # 对称矩阵

    return co_occurrence_matrix

cocurrence_data = fetch_cocurrence_data()
cocurrence_matrix = build_co_occurrence_matrix(21, cocurrence_data)
print('共现矩阵: ')
print(cocurrence_matrix)
```

```
共现矩阵:
[[34 22 12 20 20 22 15 27 19 15  7  9  1 27  5  3 21 10 21 13 13]
 [22 23 12 15 16 17 10 17 12  7  4  6  0 20  5  2 12  4 13  8 10]
 [12 12 13  9  7  8  4  9  7  2  0  3  0 12  1  0  5  1  5  4  5]
 [20 15  9 20 16 17  8 15 12 10  4  6  0 15  1  1 10  5 12  5  6]
 [20 16  7 16 21 19 10 18 13 10  4  6  0 17  3  2 14  6 15  9  8]
 [22 17  8 17 19 24 13 21 18 13  6  9  0 18  4  3 17  8 19  9 11]
 [15 10  4  8 10 13 17 17 12  7  3  6  0 14  4  3 14  3 14  4 11]
 [27 17  9 15 18 21 17 32 24 18  9 12  1 28  6  5 26 12 22 12 16]
 [19 12  7 12 13 18 12 24 24 16  9 11  1 21  4  3 21 11 17  9 12]
 [15  7  2 10 10 13  7 18 16 18  9  9  1 15  2  2 17 12 13  7  6]
 [ 7  4  0  4  4  6  3  9  9  9  9  6  1  9  2  1  8  7  7  3  4]
 [ 9  6  3  6  6  9  6 12 11  9  6 12  1 10  3  2 12  5  8  5  8]
 [ 1  0  0  0  0  0  0  1  1  1  1  1  1  1  0  0  1  1  1  1  1]
 [27 20 12 15 17 18 14 28 21 15  9 10  1 32  5  4 22 11 19 13 14]
 [ 5  5  1  1  3  4  4  6  4  2  2  3  0  5  6  3  5  1  5  2  5]
 [ 3  2  0  1  2  3  3  5  3  2  1  2  0  4  3  5  4  1  4  2  3]
 [21 12  5 10 14 17 14 26 21 17  8 12  1 22  5  4 26 12 18 11 13]
 [10  4  1  5  6  8  3 12 11 12  7  5  1 11  1  1 12 12  9  6  3]
 [21 13  5 12 15 19 14 22 17 13  7  8  1 19  5  4 18  9 23  9 13]
 [13  8  4  5  9  9  4 12  9  7  3  5  1 13  2  2 11  6  9 16  8]
 [13 10  5  6  8 11 11 16 12  6  4  8  1 14  5  3 13  3 13  8 16]]
```

In [ ]:

```
def nmf_train(V, components, iternum, e):
    '''
    非负矩阵分解函数
    :param V:  原始矩阵
    :param components:  要提取多少个特征
    :param iternum: 迭代次数
    :param e: 误差阈值
    :return:
    '''
    m, n = V.shape
    # 随机初始化两个矩阵
    W = np.random.random((m, components))
    H = np.random.random((components, n))


    for iter in range(iternum):
        V_pre = np.dot(W, H)
        E = V - V_pre

        err = np.sum(E * E)
        # print(err)
        if err < e:
            break
        # 对照更新公式
        a = np.dot(W.T, V)
        b = np.dot(W.T, np.dot(W, H))
        H[b != 0] = (H * a / b)[b != 0]

        c = np.dot(V, H.T)
        d = np.dot(W, np.dot(H, H.T))

        W[d != 0] = (W * c / d)[d != 0]
    return W, H
```

In [ ]:
```
def nmf_train_err(V, components, iternum, e):
    m, n = V.shape
    W = np.random.random((m, components))
    H = np.random.random((components, n))

    for iter in range(iternum):
        V_pre = np.dot(W, H)
        E = V - V_pre
        err = np.sum(E * E)
        if err < e:
            break
        a = np.dot(W.T, V)
        b = np.dot(W.T, np.dot(W, H))
        H[b != 0] = (H * a / b)[b != 0]
        c = np.dot(V, H.T)
        d = np.dot(W, np.dot(H, H.T))
        W[d != 0] = (W * c / d)[d != 0]
    return W, H, err

component_range = range(20, 60)  # 选择组件数范围
errors = []

for components in component_range:
    _, _, err = nmf_train_err(cocurrence_matrix, components, 1000, 1e-4)
    errors.append(err)

# 绘制重建误差与组件数的关系图
plt.plot(component_range, errors, marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Reconstruction Error')
plt.title('Reconstruction Error vs Number of Components')
plt.show()

# 可以看到components选取50左右是比较合理的
```



```
In [ ]: components = 50
        print('Start training')
        W, H = nmf_train(cocurrence_matrix, components, 1000, 1e-4)
        print('End training')
        print(W.shape)
        print('W:')
        print(W)
```

```
Start training
End training
(21, 50)
W:
[[6.59008984e-01 2.35168342e-01 6.27676144e-01 ... 9.74105159e-02
  1.36445235e-01 7.50672343e-01]
 [4.18934976e-01 1.15601575e-02 8.22213460e-01 ... 5.72653265e-02
  1.89431886e-01 1.56482326e+00]
 [7.13465199e-01 3.58166617e-04 8.39437561e-01 ... 1.52666501e-06
  6.90349309e-02 5.18110325e-01]
 ...
 [4.36996057e-01 8.04138707e-01 2.32912208e-01 ... 8.83574719e-01
  9.52373598e-01 8.81127058e-01]
 [4.52282485e-02 2.82156760e-03 2.07397871e-02 ... 2.88938455e-03
  8.36783917e-01 5.85895754e-01]
 [4.24468709e-01 7.44606433e-01 7.07123895e-01 ... 6.61217723e-01
  9.02927289e-02 8.95509413e-01]]
```

5、将上面的词向量存入pgvector，计算人物之间位向量相似度，找到每个人和他最相似的另一个人。

```
In [ ]: %sql CREATE EXTENSION IF NOT EXISTS vector;
```

```
 * postgresql://postgres:***@localhost:5432/hw
Done.
```

Out[ ]:
```
[]
```

In [ ]:
```
%%sql
CREATE TABLE IF NOT EXISTS person_vectors (
    "personId" INT PRIMARY KEY,
    "personName" TEXT,
    "vector" VECTOR(50) -- 每个词向量有50个维度
);
```

```
 * postgresql://postgres:***@localhost:5432/hw
Done.
```

Out[ ]:
```
[]
```

In [ ]:
```
def insert_vectors_to_db(W, person_df):
    conn = psycopg2.connect(dbname='hw', user='postgres', password='20020912', h
    cursor = conn.cursor()

    insert_query = "INSERT INTO person_vectors (\"personId\", \"personName\", \"

    for i, vector in enumerate(W):
        personId = int(person_df.loc[i, 'personId'])
        personName = person_df.loc[i, 'personName']
        vector_str = '[' + ','.join(map(str, vector.tolist())) + ']'
        cursor.execute(insert_query, (personId, personName, vector_str))

    conn.commit()
    cursor.close()
    conn.close()

insert_vectors_to_db(W, person_df)
```

In [ ]:
```
%%sql
WITH "similarity" AS (
    SELECT
        pv1."personId" AS "personId1",
        pv2."personId" AS "personId2",
        1 - (pv1."vector" <=> pv2."vector") AS "similarity" -- 使用向量余弦距离计
    FROM
        "person_vectors" pv1,
        "person_vectors" pv2
    WHERE
        pv1."personId" <> pv2."personId"
)
, "ranked_similarity" AS (
    SELECT
        "personId1",
        "personId2",
        "similarity",
        ROW_NUMBER() OVER (PARTITION BY "personId1" ORDER BY "similarity" DESC)
    FROM
        "similarity"
)
, "most_similar_pairs" AS (
    SELECT
        "personId1",
        "personId2",
        "similarity"
    FROM "ranked_similarity"
    WHERE rank = 1
)
SELECT
    sp."personId1",
    p1."personName" AS "personName1",
    sp."personId2",
    p2."personName" AS "personName2",
    sp."similarity"
FROM
    "most_similar_pairs" sp
JOIN
    person p1 ON sp."personId1" = p1."personId"
JOIN
    person p2 ON sp."personId2" = p2."personId";
```

```
 * postgresql://postgres:***@localhost:5432/hw
21 rows affected.
```

Out[ ]:

personId1	personName1	personId2	personName2	similarity
0	张三丰	5	殷梨亭	0.7681247199497572
1	张翠山	0	张三丰	0.7067103045327416
2	殷素素	1	张翠山	0.602189604025694
3	俞岱岩	5	殷梨亭	0.7536796634945123
4	俞莲舟	3	俞岱岩	0.7452326021813224
5	殷梨亭	0	张三丰	0.7681247199497572
6	纪晓芙	7	张无忌	0.6765070724797
7	张无忌	16	杨逍	0.7516164731049442
8	周芷若	16	杨逍	0.7469560877073133
9	赵敏	17	范遥	0.7537560419152932
10	殷离	17	范遥	0.6750395977067393
11	小昭	8	周芷若	0.6343184131480266
12	黛绮丝	20	胡青牛	0.3057606214621027
13	谢逊	8	周芷若	0.7409232802501331
14	朱长龄	15	朱九真	0.5769203893567139
15	朱九真	14	朱长龄	0.5769203893567139
16	杨逍	7	张无忌	0.7516164731049442
17	范遥	9	赵敏	0.7537560419152932
18	灭绝师太	7	张无忌	0.7145274276322152
19	何足道	0	张三丰	0.5139904676891912
20	胡青牛	7	张无忌	0.65579692234989

In [ ]:

```
#清空表项并删除表
%sql DELETE FROM yttlj
%sql DELETE FROM person
```

```
%sql DELETE FROM cocurrence
%sql DELETE FROM person_vectors
%sql DROP TABLE yttlj
%sql DROP TABLE person
%sql DROP TABLE cocurrence
%sql DROP TABLE person_vectors
```

```
 * postgresql://postgres:***@localhost:5432/hw
40 rows affected.
 * postgresql://postgres:***@localhost:5432/hw
21 rows affected.
 * postgresql://postgres:***@localhost:5432/hw
231 rows affected.
 * postgresql://postgres:***@localhost:5432/hw
21 rows affected.
 * postgresql://postgres:***@localhost:5432/hw
Done.
 * postgresql://postgres:***@localhost:5432/hw
Done.
 * postgresql://postgres:***@localhost:5432/hw
Done.
 * postgresql://postgres:***@localhost:5432/hw
Done.
```

```
Out[ ]:  []
```