# 网易云音乐用户数据的分析和可视化技术研究与实现

作者: 孟政元

指导教师: 王康平

日期: 2018年

学院: 计算机科学与技术学院

专业: 计算机科学与技术专业

## 摘要

网易云音乐中的用户大致可以分为两类, 一类是普通听众, 另一类是歌手, 音乐创作人这类发布音乐的人.在整个用户群体中, 存在着许多的关联关系. 这些用户可以随意的互相关注, 而每个用户又有着自己独特的特征, 比如性别, 地域, 年龄等等.用户之间相互关注的情况可以用一个很大的有向图来表示, 用户是有向图中的点, 每一个用户对另外一个用户的关注就是有向图的一条边. 这篇论文对部分用户信息(大概N用户)进行了采集,分析了用户的个体特征, 用户之间相互关注, 用户之间间接性的相互关联(基于floyd算法)等等在地域上的分布情况,并且基于分析结果进行回归预测, 从而估计整个用户群体的情况, 最后进行可视化展现, 使得结果直观明了.

## 数据的爬取

### 原理

跨站请求伪造 (CSRF), 伪造请求头 , 调用官方 API. 爬虫通过提供与正常请求相同的参数, 通过post方法向网易云的后台发起请求, 从而获取相应的数据.

### 调用官方后台API的具体方法

#### 访问方式

网易云音乐的后台api的url为`http://music.163.com/weapi/${api\_name}?csrf\_token=${token}, 通过发起http请求即可访问网易云音乐的后台api，但是后台的api需要加上正确的参数并对参数进行相应的处理，否则会被网易云音乐的后台判定为非法请求，直接屏蔽掉

#### 参数的处理

上面已经提到，在对网易云音乐的后台发起请求的时候，必须带上相应的参数并进行正确的处理，否则请求会被当做非法的而自己屏蔽掉。参数的具体处理方法如下(以获取歌曲的评论为例)：

1. csrf\_token

csrf(cross-site request forgery)即跨站请求伪造，这个参数主要是解决一些基于web的攻击，在请求的时候可以设为空字符串（假设客户端之前没有访问过网易云音乐的网站），但不能没有这个参数。

1. 其余的各个参数（offset，rid，limie）

网易云音乐的后台请求一律采取的是post方法，请求数据放在body部分。

将这些参数组成一个对象，如下所示，作为未加密处理的body部分。

const body =  
 {  
 offset: req.query.offset || 0,  
 rid: rid,  
 limit: req.query.limit || 20,  
 csrf\_token: ""  
 };

接下来通过相应的方法对请求进行加密处理：用一个16位长度的随机字符串作为加密的秘钥，然后按照下面代码所示的方法进行加密，最后返回一个对象，该对象具有两个属性：params是加密后的数据，encSecKey代表加密用的秘钥，用于服务器端对加密后的数据进行解密。

function Encrypt(obj)

{

const text = JSON.stringify(obj)

const secKey = createSecretKey(16)

const encText = aesEncrypt(aesEncrypt(text, nonce), secKey)

const encSecKey = rsaEncrypt(secKey, pubKey, modulus)

return {

params: encText,

encSecKey: encSecKey

}

}

然后通过post方法发起请求：url为该API对应的url，请求方法为post方法，body的内容就是上面返回的对象经过querystring序列化之后的返回结果。如下所示(注意：为了直观明了，代码有所删减，不能直接运行)：

function createWebAPIRequest

(

data,

callback,

errorcallback

)

{

const cryptoreq = Encrypt(data);

const options =

{

body: querystring.stringify

(

{

params: cryptoreq.params,

encSecKey: cryptoreq.encSecKey

}

),

proxy: proxy

};

request(options, function(error, res, body)

{

if (error)

{

console.error(error);

errorcallback(error);

}

else

{

callback(body, cookie);

}

});

}

createWebAPIRequest

(

"music.163.com",

`/weapi/v1/resource/comments/R\_SO\_4\_${rid}/?csrf\_token=`,

"POST",

data,

cookie,

music\_req =>

{

res.send(music\_req);

},

err => res.status(502).send(err.message)

)

注意：代码中用到了ES6标准中的模板字符串和箭头函数。

1. 模板字符串使用反引号 (` `) 来代替普通字符串中的用双引号和单引号。模板字符串可以包含特定语法(${expression})的占位符。占位符中的表达式和周围的文本会一起传递给一个默认函数，该函数负责将所有的部分连接起来，如果一个模板字符串由表达式开头，则该字符串被称为带标签的模板字符串，该表达式通常是一个函数，它会在模板字符串处理后被调用，在输出最终结果前，你都可以通过该函数来对模板字符串进行操作处理。在模版字符串内使用反引号（`）时，需要在它前面加转义符（\）。[1]
2. 箭头函数表达式的语法比函数表达式更短，并且不绑定自己的this，arguments，super或 new.target。这些函数表达式最适合用于非方法函数，并且它们不能用作构造函数。引入箭头函数有两个方面的作用：更简短的函数并且不绑定this。在箭头函数出现之前，每个新定义的函数都有它自己的 this值（在构造函数的情况下是一个新对象，在严格模式的函数调用中为 undefined，如果该函数被称为“对象方法”则为基础对象等）。This被证明是令人厌烦的面向对象风格的编程。箭头函数不会创建自己的this；它使用封闭执行上下文的this值。[2]
   * 1. *结果*

通过上面所述的请求方法，可以比较方便的通过调用网易云音乐后台API的方式进行数据的采集，免去了通过爬取网页来提取内容的麻烦，可以让爬虫以更快的速度进行数据的采集。但是，由于网易云音乐的后台设有严格的反爬虫机制，所以还要通过下面所说的方式来进行规避，防止爬虫被屏蔽。

* 1. 反爬虫策略

由于网易云的后台设置有反爬虫的机制, 所以不能爬取的太快, 否则IP会被封掉. 但是, 由于做分析需要的数据量较大,爬取的速度太慢无法再短时间内怕取到足够的数据. 因此必须采取一定的策略加快数据爬取的速度. 根据网易云的反爬虫机制, 单个IP地址一分钟最多发送20个请求. 因此, 这里采用多个代理并行爬取的方式. 同时, 在http请求头中, 随机化user-agent, 更好的把爬虫伪装成正常用户.

* 1. 代理程序

在搭建代理的时候, 由于我们的代理是为了隐藏爬虫, 所以必须使用http正向代理中的高匿代理, 否则会被网易云的后台发现异常. 这里使用的是自己使用Node.js编写的一个简单的匿名代理.

代理的原理是客户端先将请求发送到代理服务器, 由代理服务器向目标服务器发起请求, 并将请求结果发回客户端.

代码如下:

const PROXY\_PORT = 65535

const http = require('http')

const net = require('net')

const url = require('url')

function request(cReq, cRes)

{

const u = url.parse(cReq.url)

console.log(`[http.request] ${cReq.method} ${u.hostname}:${u.port || 80} ${u.path}`)

const options =

{

hostname : u.hostname,

port : u.port || 80,

path : u.path,

method : cReq.method,

headers : cReq.headers,

}

const pReq = http.request(options, function(pRes)

{

cRes.writeHead(pRes.statusCode, pRes.headers)

pRes.pipe(cRes);

}).on('error', function(e)

{

console.error(e)

cRes.end()

});

cReq.pipe(pReq)

}

function connect(cReq, cSock)

{

console.log(`[http.connect] ${cReq.url}`)

const u = url.parse('http://' + cReq.url);

const pSock = net.connect(u.port, u.hostname, function()

{

cSock.write('HTTP/1.1 200 Connection Established\r\n\r\n');

pSock.pipe(cSock);

}).on('error', function(e)

{

console.error(e)

cSock.end();

});

cSock.pipe(pSock);

}

http.createServer()

.on('request', request)

.on('connect', connect)

.on('error', function(err)

{

console.error(err)

})

.listen(PROXY\_PORT, function()

{

console.log(`proxy server online: http://localhost:${PROXY\_PORT}`)

});

* 1. Node.js基于事件循环的异步非阻塞并发机制

由于需要爬取的数据量较大, 传统的单线程同步程序耗时非常长, 所以必须采取并发机制. 这里采用的是Node.js基于事件循环的异步非阻塞并发机制. 在使用并发机制的时候, 需要使用互斥锁来确保程序有序运行.

## 数据的存储

* 1. 数据库

数据存储在自己搭建的MongoDB数据库中, 共五个collection, 总计N万条用户数据. MongoDB数据库是一种结构自由, 无schema的文档数据库, 便于存储这种对象化的用户数据.

## 数据的结构

* 1. 用户数据结构

1. user

{

"\_id":"5a51ba300604d374f28af6fb",

"profile":

{

"userId":363516402,

"gender":0,

"birthday":-2209017600000,

"city":220100

},

"\_\_processing":

{

"follow":false,

"followed":false,

"playlist":false,

"detail":false

},

"\_\_processed":

{

"follow":true,

"followed":true,

"playlist":false,

"detail":true

}

}

各个字段的含义如下:

1. \_id(数据库主键)

2. profile.userId(用户id)

3. profile.gender(用户性别, 0为男性, 1为女性)

4. profile.birthday(用户的出生日期)

5. profile.city(用户所在的城市)

6. \_\_processing(互斥锁, 用于并发)

7. \_\_processed(表示这个用户是否已经处理过了)

1. follow

用户的个人profile存储在follow这个collection中, 主要的结构如下:

{

"\_id":"5a4b488c954454ebecda66cb",

"from":"test",

"to":"test1",

"updatedAt":1515168662545

}

各个字段的含义如下:

1. \_id(数据库主键)

2. from(关注别人的那个人的userId)

3. to(被关注的那个人的userId)

4. updatedAt(这条记录的插入时间)

* 1. 待定

## 数据分析

* 1. 用户的总体分布情况

用饼状图来表示比例

4.1.1 性别分布情况

在表示用户个人信息的数据结构中, profile.gender表示用户的性别(0代表未知, 1代表男性, 2代表女性)

样本所包含的用户总数: 232433

查询性别未知的用户的数量: {'profile.gender': 0}, 结果为 39466

查询男性用户的的数量: {'profile.gender': 1}, 结果为 103419

查询女性用户的的数量: {'profile.gender': 2}, 结果为 89548

|  |  |  |
| --- | --- | --- |
| 性别不明 | 男性用户数量 | 女性用户数量 |
| 39466 | 102419 | 89548 |

采用d3.js可视化之后如图4.1.1所示

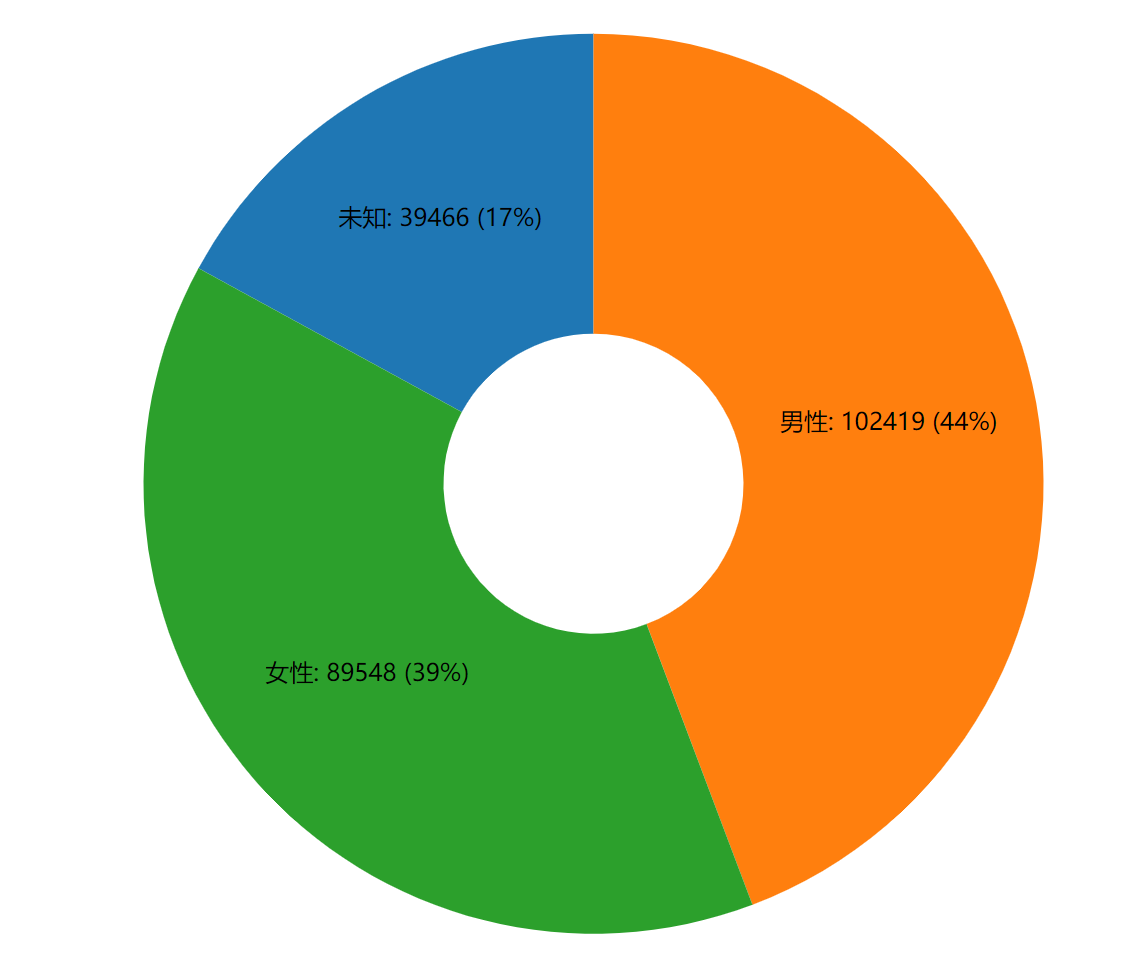


图4.1.1

4.2.2 年龄分布情况

查询70后的用户数量: {'profile.birthday': {$gte: 0, $lt: 315532800000}}

查询80后的用户数量: {'profile.birthday': {$gte: 315532800000, $lt: 631152000000}}

查询90后的用户数量: {'profile.birthday': {$gte: 631152000000, $lt: 946684800000}}

查询00后的用户数量: {'profile.birthday': {$gte: 946684800000, $lt: 1262304000000}}

查询10后的用户数量: {'profile.birthday': {$gte: 1262304000000}}

样本总人数: 88260

|  |  |
| --- | --- |
| 年龄段 | 人数 |
| 70后 | 849 |
| 80后 | 10873 |
| 90后 | 53504 |
| 00后 | 20019 |
| 10后 | 3015 |

采用d3.js可视化之后如图4.1.2所示

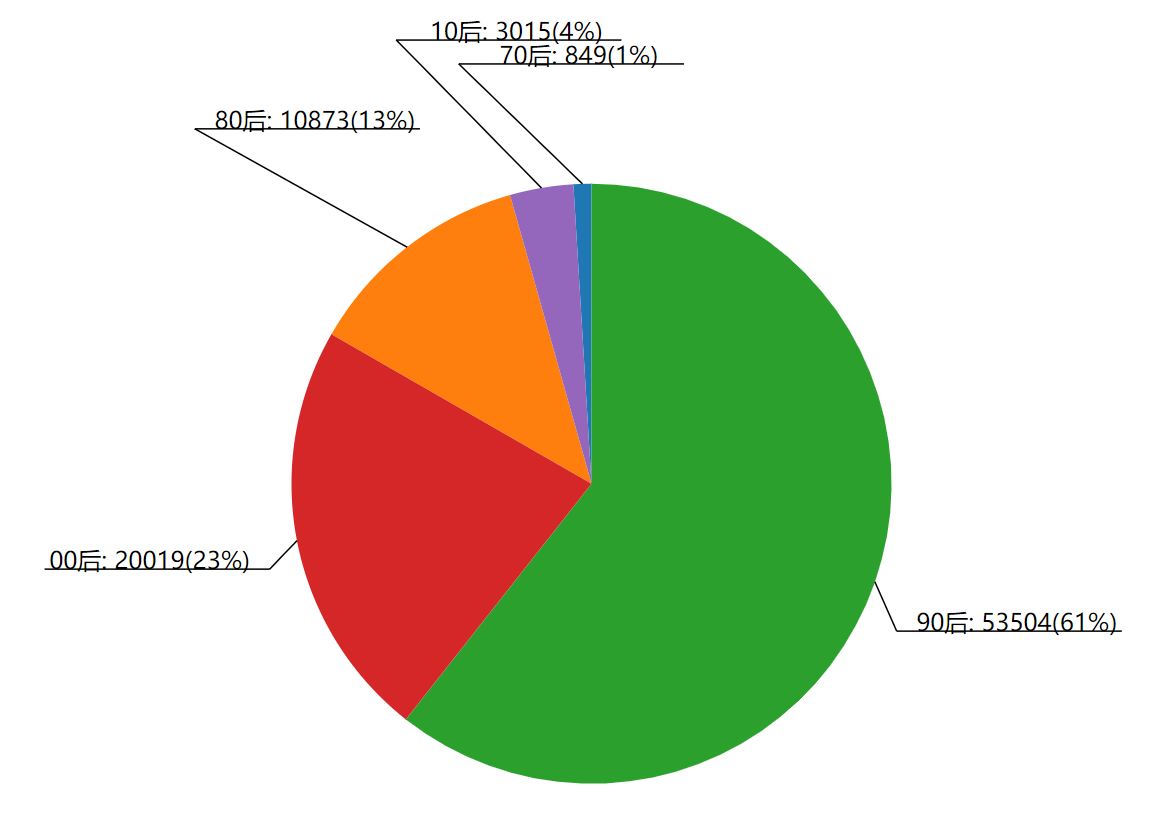


图4.1.2

* 1. 用户在不同省份的分布

用中国地图来表示不同省份的分布情况

4.2.1 数量分布情况

查询代码如下:

const mongodb = require('./mongodb')

const fs = require('fs')

const regions = JSON.parse(fs.readFileSync('./region.json').toString())

const path = require('path')

const $conn = mongodb.$conn

;(async function()

{

const db = await $conn

const user = db.collection('user')

const result = {}

for(const gb in regions)

{

const num = await user.find

(

{

'profile.province': Number(gb)

}

).count()

console.log(`| ${regions[gb]} | ${num} |`)

result[regions[gb]] = num

}

// console.log(result)

fs.writeFileSync(path.resolve(\_\_dirname, 'province.json'), JSON.stringify(result))

})()

结果如下表所示:

|  |  |
| --- | --- |
| 省份 | 用户数量 |
| 北京 | 11766 |
| 天津 | 2781 |
| 河北 | 5923 |
| 山西 | 3889 |
| 内蒙古 | 3526 |
| 辽宁 | 5265 |
| 吉林 | 2901 |
| 黑龙江 | 3730 |
| 上海 | 7317 |
| 江苏 | 14033 |
| 浙江 | 11514 |
| 安徽 | 7044 |
| 福建 | 5835 |
| 江西 | 5772 |
| 山东 | 11301 |
| 河南 | 10174 |
| 湖北 | 8216 |
| 湖南 | 7733 |
| 广东 | 23985 |
| 广西 | 4246 |
| 海南 | 1146 |
| 重庆 | 4387 |
| 四川 | 10242 |
| 贵州 | 2656 |
| 云南 | 5013 |
| 西藏 | 516 |
| 陕西 | 6274 |
| 甘肃 | 2798 |
| 青海 | 917 |
| 宁夏 | 969 |
| 新疆 | 5598 |
| 台湾 | 635 |
| 香港 | 832 |
| 澳门 | 196 |

对每个省份的用户人数使用D3.js可视化之后结果如图4.2.1.1所示:

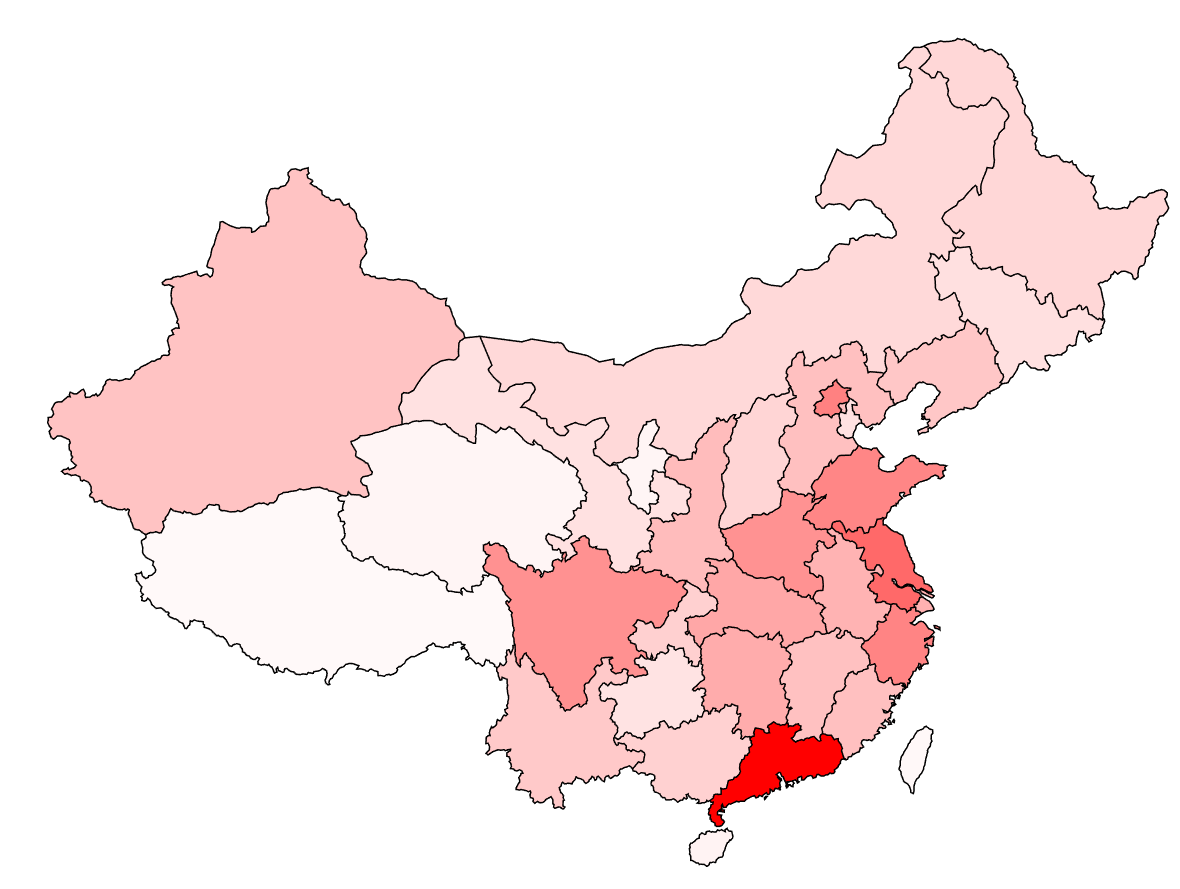
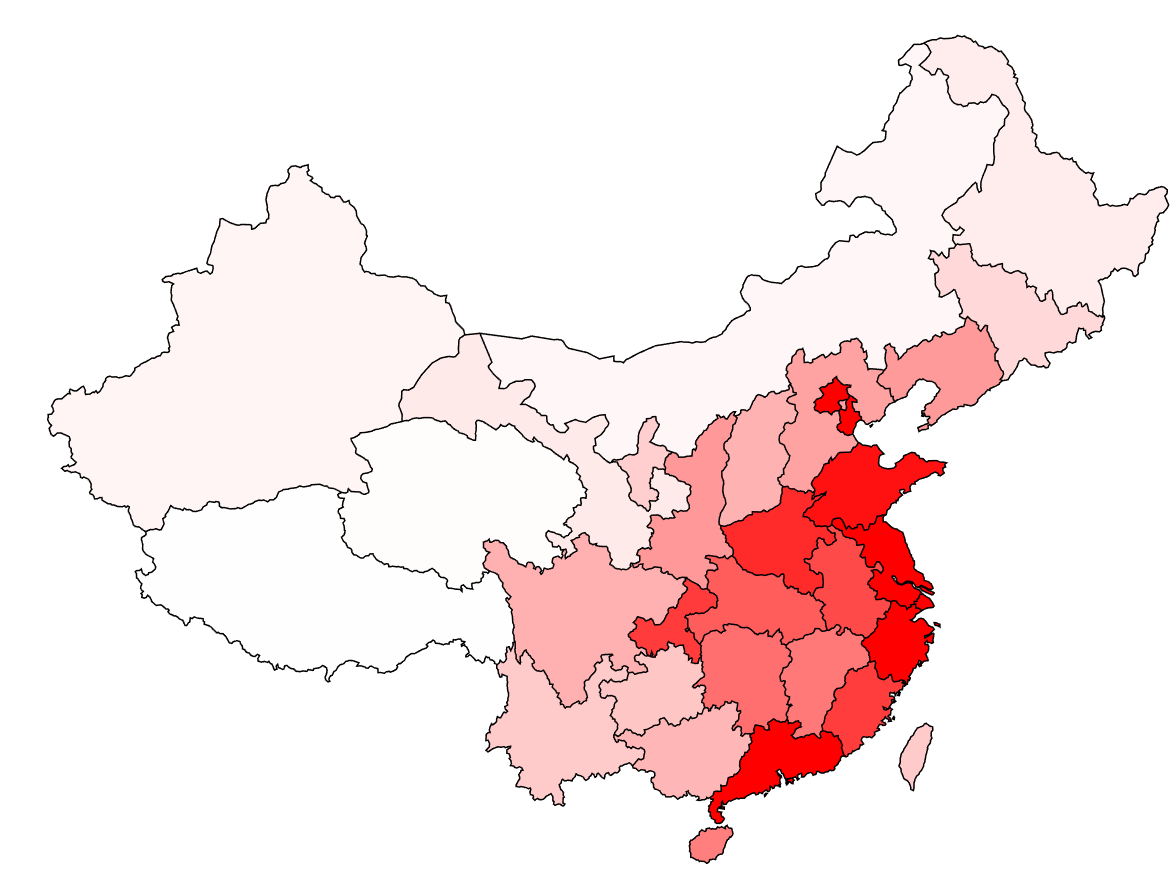


图4.2.1.1

对每个省份的用户密度(用户数 / 省份面积)使用D3.js可视化之后结果如图4.2.1.2所示:

图4.2.1.2

4.2.2 性别分布情况

在基于所有的样本进行分析之后, 得到的不同性别的用户在全国各个省份的分布情况.

代码如下:

const mongodb = require('./mongodb')

const fs = require('fs')

const regions = JSON.parse(fs.readFileSync('./region.json').toString())

const path = require('path')

const $conn = mongodb.$conn

;(async function()

{

const db = await $conn

const user = db.collection('user')

const result = {}

for(const gb in regions)

{

const female = await user.find

(

{

'profile.province': Number(gb),

'profile.gender': 2

}

).count()

const male = await user.find

(

{

'profile.province': Number(gb),

'profile.gender': 1

}

).count()

console.log(`| ${regions[gb]} | ${male} | ${female} |`)

result[regions[gb]] = [male, female]

}

fs.writeFileSync(path.resolve(\_\_dirname, 'sex.json'), JSON.stringify(result))

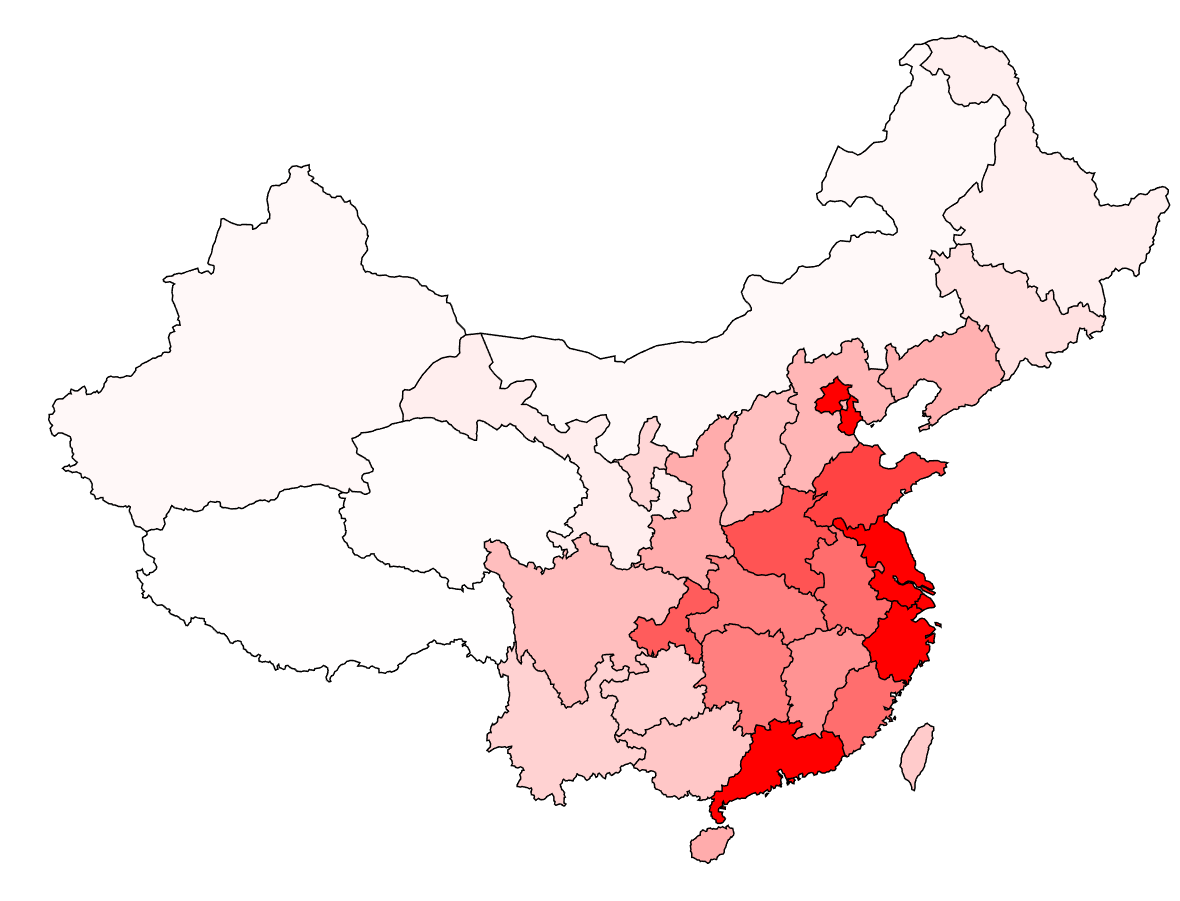
})()

结果如下:

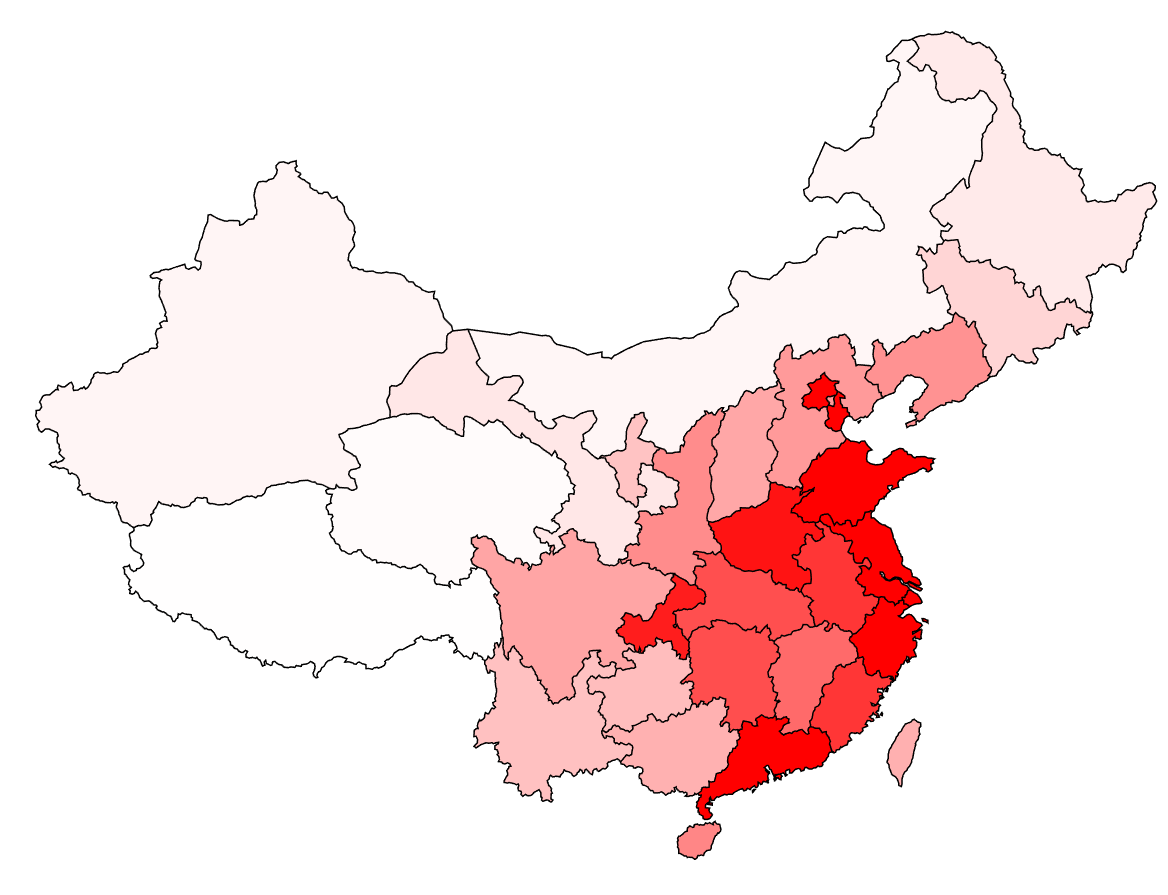
|  |  |  |
| --- | --- | --- |
| 省份 | 男性用户数量 | 女性用户数量 |
| 北京 | 5513 | 4058 |
| 天津 | 1249 | 1005 |
| 河北 | 2696 | 2197 |
| 山西 | 1688 | 1522 |
| 内蒙古 | 1581 | 1357 |
| 辽宁 | 2462 | 1951 |
| 吉林 | 1324 | 1092 |
| 黑龙江 | 1654 | 1467 |
| 上海 | 3223 | 2641 |
| 江苏 | 6319 | 5136 |
| 浙江 | 4848 | 4184 |
| 安徽 | 3233 | 2671 |
| 福建 | 2582 | 2065 |
| 江西 | 2547 | 2188 |
| 山东 | 5046 | 4204 |
| 河南 | 4523 | 3837 |
| 湖北 | 3729 | 3027 |
| 湖南 | 3314 | 3250 |
| 广东 | 11343 | 8195 |
| 广西 | 1956 | 1546 |
| 海南 | 574 | 383 |
| 重庆 | 1904 | 1725 |
| 四川 | 4558 | 3971 |
| 贵州 | 1195 | 1046 |
| 云南 | 2288 | 2005 |
| 西藏 | 232 | 200 |
| 陕西 | 2908 | 2433 |
| 甘肃 | 1274 | 1104 |
| 青海 | 438 | 357 |
| 宁夏 | 389 | 403 |
| 新疆 | 2696 | 2058 |
| 台湾 | 274 | 298 |
| 香港 | 382 | 358 |
| 澳门 | 96 | 76 |

使用d3.js进行可视化展示之后的结果如下:

1. 男性用户在全国的密度分布, 如图4.2.2.1所示

图4.2.2.1

1. 女性用户在全国的密度分布, 如图4.2.2.2所示

图4.2.2.2

3. 女性用户在所在的省份的用户中的比率在全国的分布, 如图4.2.2.3所示

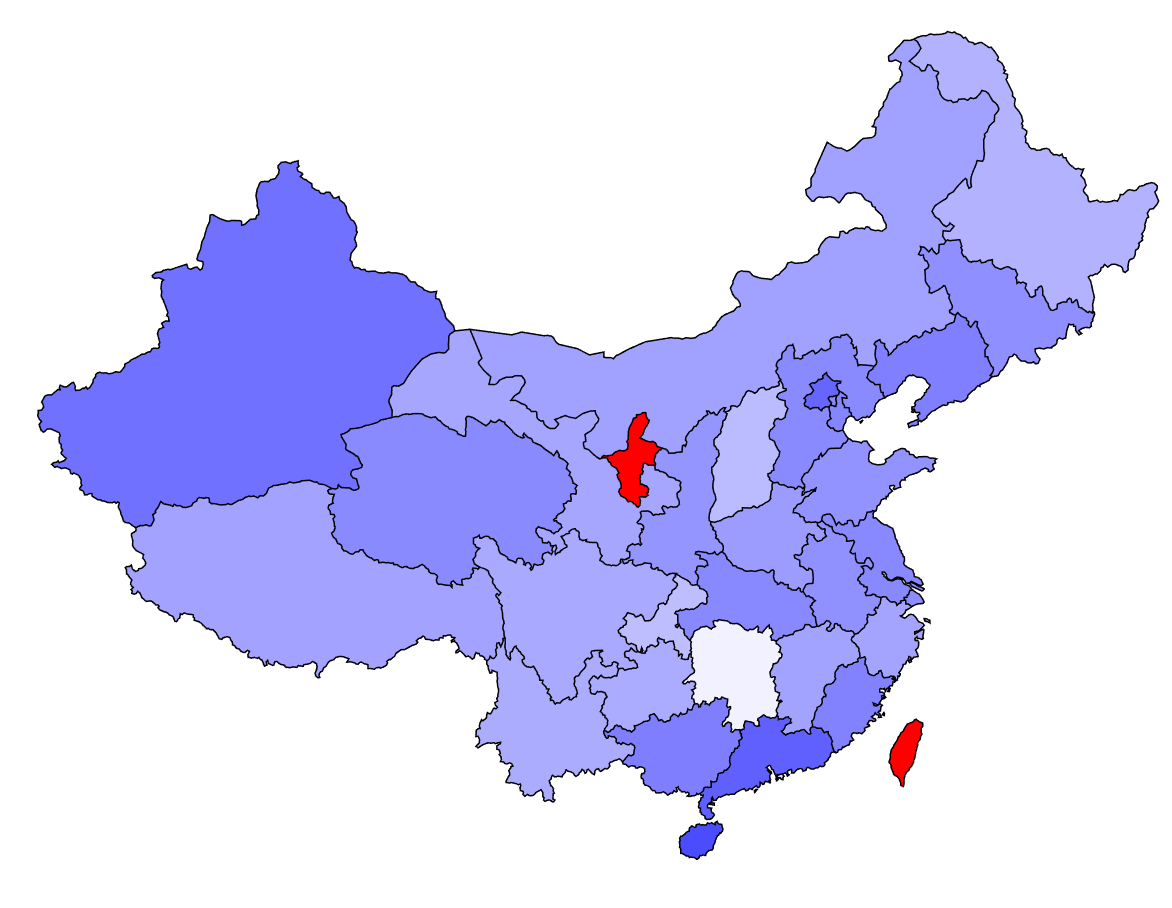


图4.2.2.3

4.2.3 年龄分布情况

查询代码如下:

const mongodb = require('./mongodb')

const fs = require('fs')

const regions = JSON.parse(fs.readFileSync('./region.json').toString())

const path = require('path')

const $conn = mongodb.$conn

;(async function()

{

const db = await $conn

const user = db.collection('user')

const result = {}

for(const gb in regions)

{

const age70 = await user.find

(

{

'profile.province': Number(gb),

'profile.birthday': {$gte: 0, $lt: 315532800000}

}

).count()

const age80 = await user.find

(

{

'profile.province': Number(gb),

'profile.birthday': {$gte: 315532800000, $lt: 631152000000}

}

).count()

const age90 = await user.find

(

{

'profile.province': Number(gb),

'profile.birthday': {$gte: 631152000000, $lt: 946684800000}

}

).count()

const age00 = await user.find

(

{

'profile.province': Number(gb),

'profile.birthday': {$gte: 946684800000, $lt: 1262304000000}

}

).count()

const age10 = await user.find

(

{

'profile.province': Number(gb),

'profile.birthday': {$gte: 1262304000000}

}

).count()

console.log(`| ${regions[gb]} | ${age70} | ${age80} | ${age90} | ${age90} | ${age10} |`)

result[regions[gb]] = [age70, age80, age90, age00, age10]

}

fs.writeFileSync(path.resolve(\_\_dirname, 'age.json'), JSON.stringify(result))

})()

结果如下:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 省份 | 70后 | 80后 | 90后 | 00后 | 10后 |
| 北京 | 132 | 1167 | 2849 | 2849 | 271 |
| 天津 | 11 | 153 | 756 | 756 | 26 |
| 河北 | 26 | 308 | 1388 | 1388 | 50 |
| 山西 | 12 | 174 | 948 | 948 | 23 |
| 内蒙古 | 7 | 153 | 891 | 891 | 21 |
| 辽宁 | 22 | 324 | 1392 | 1392 | 53 |
| 吉林 | 14 | 146 | 746 | 746 | 23 |
| 黑龙江 | 23 | 175 | 969 | 969 | 30 |
| 上海 | 58 | 553 | 1981 | 1981 | 127 |
| 江苏 | 57 | 656 | 3820 | 3820 | 139 |
| 浙江 | 42 | 578 | 2749 | 2749 | 145 |
| 安徽 | 15 | 245 | 1919 | 1919 | 65 |
| 福建 | 21 | 306 | 1266 | 1266 | 78 |
| 江西 | 9 | 199 | 1342 | 1342 | 64 |
| 山东 | 29 | 568 | 3034 | 3034 | 109 |
| 河南 | 22 | 409 | 2383 | 2383 | 93 |
| 湖北 | 24 | 391 | 2206 | 2206 | 117 |
| 湖南 | 24 | 301 | 1830 | 1830 | 98 |
| 广东 | 75 | 1246 | 5387 | 5387 | 301 |
| 广西 | 14 | 211 | 891 | 891 | 53 |
| 海南 | 3 | 54 | 254 | 254 | 17 |
| 重庆 | 18 | 202 | 1211 | 1211 | 54 |
| 四川 | 34 | 451 | 2686 | 2686 | 137 |
| 贵州 | 8 | 94 | 625 | 625 | 28 |
| 云南 | 11 | 189 | 1293 | 1293 | 61 |
| 西藏 | 2 | 28 | 152 | 152 | 7 |
| 陕西 | 16 | 275 | 1759 | 1759 | 39 |
| 甘肃 | 6 | 111 | 746 | 746 | 27 |
| 青海 | 3 | 28 | 199 | 199 | 10 |
| 宁夏 | 2 | 33 | 224 | 224 | 11 |
| 新疆 | 21 | 266 | 1582 | 1582 | 83 |
| 台湾 | 13 | 47 | 152 | 152 | 13 |
| 香港 | 5 | 99 | 286 | 286 | 38 |
| 澳门 | 1 | 35 | 65 | 65 | 9 |

使用d3.js进行可视化展示之后的结果如下:

1. 70后在全国的分布, 如图4.2.3.1所示

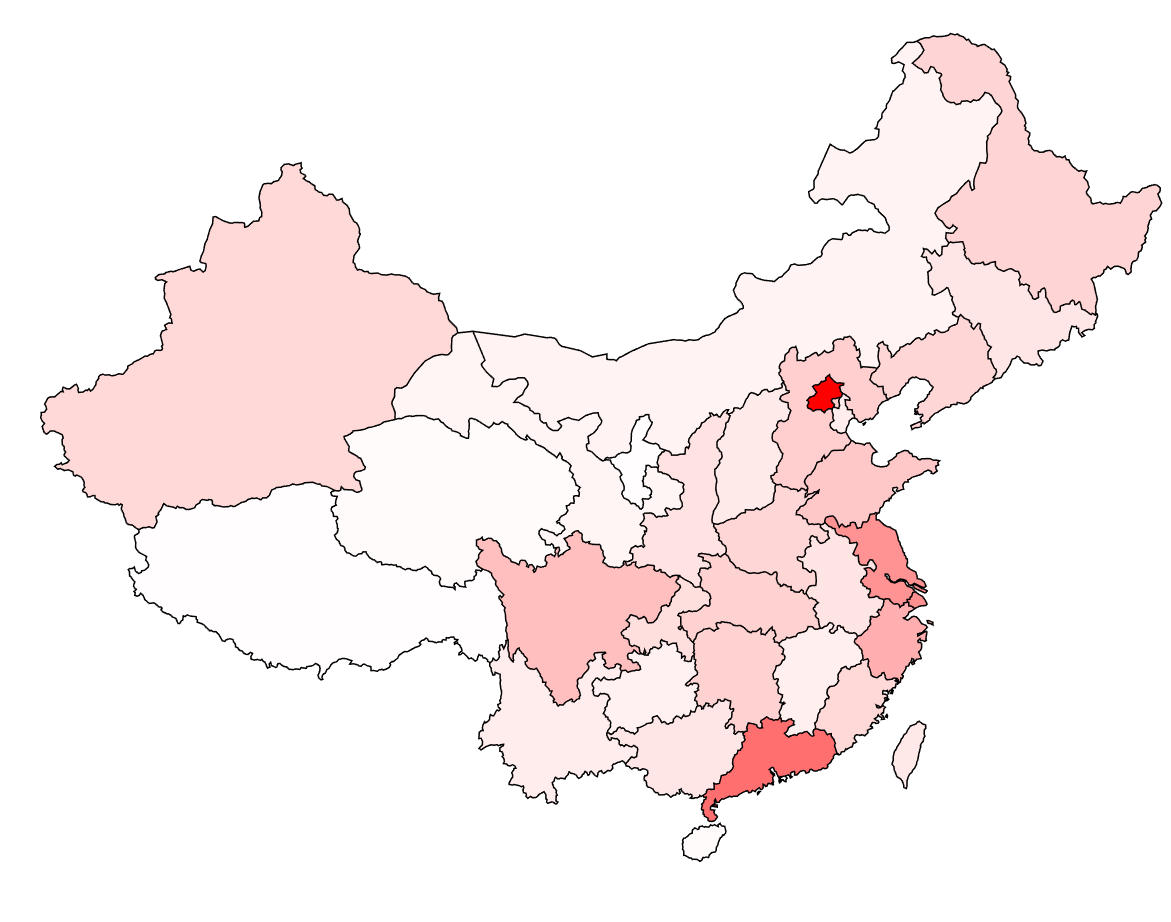


图4.2.3.1

2. 80后在全国的分布, 如图4.2.3.2所示

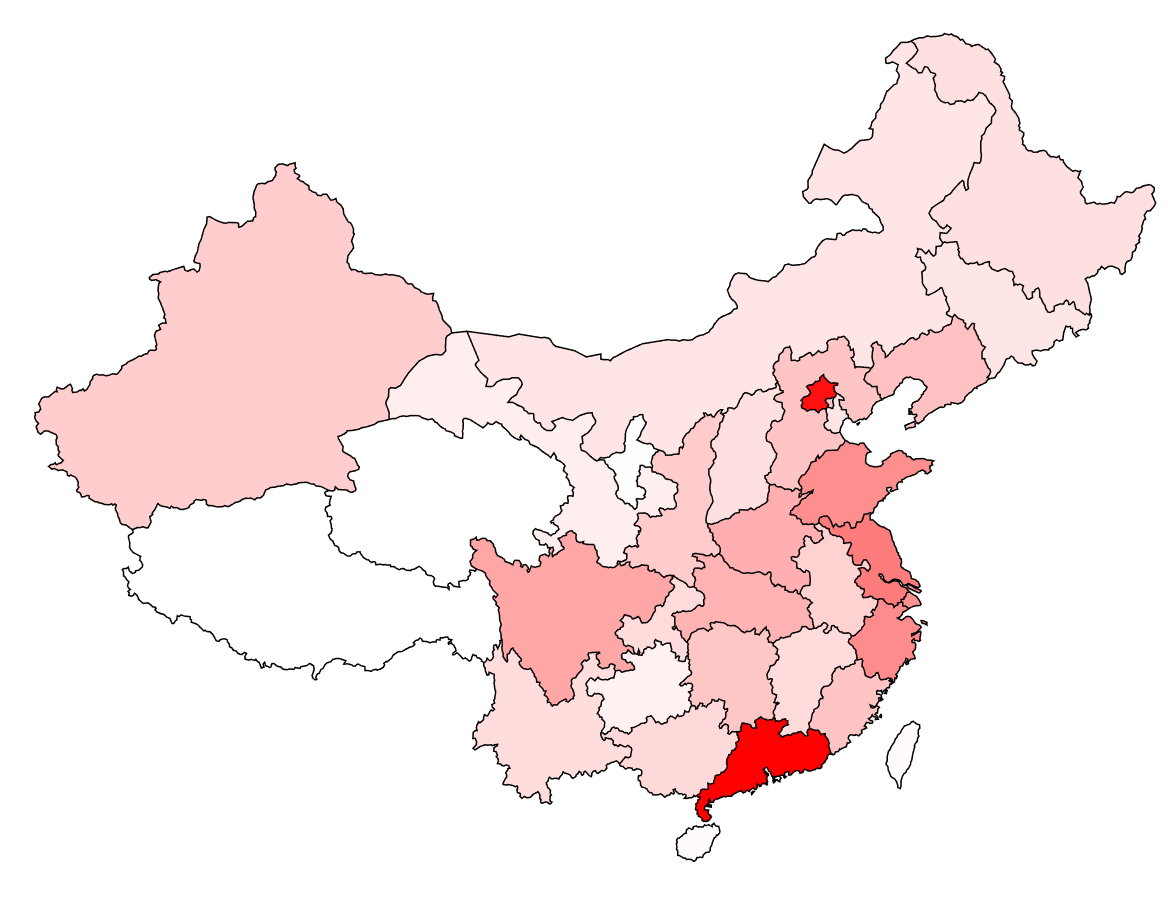


图4.2.3.2

1. 90后在全国的分布, 如图4.2.3.3所示

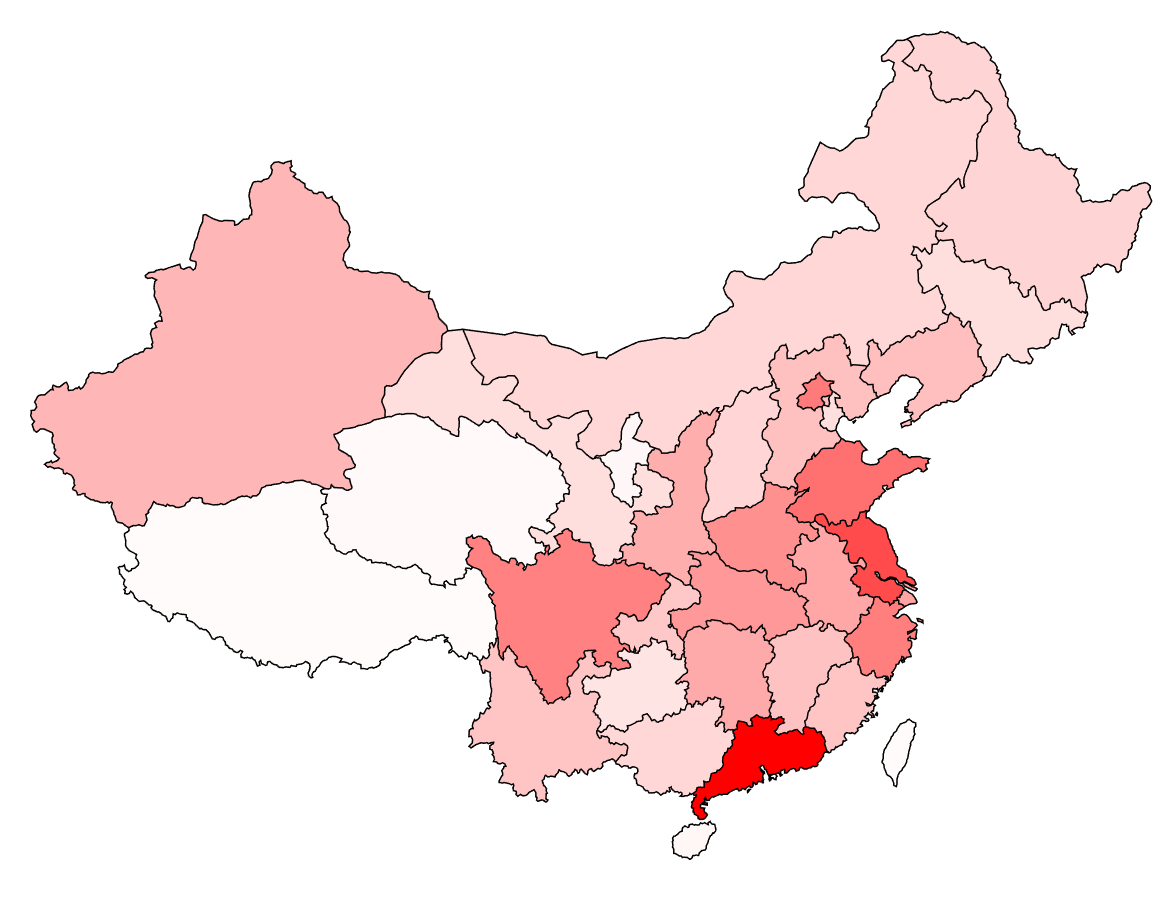


图4.2.3.3

1. 00后在全国的分布, 如图4.2.3.4所示

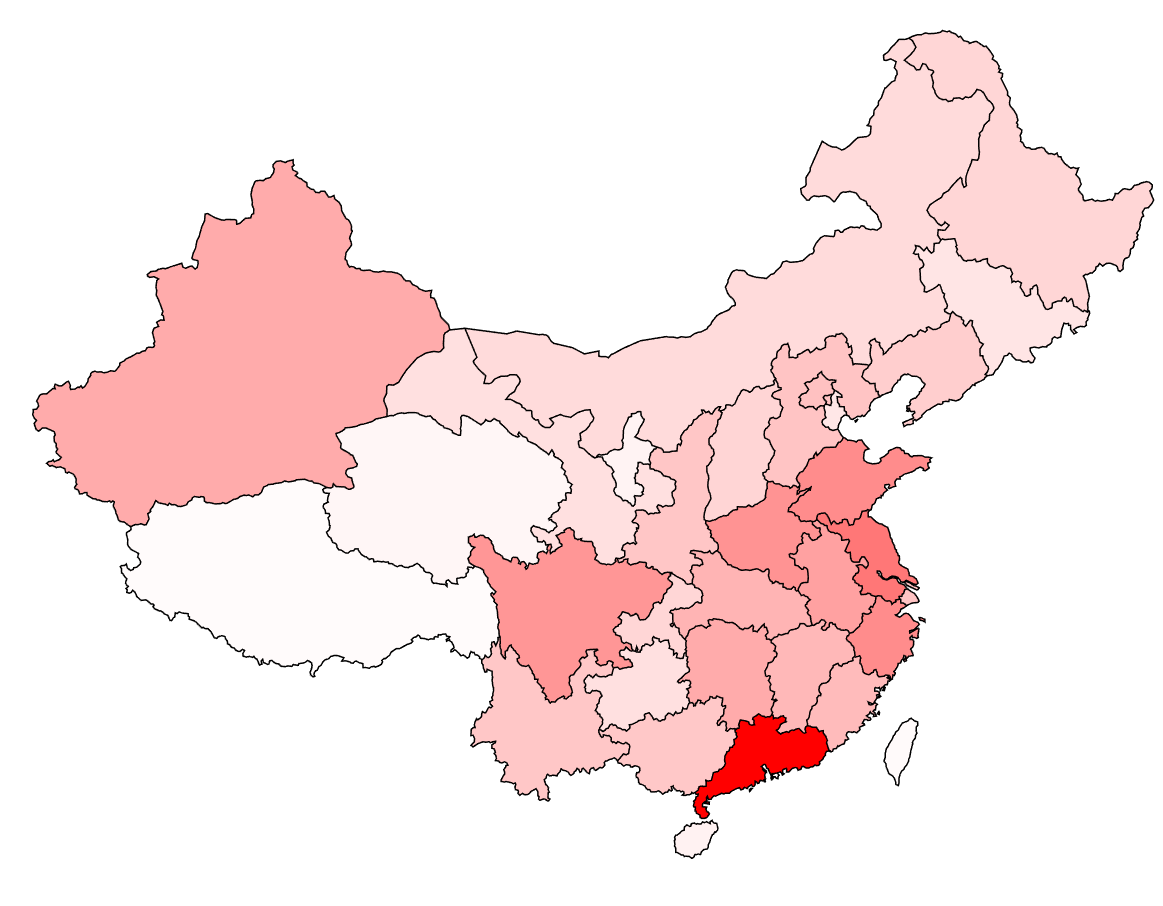


图4.2.3.4

5. 10后在全国的分布, 如图4.2.3.5所示

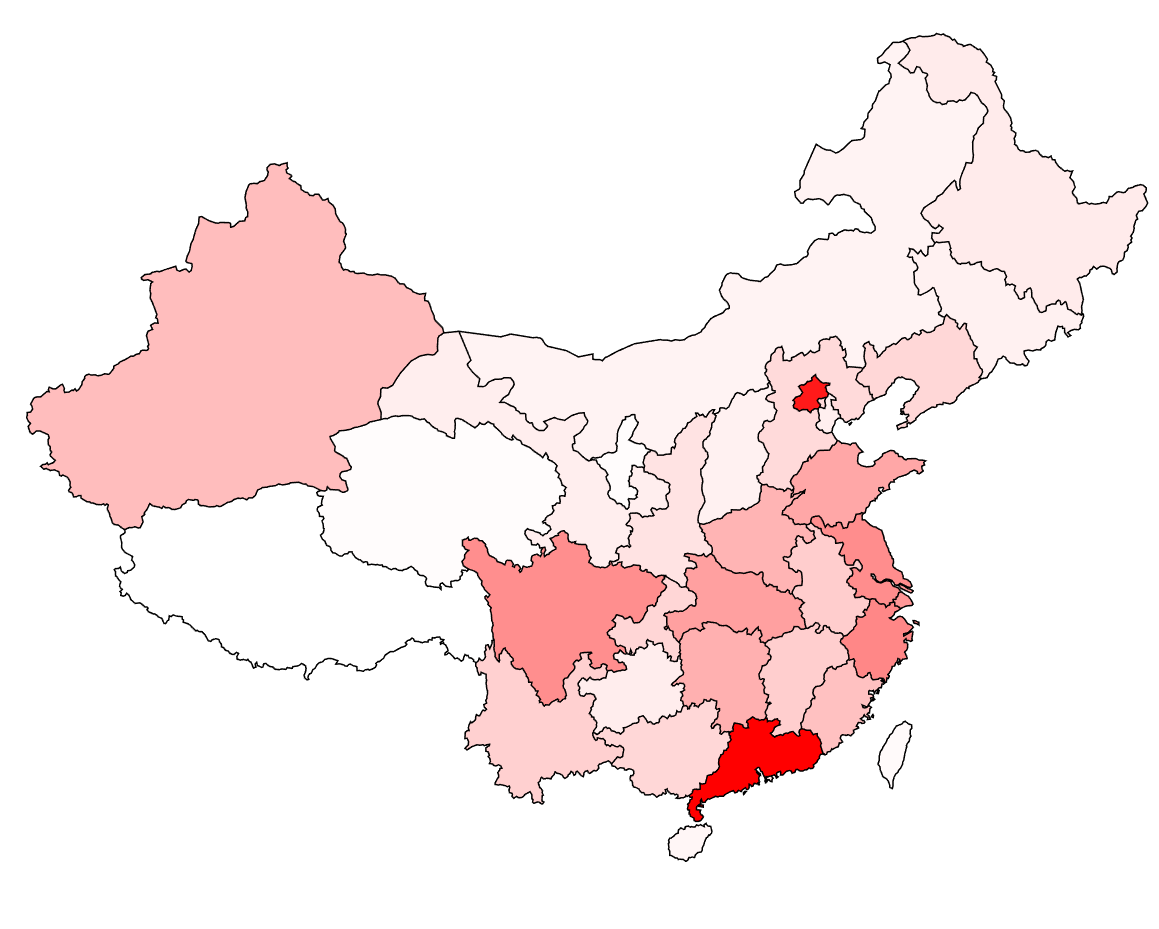


图4.2.3.5

* 1. 用户之间相互关注的情况的对比分析

使用弦图来进行展示

4.3.1 男女之间互相关注的数量的对比

查询男性用户对女性用户的关注数量: {'from.gender': 1, 'to.gender': 2}

查询男性用户对男性用户的关注数量: {'from.gender': 1, 'to.gender': 1}

查询女性用户对女性用户的关注数量: {'from.gender': 2, 'to.gender': 2}

查询女性用户对男性用户的关注数量: {'from.gender': 2, 'to.gender': 1}

具体代码如下:

const mongodb = require('./mongodb')

const $conn = mongodb.$conn

;(async function()

{

const $db = await $conn

const $follow\_sex = $db.collection('follow\_sex')

const male2male = await $follow\_sex.find

(

{

'from.gender': 1,

'to.gender': 1,

}

).count()

const male2female = await $follow\_sex.find

(

{

'from.gender': 1,

'to.gender': 2,

}

).count()

const female2male = await $follow\_sex.find

(

{

'from.gender': 2,

'to.gender': 1,

}

).count()

const female2female = await $follow\_sex.find

(

{

'from.gender': 2,

'to.gender': 2,

}

).count()

const matrix =

[

[male2male, male2female],

[female2male, female2female]

]

console.log(matrix)

})()

查询后, 结果如下表所示:

|  |  |  |
| --- | --- | --- |
| 关注 | 男性 | 女性 |
| 男性 | 1312487 | 927842 |
| 女性 | 633642 | 478432 |

使用d3.js可视化之后结果如图4.3.1所示:

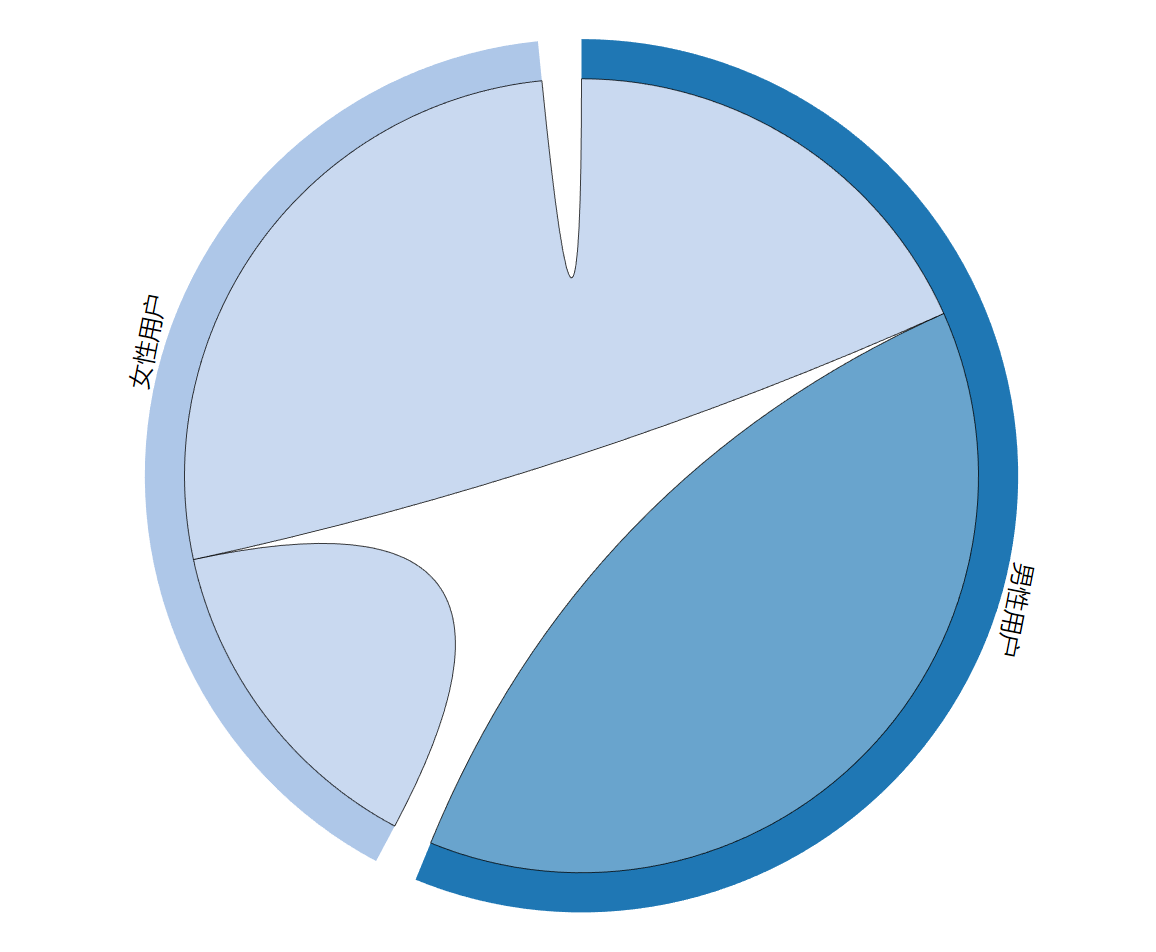


图4.3.1

4.3.2 不同省份的用户之间相互关注数量的对比

不同省份的用户之间相互关注数量的对比的查询代码具体如下:

const mongodb = require('./mongodb')

const fs = require('fs')

const regions = JSON.parse(fs.readFileSync('./region.json').toString())

const path = require('path')

const $conn = mongodb.$conn

;(async function()

{

const $db = await $conn

const $user = $db.collection('user')

const $follow\_sex = $db.collection('follow\_sex')

const rows = []

for(let name in regions)

{

const row = []

for(let name\_inner in regions)

{

const num = await $follow\_sex.find

(

{

'from.province': regions[name],

'to.province': regions[name\_inner],

}

).count()

row.push(num)

}

rows.push(row)

}

console.log(rows)

})()

查询结果较为复杂, 这里不列表展示.

使用d3.js可视化之后结果如图4.3.2所示:

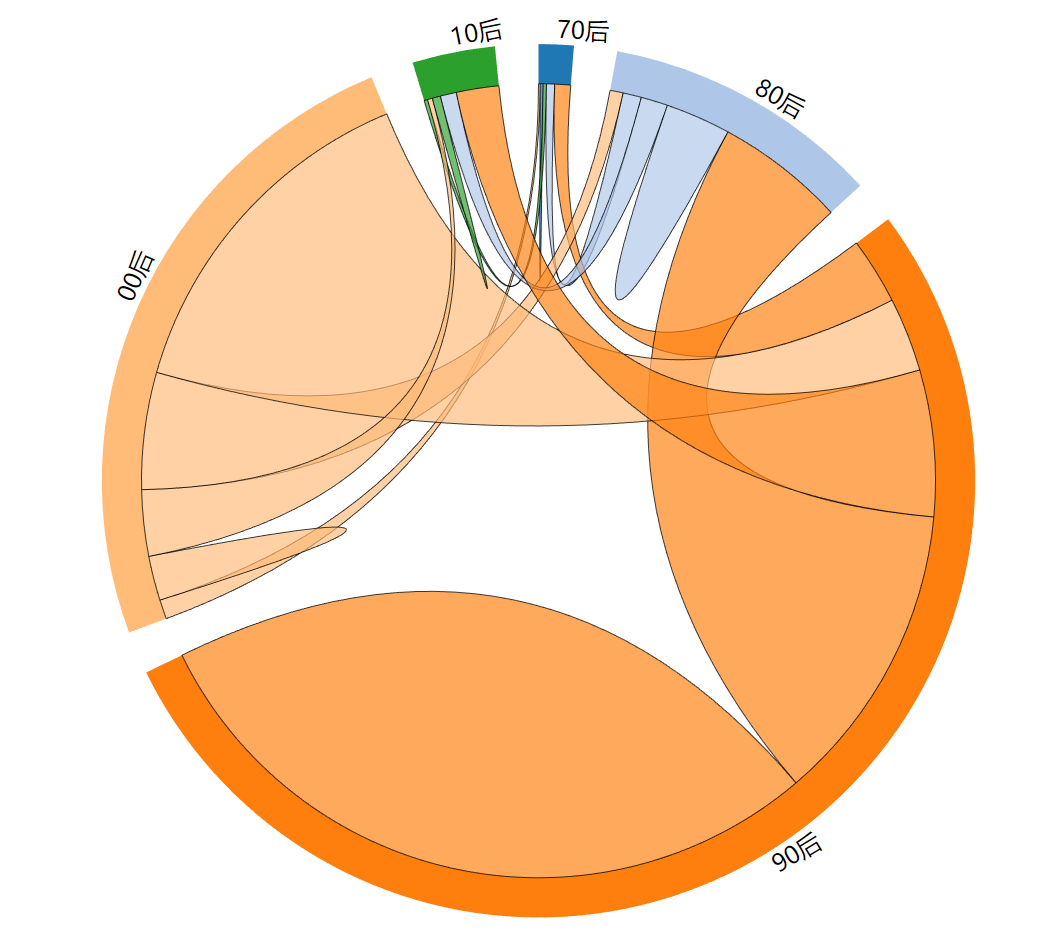


图4.3.2

4.3.3 不同年龄的用户的互相关注数量对比

查询不同年龄的用户互相关注数量对比的具体代码如下:

const mongodb = require('./mongodb')

const year70 = 0

const year80 = 315532800000

const year90 = 631152000000

const year00 = 946684800000

const year10 = 1262304000000

const year20 = 1562304000000

const years = [year70, year80, year90, year00, year10, year20]

const $conn = mongodb.$conn

;(async function()

{

const $db = await $conn

const $user = $db.collection('user')

const $follow\_sex = $db.collection('follow\_sex')

const rows = []

for(let index = 0; index < years.length - 1; index ++)

{

const row = []

for(let index\_inner = 0; index\_inner < years.length - 1; index\_inner ++)

{

const num = await $follow\_sex.find

(

{

'from.birthday':

{

$gte: years[index],

$lt: years[index + 1]

},

'to.birthday':

{

$gte: years[index\_inner],

$lt: years[index\_inner + 1]

},

}

).count()

row.push(num)

}

rows.push(row)

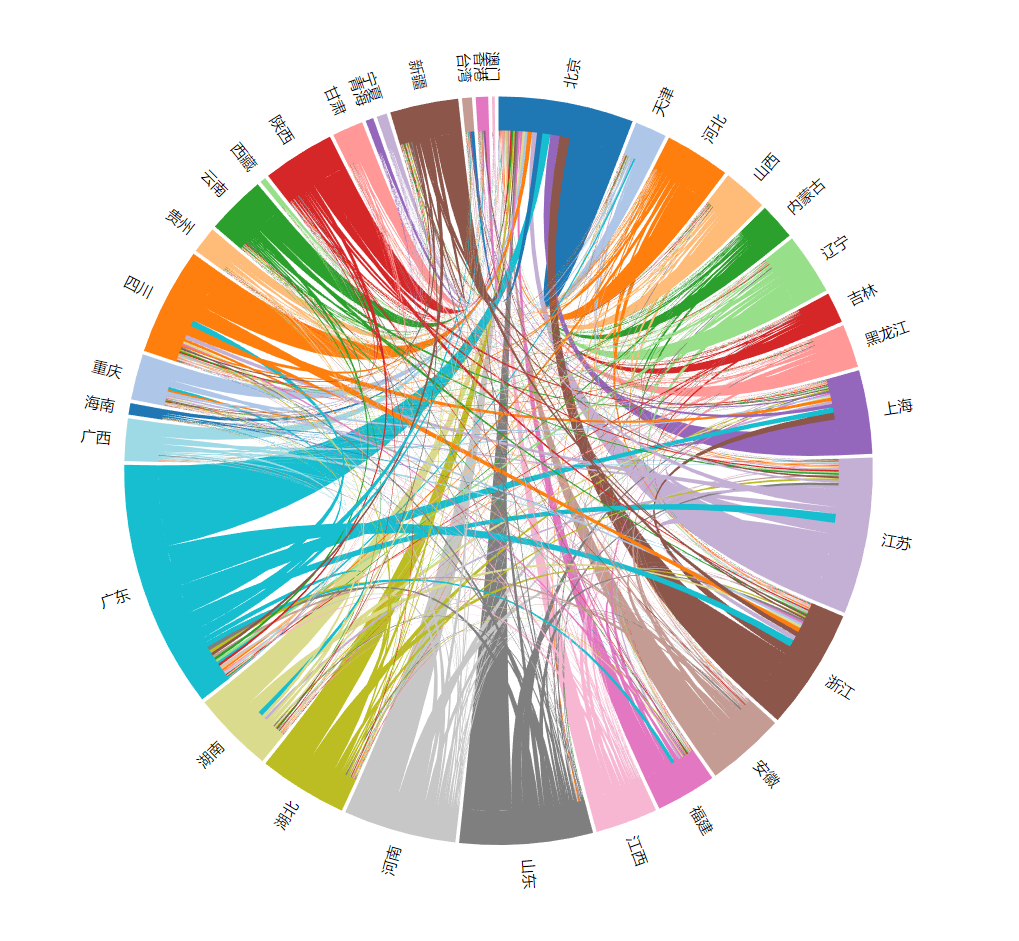
}

console.log(rows)

})()

查询结果较为复杂, 这里不列表展示.

使用d3.js可视化之后结果如图4.3.3所示:

图4.3.3

* 1. 采用floyd算法进行计算用户之间的距离

4.4.1 用有向图展示用户之间的关联关系

为了直观的展现用户之间的关联, 我们用有向图来描述这种关系.

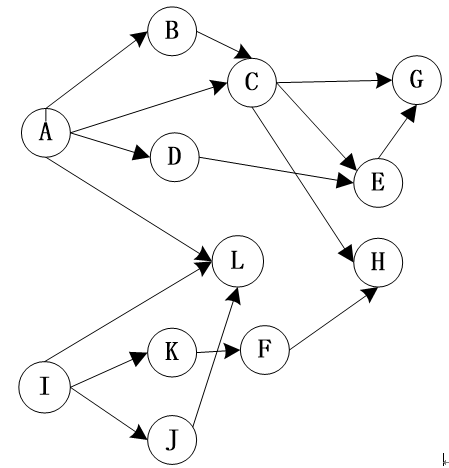
我们用有向图中的节点来代表用户实体, 用有向图中的边来代表用户之间的相互关注, 这样, 用户之间的关联关系就表示成了一个巨大的有向图(见图4.4.1.1), 我们下面的分析也是基于这个有向图来做的.

图4.4.1.1

在这里, 用户之间的距离是指在由用户节点和关注边所构成的有向图中, 某个用户节点与另外一个用户节点之间的最短距离.

4.4.2 计算用户的关联性

在代表用户关联的有向图中, 我们通过floyd算法来计算任意两点之间的最短距离

* + 1. 矩阵的存储

由于矩阵比较大, 存储在数据库中不方便进行运算, 运算时临时生成又比较耗时, 因此我们把矩阵序列化之后以JSON的形式存储在文件中, 运算的时候从文件中读入并且进行反序列化以及初始化.

* + 1. 矩阵的运算

从数据库中根据用户之间相互关注的信息, 生成表示有向图的矩阵, 下面所进行的运算都是基于该矩阵来进行的.

首先生成有向图的节点, 该过程中对有向图的边进行遍历是要去除重复的节点.

const mongodb = require('../mongodb')

const fs = require('fs')

const path = require('path')

const $conn = mongodb.$conn

;(async function()

{

const $db = await $conn

const $follow\_sex = $db.collection('follow\_sex')

const idSet = new Set()

const cursor = $follow\_sex.find().limit(1 \* 10000)

cursor.batchSize(1000)

let n = 0

while(await cursor.hasNext())

{

const e = await cursor.next()

idSet.add(e.from.userId)

idSet.add(e.to.userId)

n ++

console.log(n)

}

console.log(idSet.size)

const idList = []

let index = 0

idSet.forEach(async function(e)

{

const item =

{

index: index,

userId: e

}

idList.push(item)

index ++

console.log(index)

})

// await $vertex.insert(idList)

fs.writeFileSync(path.resolve(\_\_dirname, 'vertex.json'), JSON.stringify(idList))

})()

然后对节点进行处理, 便于编号和索引, 方便下面所要进行的运算

const fs = require('fs')

const path = require('path')

const vertex = JSON.parse(fs.readFileSync('vertex.json').toString())

const list = []

vertex.forEach(function(e)

{

console.log(e.index)

list.push(e.userId)

})

console.log(list)

fs.writeFileSync('vertex.list.json', JSON.stringify(list))

然后遍历有向图的边来生成有向图的矩阵, 代码如下:

const mongodb = require('../mongodb')

const fs = require('fs')

const $conn = mongodb.$conn

const vertex = JSON.parse(fs.readFileSync('vertex.list.json').toString())

const Inf = Infinity

const findIndex = function(userId)

{

return vertex.indexOf(userId)

}

const createMatrix = function(n, init = Inf)

{

const result = []

for(let i = 0; i < n; i ++)

{

const row = []

for(let i = 0; i < n; i ++)

{

row.push(init)

}

result.push(row)

}

return result

}

;(async function()

{

const $db = await $conn

const $follow\_sex = $db.collection('follow\_sex')

const cursor = $follow\_sex.find().limit(1 \* 10000)

cursor.batchSize(1000)

const matrix = createMatrix(vertex.length)

while(await cursor.hasNext())

{

const e = await cursor.next()

const row = findIndex(e.from.userId)

const col = findIndex(e.to.userId)

console.log(`[${row}][${col}] 1`)

try

{

matrix[row][col] = 1

}

catch(e)

{

}

}

console.log(matrix)

fs.writeFileSync('matrix.json', JSON.stringify(matrix))

})()

最后根据生面生成的有向图来运用floyd算法计算出距离矩阵和路径矩阵, 代码如下:

const Inf = Infinity

const fs = require('fs')

const matrix = JSON.parse(fs.readFileSync('matrix.json').toString())

let n = 0

for(let row = 0; row < matrix.length; row ++)

{

for(let col = 0; col < matrix.length; col ++)

{

if(matrix[row][col] === null)

{

matrix[row][col] = Inf

}

}

}

const numberOfNotInf = function(matrix)

{

let n = 0

for(let row = 0; row < matrix.length; row ++)

{

for(let col = 0; col < matrix.length; col ++)

{

if(matrix[row][col] !== Inf)

{

n ++

}

}

}

return n

}

console.log(numberOfNotInf(matrix))

// console.log(matrix)

const createSameSize = function(matrix, init = 0)

{

const result = []

for(let i = 0; i < matrix.length; i ++)

{

const row = []

for(let i = 0; i < matrix.length; i ++)

{

row.push(init)

}

result.push(row)

}

return result

}

// console.log(createSameSize(matrix))

const floyd = function(matrix)

{

const dist = createSameSize(matrix, Inf)

const path = createSameSize(matrix, Inf)

for (let i = 0; i < matrix.length; i ++)

{

for (let j = 0; j < matrix.length; j ++)

{

dist[i][j] = matrix[i][j]

path[i][j] = j // 顶点i"到"顶点j"的最短路径是经过顶点j, path[i][j] = i 也是一样的

}

}

for (let i = 0; i < matrix.length; i ++) // 循环遍历每个节点

{

console.log(`i: ${i}`)

for (let j = 0; j < matrix.length; j ++) // 每一行

{

for (let k = 0; k < matrix.length; k ++) // 每一列

{

const temp = (dist[i][k] === Inf || dist[k][j] === Inf) ? Inf : dist[i][k] + dist[k][j]

if(dist[i][j] > temp)

{

dist[i][j] = temp

path[i][j] = k

}

}

}

}

return [dist, path]

}

const result = floyd(matrix)

fs.writeFileSync('result.json', JSON.stringify(result))

console.log(numberOfNotInf(result[0]))

console.log(numberOfNotInf(result[1]))

至此, 有向图的运算已经完成, 我们下面的分析都是基于该运算结果来进行的.

* + 1. 对矩阵的运算结果进行分析

在前面矩阵的运算结果的基础上, 对矩阵进行分析.

代码如下:

const fs = require('fs')

const result = JSON.parse(fs.readFileSync('result.json').toString())

const dist = result[0]

const Inf = Infinity

for(let row = 0; row < dist.length; row ++)

{

for(let col = 0; col < dist.length; col ++)

{

if(dist[row][col] === null)

{

dist[row][col] = Inf

}

}

}

const numberOfNotInf = function(matrix)

{

let n = 0

for(let row = 0; row < matrix.length; row ++)

{

for(let col = 0; col < matrix.length; col ++)

{

if(matrix[row][col] !== Inf)

{

n ++

}

}

}

return n

}

const maxValueOfMatrix = function(matrix)

{

let max = 0

for(let row = 0; row < matrix.length; row ++)

{

for(let col = 0; col < matrix.length; col ++)

{

if(matrix[row][col] > max && matrix[row][col] !== Inf)

{

max = matrix[row][col]

}

}

}

return max

}

console.log(dist.length)

const nxn = dist.length \* dist.length

const noni = numberOfNotInf(dist)

const max = maxValueOfMatrix(dist)

console.log(nxn, noni, max)

有向图共有7586个节点, 10000条有向边

有7586 \* 7586 = 57547396对节点

可以联通的节点有2457173对

因此任意一个节点到另外一个节点有路径的概率为2457173 / 57547396 = 4.3%

最长的路径长度为7

由此可以看出网易云音乐用户之间的关联度极高.

## 数据的可视化

5.1 数据可视化概述

在近十年时间里,数据采集、存储和数据分析技术飞速发展,大大降低了数据储存和处理的成本,使得一个大数据时代逐渐展现在我们的面前。曾经,枯燥的数据统计和分析只是统计学家、数据分析师和科研学者们的专利,而大数据革新性的将海量数据处理变为可能,并且大幅降低了成本,使得越来越多跨专业学科的人投入到大数据的开发应用中来。大数据能够帮助人们更明智的制定决策、更清晰的传达理念。面对大数据瑰丽且深奥的面貌,如何才能让大型数据集变得亲切和易于理解,可视化无疑是最有效的途径。对大数据背景下的数据可视化应用展开研究,将有助于我们发展和创新数据可视化技术。

5.2 基于D3的数据可视化技术

D3 是 Data-Driven Documents（数据驱动文档）的缩 写，是一种在数据文档 JavaScript 库基础上开发的可视化 数据处理技术，其功能的实现机制是将数据库中的数据与 HTML、SVG、CSS 结合起来，数据信息与这些规则的结 合能够让原本的二维数据，在立体展示空间内形成一个缜 密的相互连接的立体数据体系，全面的体现出数据信息之 间的相互关系，保证数据信息的具象可视化效果。在这一 过程中的数据主要来源于作者，文档代表基于 Web 的文 档或者网页文档，而 D3 在这一过程中扮演的角色是整个 驱动程序的推动者，在数据的可视化过程中将数据和文档 元素结合在一起。 在具体的数据可视化实践中 D3.js 采用的是链式语 法，能够很好的实现对数据库中线性数据的提取和应用。[8]

5. 3 基于地图的数据可视化技术

5.3.1 更好的效果

基于地图的可视化能够带来很好的视觉效果, 尤其适合宽屏展示, 很适合人多的热闹气氛. 地图的可视化还能够带给人"高科技感", "未来感". 在本文中,为了更好的展示用户的各个特征在地理空间上的分布情况, 我们借助于D3, 使用基于地图的可视化技术来更加直观明了而又清晰的来展示.

5.3.2 地图数据基于GeoJSON格式

GeoJSON是用于描述地理空间信息的数据格式. GeoJSON不是一种新的格式, 其语法规则是符合JSON格式的, 只不过对其名称进行了规范, 专门用于表示地理信息. GeoJSON里的对象也是由名称/值对的集合构成, 名称总是字符串, 值可以是字符串, 数字, 布尔值, 对象, 数组, null.

GeoJSON的最外层是一个单独的对象(object). 这个对象可表示:

- 几何体(Geometry)

- 特征(Feature)

- 特征集合(FeatureCollection)

最外层的GeoJSON里可能包含有很多子对象, 每一个GeoJSON对象都有一个type属性, 表示对象的类型, type必须是下面之一:

- Point: 点

- MultiPoint: 多点

- LineString: 线

- MultiLineString: 多线

- Polygon: 面

- MultiPolygon: 多面

- GeometryCollection: 几何体集合

- Feature: 特征

- FeatureCollection: 特征集合

举例如下:

点对象:

{

"type": "Point",

"coordinates": [-105, 39]

}

线对象:

{

"type": "LineString",

"coordinates": [[-105, 39], [-107, 38]]

}

面对象:

{

"type": "Polygon",

"coordinates": [[[30, 0], [31, 0], [31, 5], [30, 5], [30, 0]]]

}

## 结论

在本文中, 通过对网易云音乐用户数据的爬取, 处理, 分析以及可视化, 同时借助于比较新颖的基于地图的可视化技术, 比较直观明了的展示了网易云音乐用户在几个常见的维度以及在地理空间上的一些特征.

主要的结论有:

1. 男性用户比较多, 但是与女性用户的比例差距并不大(男女比例为: 44:39)

2. 90后用户占据了用户的大多数(61%), 其次是00后(23%), 再其次是80后(13%), 10后和70后只占据了很少的一部分, 因此可以看出网易云音乐的用户群体非常的年轻化.

3. 从地理空间的角度来看, 无论是用户总人数, 不同年龄段的用户还是男女用户人数在沿海的省份的人数和密度都比内陆地区要高

4. 从地理空间的角度来看, 除了台湾和宁夏之外, 所有的身份都是男性用户较多, 女性用户较少

5. 从用户互相关注的的维度来看, 最为活跃的是90后的用户.

与此同时, 本文通过对网易云音乐部分用户之间相互关注的情况的分析计算, 可以看出网易云音乐的用户之间具有极其强烈的关联性.

##### 参考文献

[1] https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/template\_strings [2] https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Functions/Arrow\_functions

[7] 涂聪. 大数据时代背景下的数据可视化应用研究[J]. 电子制作, 2013, 47(5x):118-118. [8] 赵聪. 可视化库D3.js的应用研究[J]. 信息技术与信息化, 2015(2):107-109.