数据科学 1 体会

数据产生：

网站：用户每次点击
手机：位置和速度
智能手表、手环：心率、行动、饮食、睡眠
智能汽车：驾驶习惯
智能家居：生活习惯

数据科学家

从混乱数据中理出价值的人

案例：寻找关键联系人

根据用户网络关系数据识别关键联系人

用户列表

```
users = [
    { "id": 0, "name": "Hero" },
    { "id": 1, "name": "Dunn" },
    { "id": 2, "name": "Sue" },
    { "id": 3, "name": "Chi" },
    { "id": 4, "name": "Thor" },
    { "id": 5, "name": "Clive" },
    { "id": 6, "name": "Hicks" },
    { "id": 7, "name": "Devin" },
    { "id": 8, "name": "Kate" },
    { "id": 9, "name": "Klein" },
    { "id": 10, "name": "Jen" }
]
```

用户好友关系

```
friendships = [(0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (3, 4),
               (4, 5), (5, 6), (5, 7), (6, 8), (7, 8), (8, 9)]
```

为每个用户创建朋友列表

```
for user in users:
    user["friends"] = []
```

填充好友数据

```
for i, j in friendships:
    users[i]["friends"].append(users[j])
    users[j]["friends"].append(users[i])
```

.

问题： 平均朋友联系数是多少?
答： 全部联系数除以用户个数

```
def number_of_friends(user):
    return len(user["friends"])

total_connections = sum(number_of_friends(user)
                        for user in users) # 24

num_users = len(users)
avg_connections = total_connections / num_users # 2.4
```

按朋友数多少排序

```
# 取(user_id, number_of_friends)
num_friends_by_id = [(user['id'], number_of_friends(user)) for user in users]

sorted(num_friends_by_id, key=lambda item: item[1], reverse=True)
```

案例: 你可能知道的人

找朋友的朋友

```
def friends_of_friend_ids_bad(user):
    return [foaf["id"]
            for friend in user["friends"]
            for foaf in friend["friends"]]
```

查找共同的朋友

```
from collections import Counter # not loaded by default

def not_the_same(user, other_user):
    return user["id"] != other_user["id"]

def not_friends(user, other_user):
    return all(not_the_same(friend, other_user)
               for friend in user["friends"])

def friends_of_friend_ids(user):
    return Counter(foaf["id"]
                   for friend in user["friends"]
                   for foaf in friend["friends"]
                   if not_the_same(user, foaf)
                   and not_friends(user, foaf))

print(friends_of_friend_ids(users[3]))
```

找共同兴趣的人

```
interests = [
    (0, "Hadoop"), (0, "Big Data"), (0, "HBase"), (0, "Java"),
    (0, "Spark"), (0, "Storm"), (0, "Cassandra"),
    (1, "NoSQL"), (1, "MongoDB"), (1, "Cassandra"), (1, "HBase"),
    (1, "Postgres"), (2, "Python"), (2, "scikit-learn"), (2, "scipy"),
    (2, "numpy"), (2, "statsmodels"), (2, "pandas"), (3, "R"), (3, "Python"),
    (3, "statistics"), (3, "regression"), (3, "probability"),
    (4, "machine learning"), (4, "regression"), (4, "decision trees"),
```

```
    (4, "libsvm"), (5, "Python"), (5, "R"), (5, "Java"), (5, "C++"),
    (5, "Haskell"), (5, "programming languages"), (6, "statistics"),
    (6, "probability"), (6, "mathematics"), (6, "theory"),
    (7, "machine learning"), (7, "scikit-learn"), (7, "Mahout"),
    (7, "neural networks"), (8, "neural networks"), (8, "deep learning"),
    (8, "Big Data"), (8, "artificial intelligence"), (9, "Hadoop"),
    (9, "Java"), (9, "MapReduce"), (9, "Big Data")
]
```

```
def data_scientists_who_like(target_interest):
    return [user_id
            for user_id, user_interest in interests
            if user_interest == target_interest]
```

每次搜索都要遍历列表，性能差，建立一个字典

```
from collections import defaultdict

user_ids_by_interest = defaultdict(list)

for user_id, interest in interests:
    user_ids_by_interest[interest].append(user_id)

interests_by_user_id = defaultdict(list)

for user_id, interest in interests:
    interests_by_user_id[user_id].append(interest)
```

找与指定用户爱好最多相似的用户

```
def most_common_interests_with(user_id):
    return Counter(interested_user_id
        for interest in interests_by_user_id[user_id]
        for interested_user_id in user_ids_by_interest[interest]
        if interested_user_id != user_id)
```

案例：工资与工作年限

```
salaries_and_tenures = [(83000, 8.7), (88000, 8.1),
                        (48000, 0.7), (76000, 6),
                        (69000, 6.5), (76000, 7.5),
                        (60000, 2.5), (83000, 10),
                        (48000, 1.9), (63000, 4.2)]
```

绘图

```
def make_chart_salaries_by_tenure():
    tenures = [tenure for salary, tenure in salaries_and_tenures]
    salaries = [salary for salary, tenure in salaries_and_tenures]
    plt.scatter(tenures, salaries)
    plt.xlabel("Years Experience")
    plt.ylabel("Salary")
    plt.show()
```

按工作年线算平均收入

```
salary_by_tenure = defaultdict(list)

for salary, tenure in salaries_and_tenures:
    salary_by_tenure[tenure].append(salary)

average_salary_by_tenure = {
    tenure : sum(salaries) / len(salaries)
    for tenure, salaries in salary_by_tenure.items()
}
```

分组后计算

```
def tenure_bucket(tenure):
```

```
    if tenure < 2: return "less than two"
    elif tenure < 5: return "between two and five"
    else: return "more than five"

salary_by_tenure_bucket = defaultdict(list)

for salary, tenure in salaries_and_tenures:
    bucket = tenure_bucket(tenure)
    salary_by_tenure_bucket[bucket].append(salary)

average_salary_by_bucket = {
  tenure_bucket : sum(salaries) / len(salaries)
  for tenure_bucket, salaries in salary_by_tenure_bucket.items()
}
```

案例：兴趣主题

简单方法：数兴趣词汇个数

```
words_and_counts = Counter(word
                           for user, interest in interests
                           for word in interest.lower().split())

for word, count in words_and_counts.most_common():
    if count > 1:
        print(word, count)
```