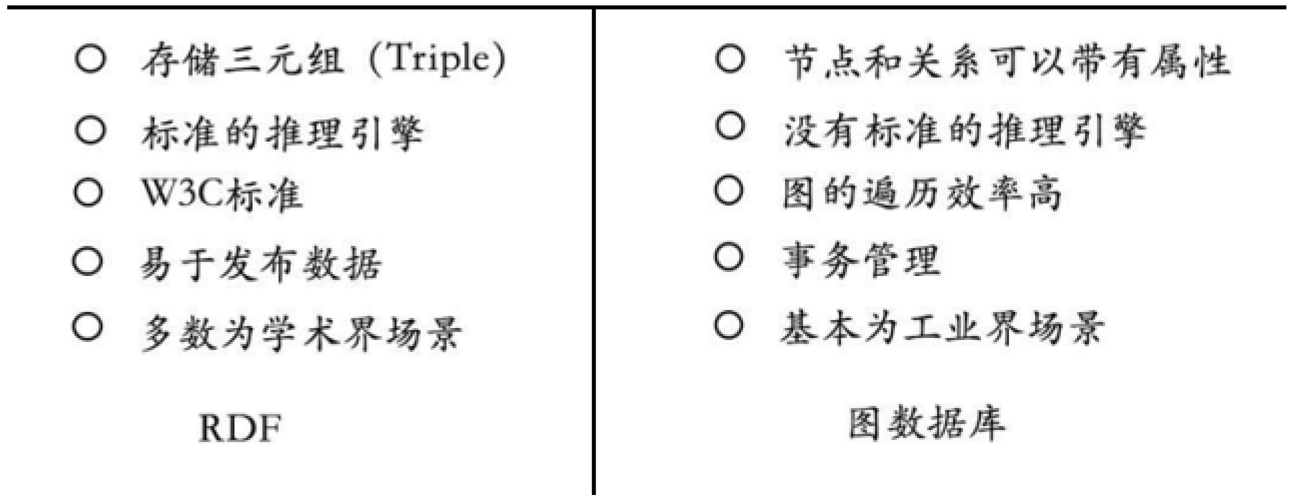
小型金融知识图谱搭建示范

知识图谱存储方式

存储方式

1. 基于RDF的存储
2. 基于图数据库的存储

方式对比

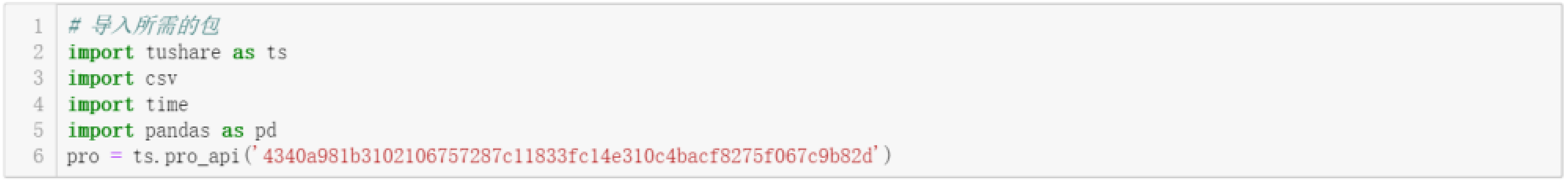


○ 存储三元组（Triple）	○ 节点和关系可以带有属性
○ 标准的推理引擎	○ 没有标准的推理引擎
○ W3C标准	○ 图的遍历效率高
○ 易于发布数据	○ 事务管理
○ 多数为学术界场景	○ 基本为工业界场景
RDF	图数据库

知识图谱构建流程

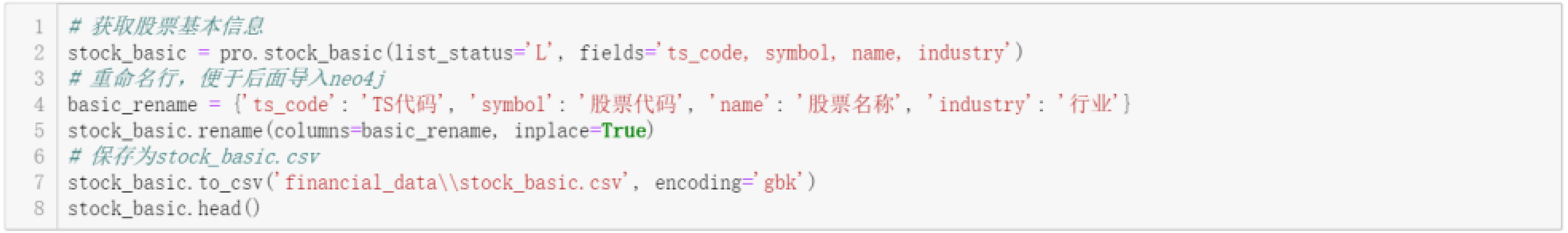
1. 数据获取
   - 股票基本信息
   - 股票Top10股东信息
   - 股票概念信息
   - 股票公告信息
   - 财经新闻信息（该数据集已获取但需进一步处理，未存入图数据库）
   - 概念信息
   - 股票价格信息
2. 数据预处理
   - 基本信息存在空值
   - 股东信息存在重复数据
   - CSV文件格式更改为UTF-8格式
   - 计算股票对数收益
   - 保留股票价格交易日为242（众数）&计算皮尔逊相关系数
3. 数据存储
   - 明确实体&关系
   - 使用py2neo交互neo4j创建节点和关系
4. 数据可视化查询
   - 基于Crypher语言
5. 相关应用
   - 中心度算法(Centralities)
   - 社区检测算法(Community detection)
   - 路径搜索算法(Path finding)
   - 相似性算法(Similarity)
   - 链接预测(Link Prediction)

数据获取

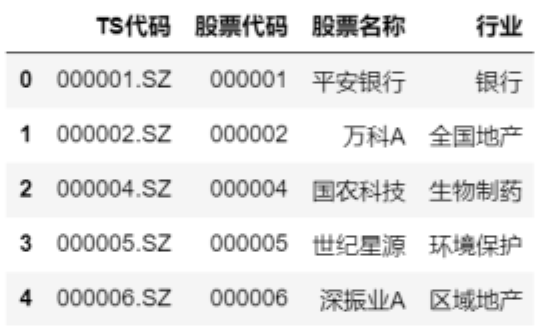


```python
# 导入所需的包
import tushare as ts
import csv
import time
import pandas as pd
pro = ts.pro_api('4340a981b3102106757287c11833fc14e310c4bacf8275f067c9b82d')
```

1.1 股票基本信息

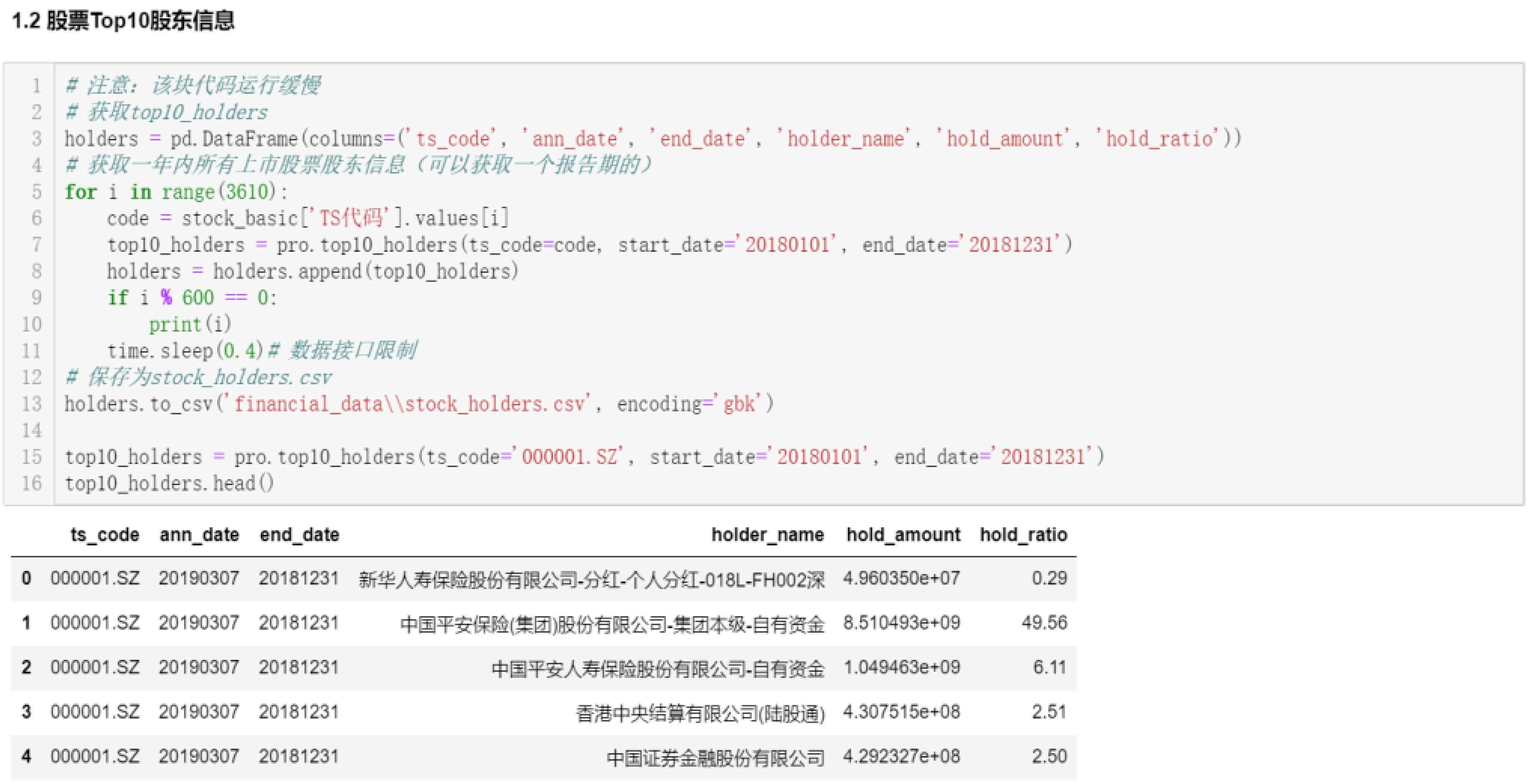


```python
# 获取股票基本信息
stock_basic = pro.stock_basic(list_status='L', fields='ts_code, symbol, name, industry')
# 重命名行，便于后面导入neo4j
basic_rename = {'ts_code': 'TS代码', 'symbol': '股票代码', 'name': '股票名称', 'industry': '行业'}
stock_basic.rename(columns=basic_rename, inplace=True)
# 保存为stock_basic.csv
stock_basic.to_csv('financial_data\\stock_basic.csv', encoding='gbk')
stock_basic.head()
```



	TS代码	股票代码	股票名称	行业
0	000001.SZ	000001	平安银行	银行
1	000002.SZ	000002	万科A	全国地产
2	000004.SZ	000004	国农科技	生物制药
3	000005.SZ	000005	世纪星源	环境保护
4	000006.SZ	000006	深振业A	区域地产

数据获取



1.2 股票Top10股东信息

```python
# 注意：该块代码运行缓慢
# 获取top10_holders
holders = pd.DataFrame(columns=('ts_code', 'ann_date', 'end_date', 'holder_name', 'hold_amount', 'hold_ratio'))
# 获取一年内所有上市股票股东信息（可以获取一个报告期的）
for i in range(3610):
    code = stock_basic['TS代码'].values[i]
    top10_holders = pro.top10_holders(ts_code=code, start_date='20180101', end_date='20181231')
    holders = holders.append(top10_holders)
    if i % 600 == 0:
        print(i)
    time.sleep(0.4)# 数据接口限制
# 保存为stock_holders.csv
holders.to_csv('financial_data\\stock_holders.csv', encoding='gbk')

top10_holders = pro.top10_holders(ts_code='000001.SZ', start_date='20180101', end_date='20181231')
top10_holders.head()
```

	ts_code	ann_date	end_date	holder_name	hold_amount	hold_ratio
0	000001.SZ	20190307	20181231	新华人寿保险股份有限公司-分红-个人分红-018L-FH002深	4.960350e+07	0.29
1	000001.SZ	20190307	20181231	中国平安保险(集团)股份有限公司-集团本级-自有资金	8.510493e+09	49.56
2	000001.SZ	20190307	20181231	中国平安人寿保险股份有限公司-自有资金	1.049463e+09	6.11
3	000001.SZ	20190307	20181231	香港中央结算有限公司(陆股通)	4.307515e+08	2.51
4	000001.SZ	20190307	20181231	中国证券金融股份有限公司	4.292327e+08	2.50

数据预处理

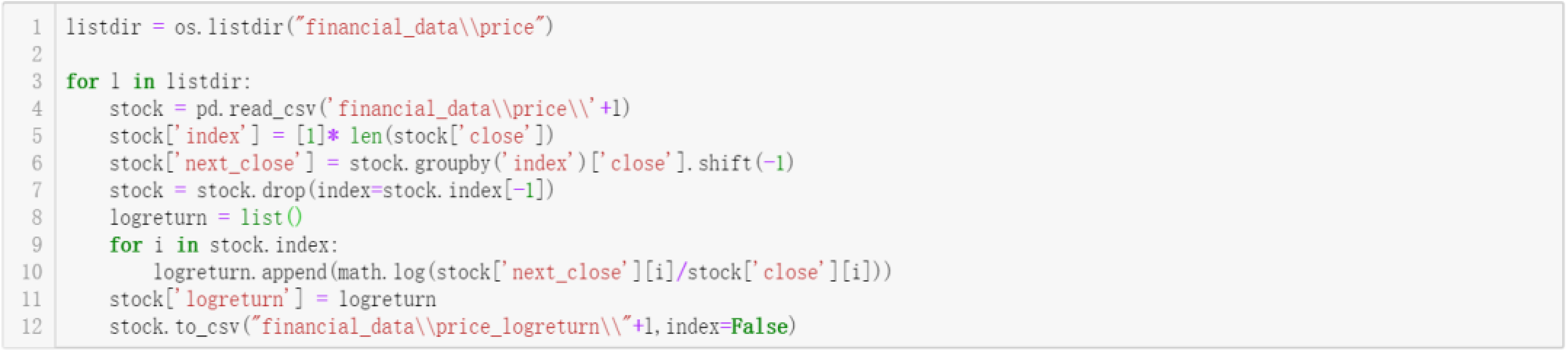
股票的对数收益

$$R_{i,t} = ln\frac{c_{i,t+1}}{c_{i,t}}, (t = 1,2, \dots d; i = 1,2, \dots, m)$$

皮尔逊积矩相关系数

$$\rho_{i,j} = \frac{\sum_{t=1}^{d}(R_{i,t} - \overline{\mathrm{R}_i})(R_{j,t} - \overline{\mathrm{R}_j})}{\sqrt{\sum_{t=1}^{d}(R_{i,t} - \overline{\mathrm{R}_i})^2 \sum_{t=1}^{d}(R_{j,t} - \overline{\mathrm{R}_j})^2}}$$

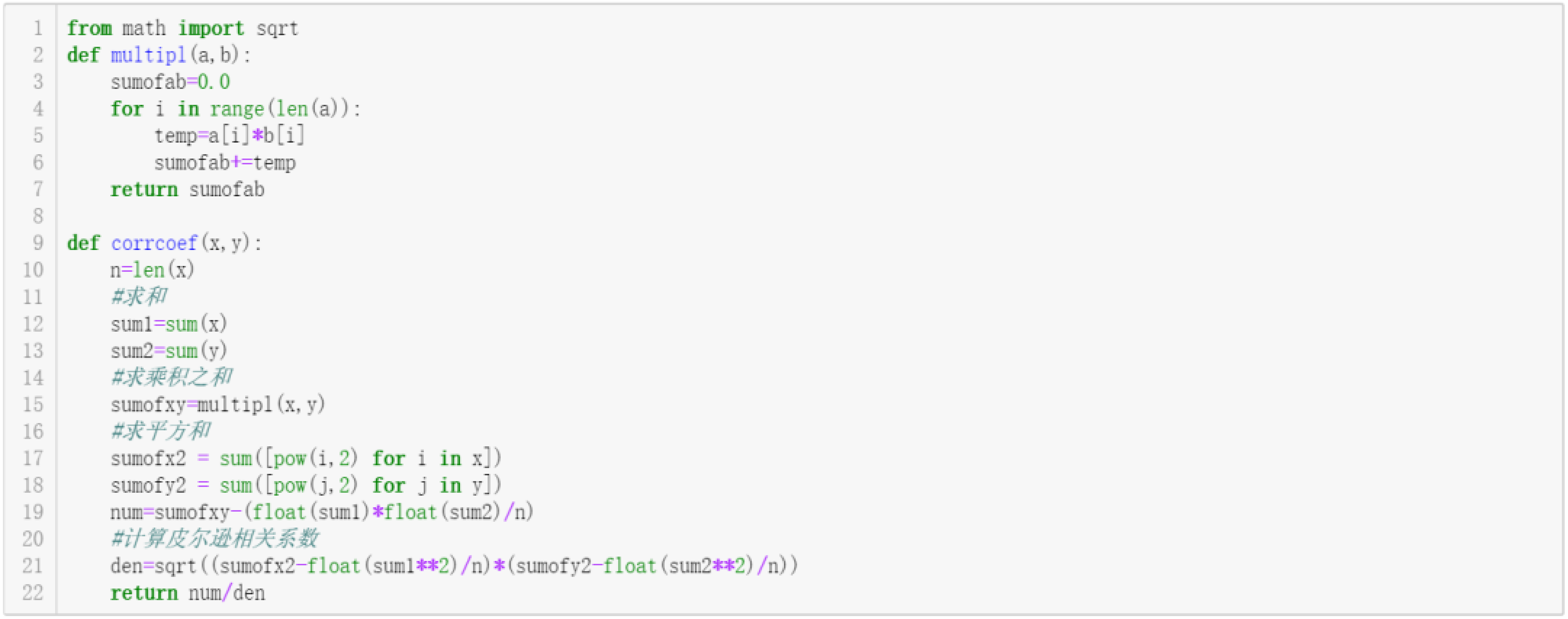
2.1 计算股票的对数收益



```
listdir = os.listdir("financial_data\\price")

for l in listdir:
    stock = pd.read_csv('financial_data\\price\\'+l)
    stock['index'] = [1]* len(stock['close'])
    stock['next_close'] = stock.groupby('index')['close'].shift(-1)
    stock = stock.drop(index=stock.index[-1])
    logreturn = list()
    for i in stock.index:
        logreturn.append(math.log(stock['next_close'][i]/stock['close'][i]))
    stock['logreturn'] = logreturn
    stock.to_csv("financial_data\\price_logreturn\\"+l,index=False)
```

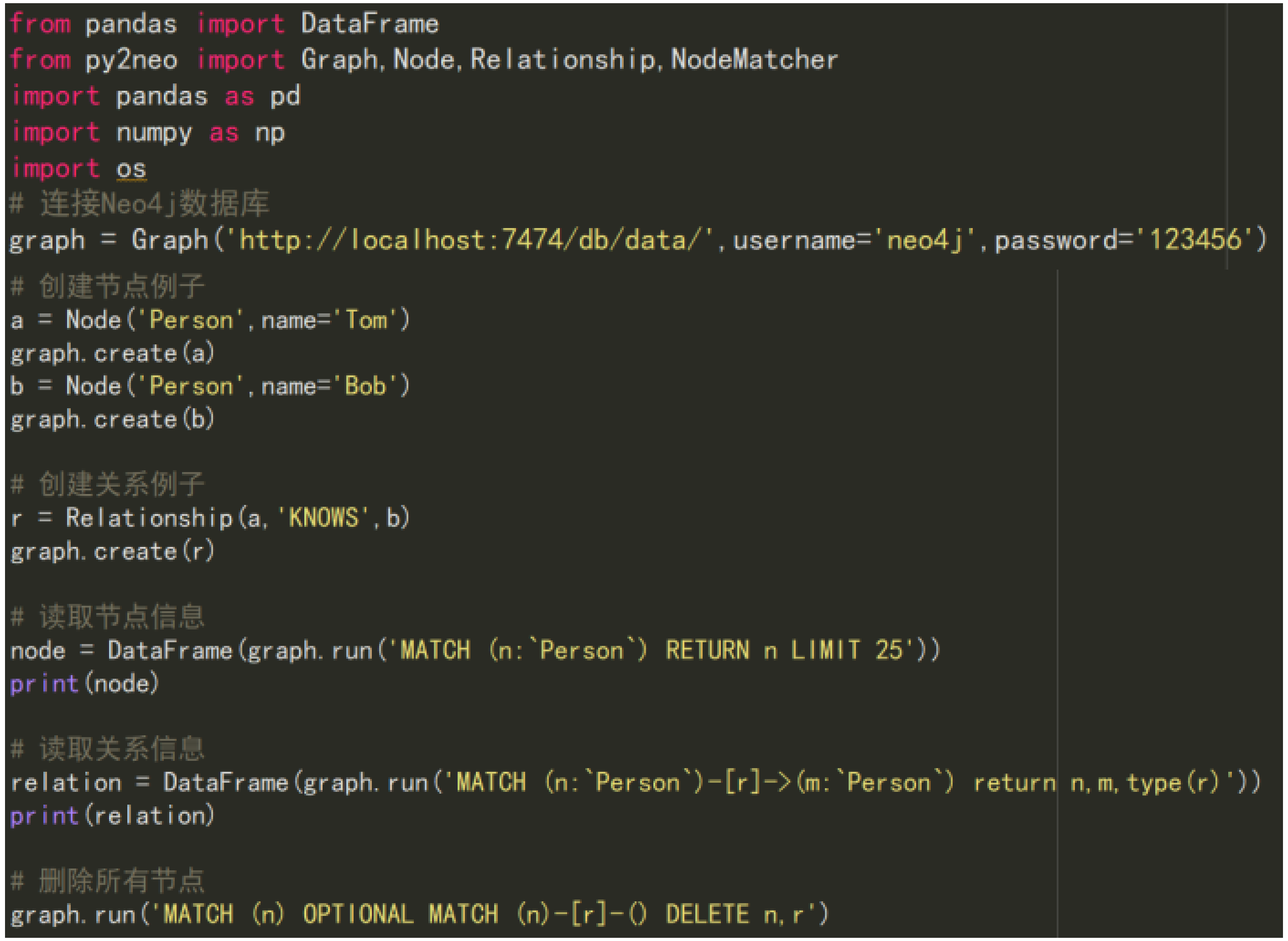
数据预处理

2.2 计算股票间的对数收益的皮尔逊相关系数

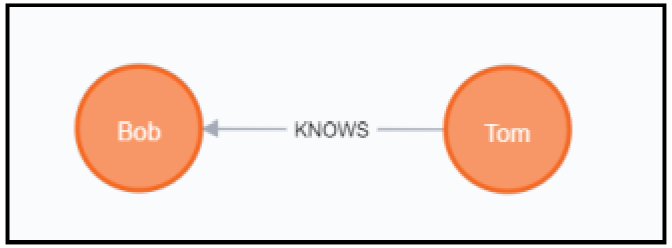


```python
from math import sqrt
def multipl(a,b):
    sumofab=0.0
    for i in range(len(a)):
        temp=a[i]*b[i]
        sumofab+=temp
    return sumofab

def corrcoef(x,y):
    n=len(x)
    #求和
    sum1=sum(x)
    sum2=sum(y)
    #求乘积之和
    sumofxy=multipl(x,y)
    #求平方和
    sumofx2 = sum([pow(i,2) for i in x])
    sumofy2 = sum([pow(j,2) for j in y])
    num=sumofxy-(float(sum1)*float(sum2)/n)
    #计算皮尔逊相关系数
    den=sqrt((sumofx2-float(sum1**2)/n)*(sumofy2-float(sum2**2)/n))
    return num/den
```

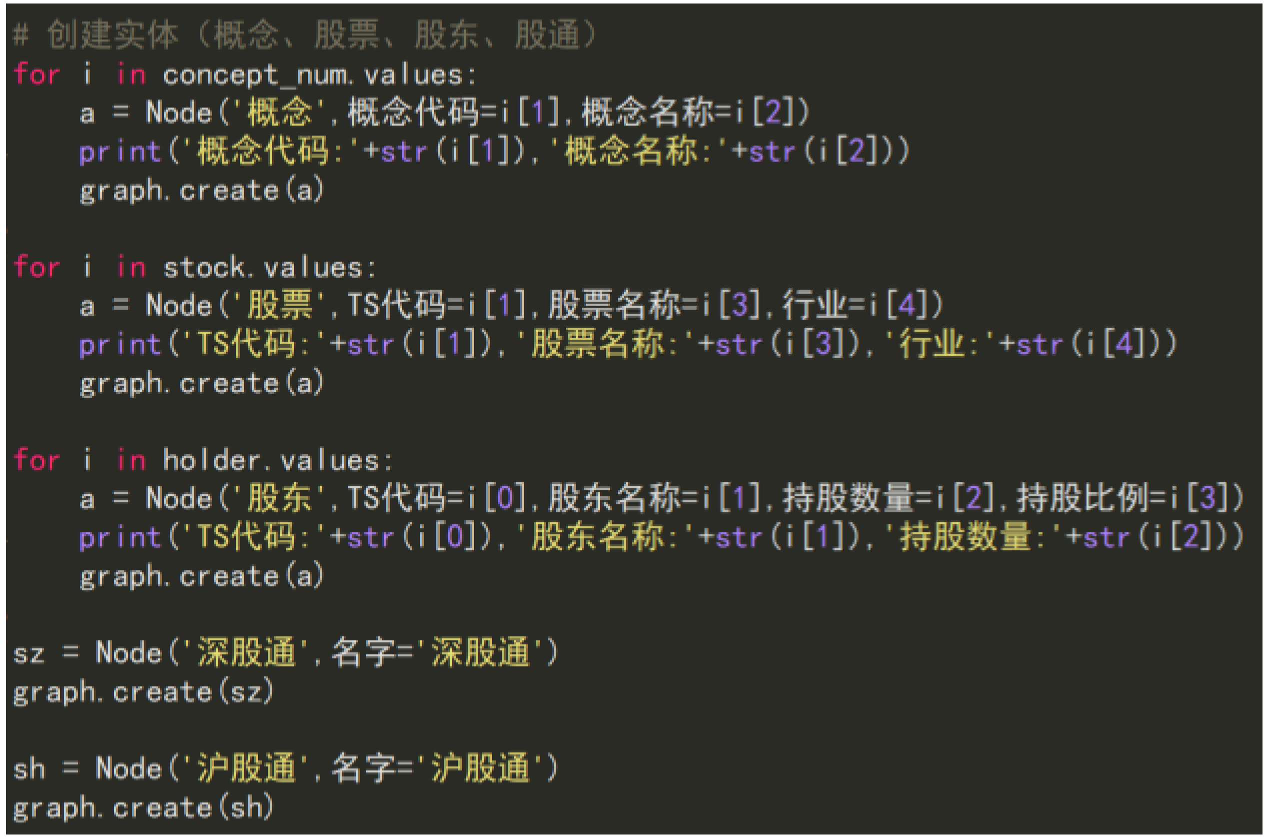
数据交互（Sample）



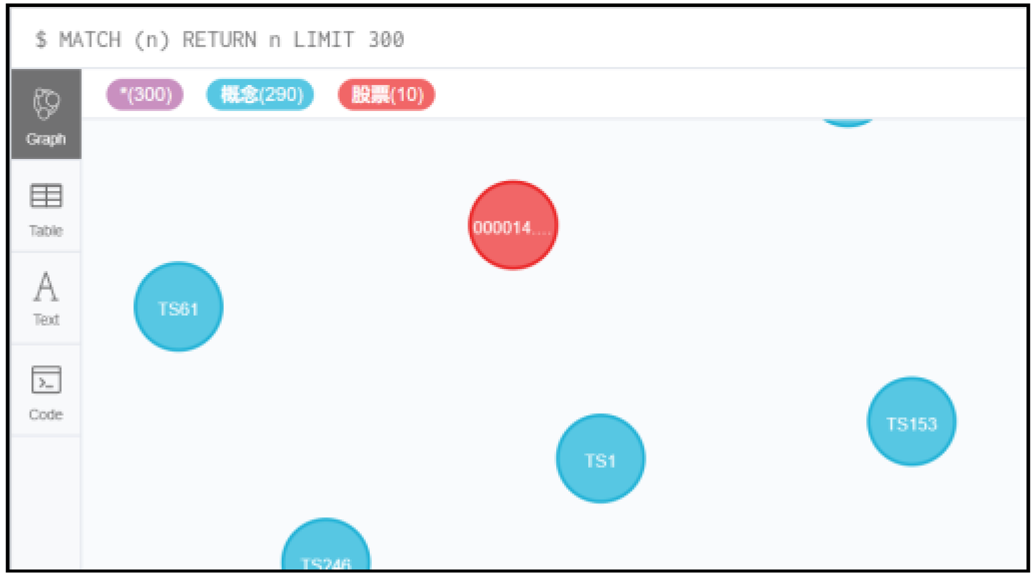
```python
from pandas import DataFrame
from py2neo import Graph,Node,Relationship,NodeMatcher
import pandas as pd
import numpy as np
import os
# 连接Neo4j数据库
graph = Graph('http://localhost:7474/db/data/',username='neo4j',password='123456')
# 创建节点例子
a = Node('Person',name='Tom')
graph.create(a)
b = Node('Person',name='Bob')
graph.create(b)

# 创建关系例子
r = Relationship(a,'KNOWS',b)
graph.create(r)

# 读取节点信息
node = DataFrame(graph.run('MATCH (n:`Person`) RETURN n LIMIT 25'))
print(node)

# 读取关系信息
relation = DataFrame(graph.run('MATCH (n:`Person`)-[r]->(m:`Person`) return n,m,type(r)'))
print(relation)

# 删除所有节点
graph.run('MATCH (n) OPTIONAL MATCH (n)-[r]-() DELETE n,r')
```



数据存储（创建实体）

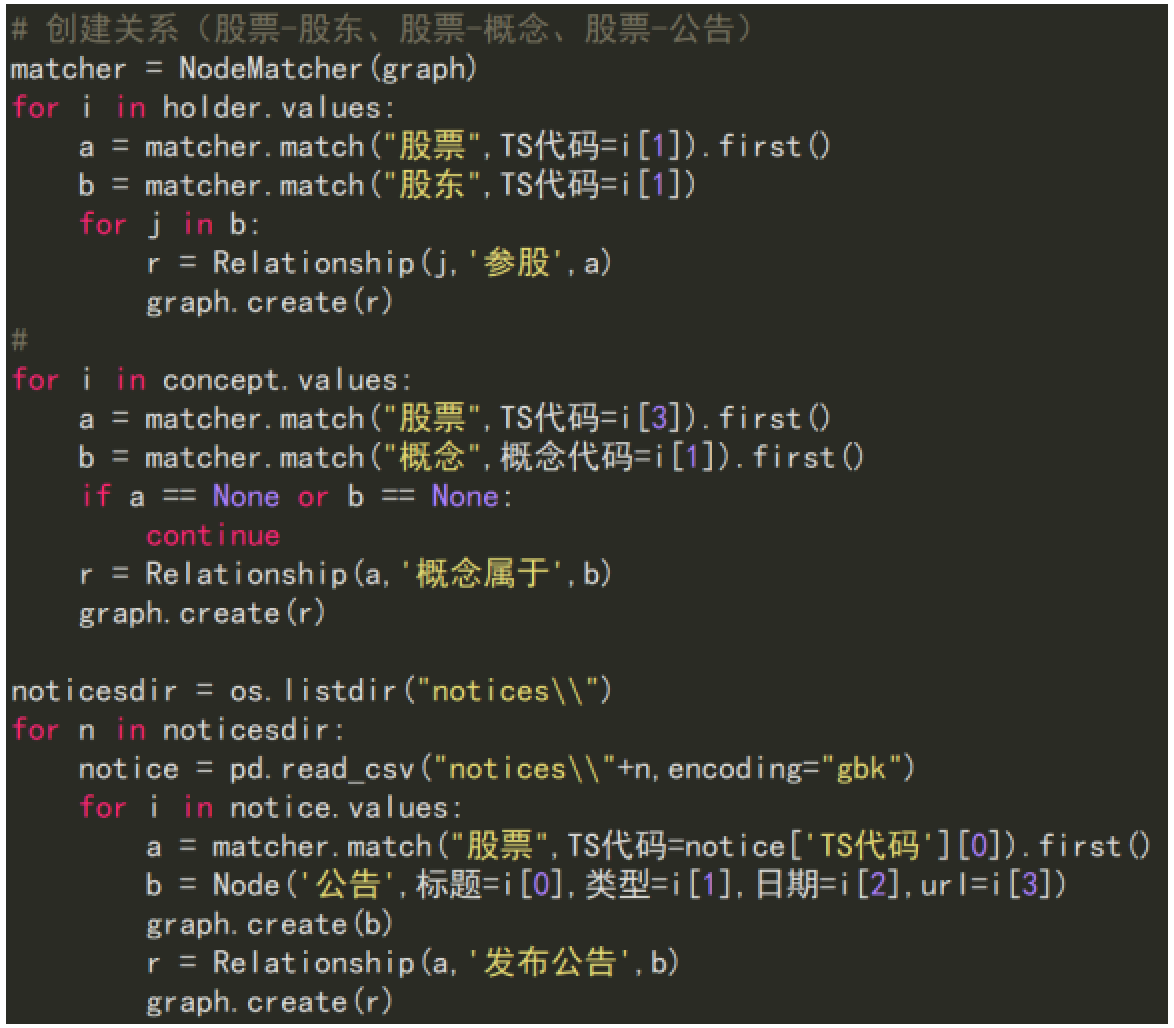


```
# 创建实体（概念、股票、股东、股通）
for i in concept_num.values:
    a = Node('概念',概念代码=i[1],概念名称=i[2])
    print('概念代码:'+str(i[1]),'概念名称:'+str(i[2]))
    graph.create(a)

for i in stock.values:
    a = Node('股票',TS代码=i[1],股票名称=i[3],行业=i[4])
    print('TS代码:'+str(i[1]),'股票名称:'+str(i[3]),'行业:'+str(i[4]))
    graph.create(a)

for i in holder.values:
    a = Node('股东',TS代码=i[0],股东名称=i[1],持股数量=i[2],持股比例=i[3])
    print('TS代码:'+str(i[0]),'股东名称:'+str(i[1]),'持股数量:'+str(i[2]))
    graph.create(a)

sz = Node('深股通',名字='深股通')
graph.create(sz)

sh = Node('沪股通',名字='沪股通')
graph.create(sh)
```

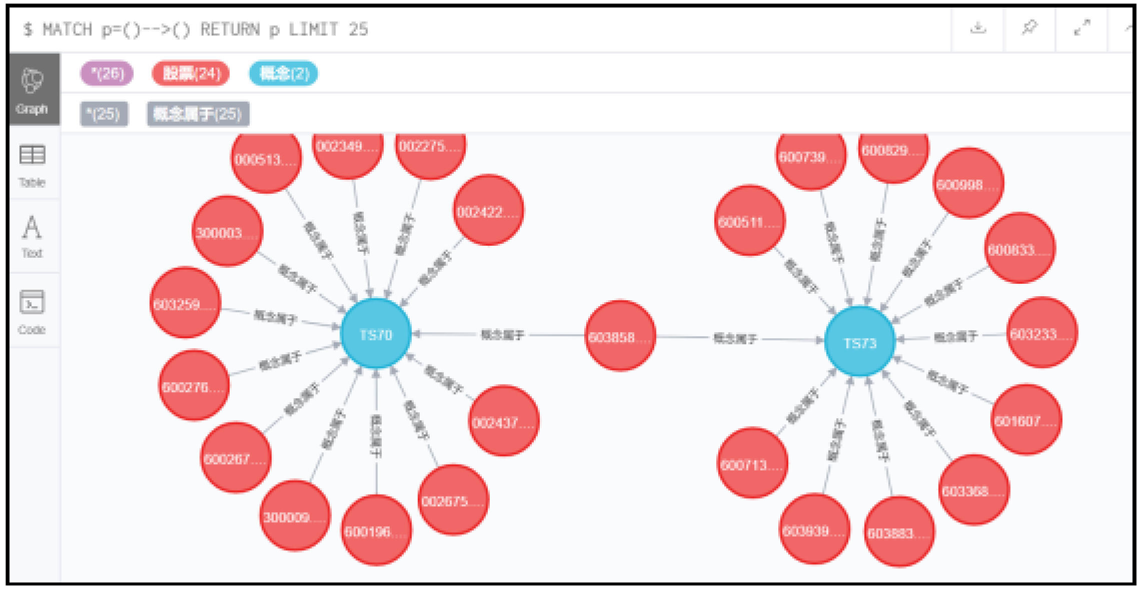


```
$ MATCH (n) RETURN n LIMIT 300
```

数据存储（创建关系）



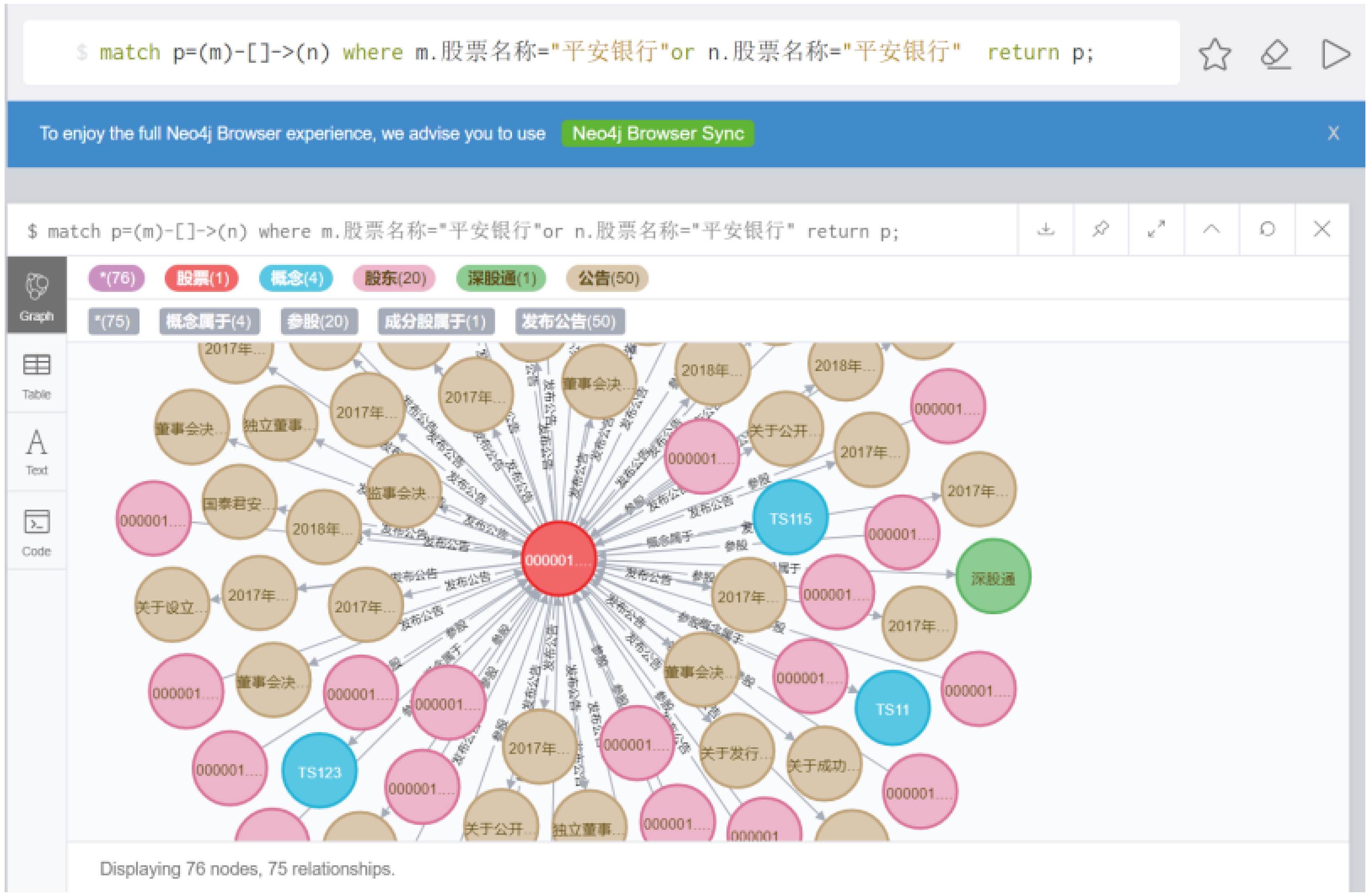
```
# 创建关系（股票-股东、股票-概念、股票-公告）
matcher = NodeMatcher(graph)
for i in holder.values:
    a = matcher.match("股票",TS代码=i[1]).first()
    b = matcher.match("股东",TS代码=i[1])
    for j in b:
        r = Relationship(j,'参股',a)
        graph.create(r)
#
for i in concept.values:
    a = matcher.match("股票",TS代码=i[3]).first()
    b = matcher.match("概念",概念代码=i[1]).first()
    if a == None or b == None:
        continue
    r = Relationship(a,'概念属于',b)
    graph.create(r)

noticesdir = os.listdir("notices\\")
for n in noticesdir:
    notice = pd.read_csv("notices\\"+n,encoding="gbk")
    for i in notice.values:
        a = matcher.match("股票",TS代码=notice['TS代码'][0]).first()
        b = Node('公告',标题=i[0],类型=i[1],日期=i[2],url=i[3])
        graph.create(b)
        r = Relationship(a,'发布公告',b)
        graph.create(r)
```



```
$ MATCH p=()-->() RETURN p LIMIT 25
```

数据可视化查询

查询与“平安银行”相关信息（所属概念板块、发布公告、属于深股通/沪股通、股东信息）



```
$ match p=(m)-[]->(n) where m.股票名称="平安银行"or n.股票名称="平安银行"  return p;
```

To enjoy the full Neo4j Browser experience, we advise you to use Neo4j Browser Sync

```
$ match p=(m)-[]->(n) where m.股票名称="平安银行"or n.股票名称="平安银行" return p;
```

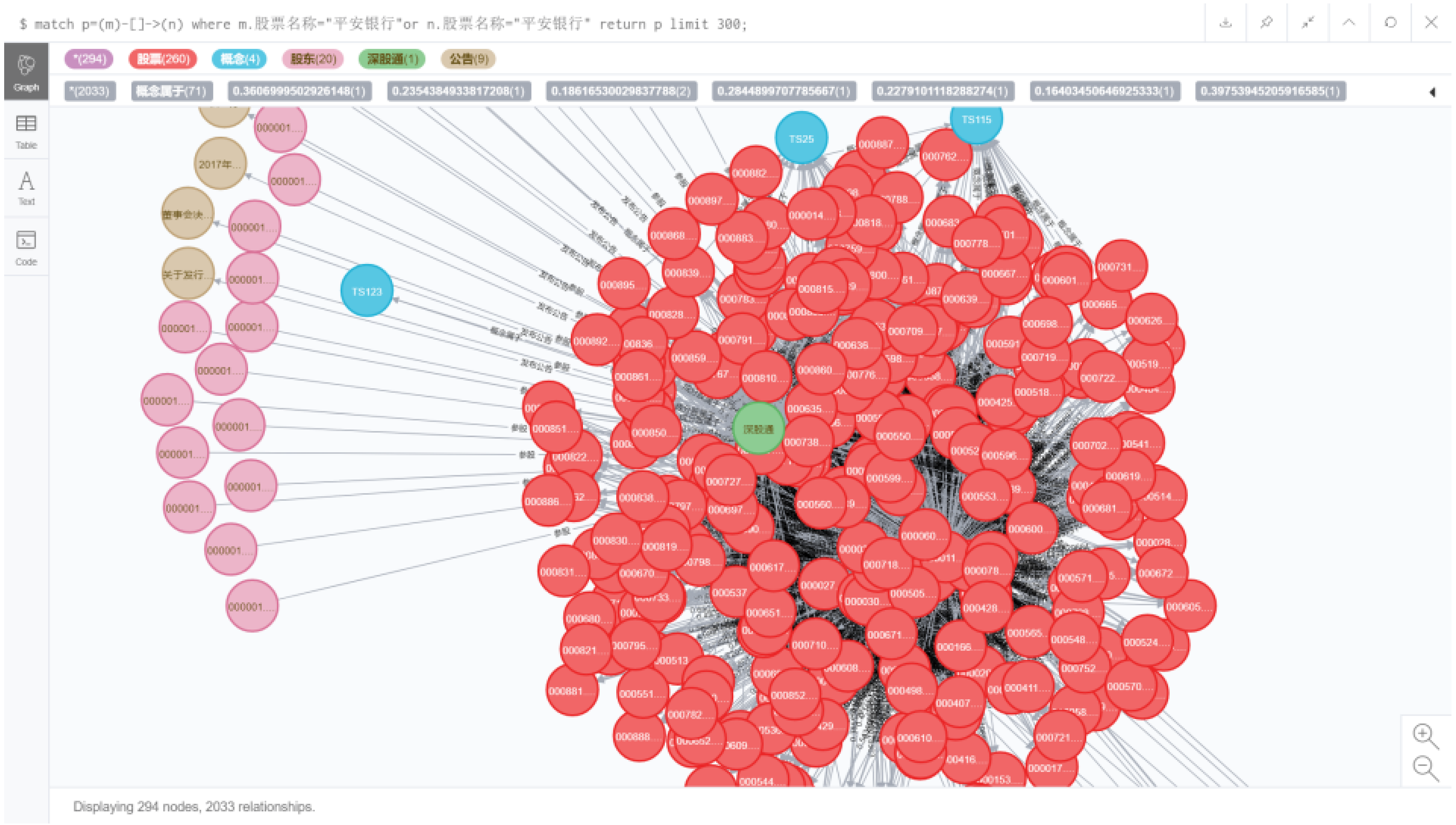
*(76) 股票(1) 概念(4) 股东(20) 深股通(1) 公告(50)

*(75) 概念属于(4) 参股(20) 成分股属于(1) 发布公告(50)

Displaying 76 nodes, 75 relationships.

数据可视化查询

插入股票间相关系数之后，显示与“平安银行”所有相关信息



```
$ match p=(m)-[]->(n) where m.股票名称="平安银行"or n.股票名称="平安银行" return p limit 300;
```

*(294) 股票(260) 概念(4) 股东(20) 深股通(1) 公告(9)

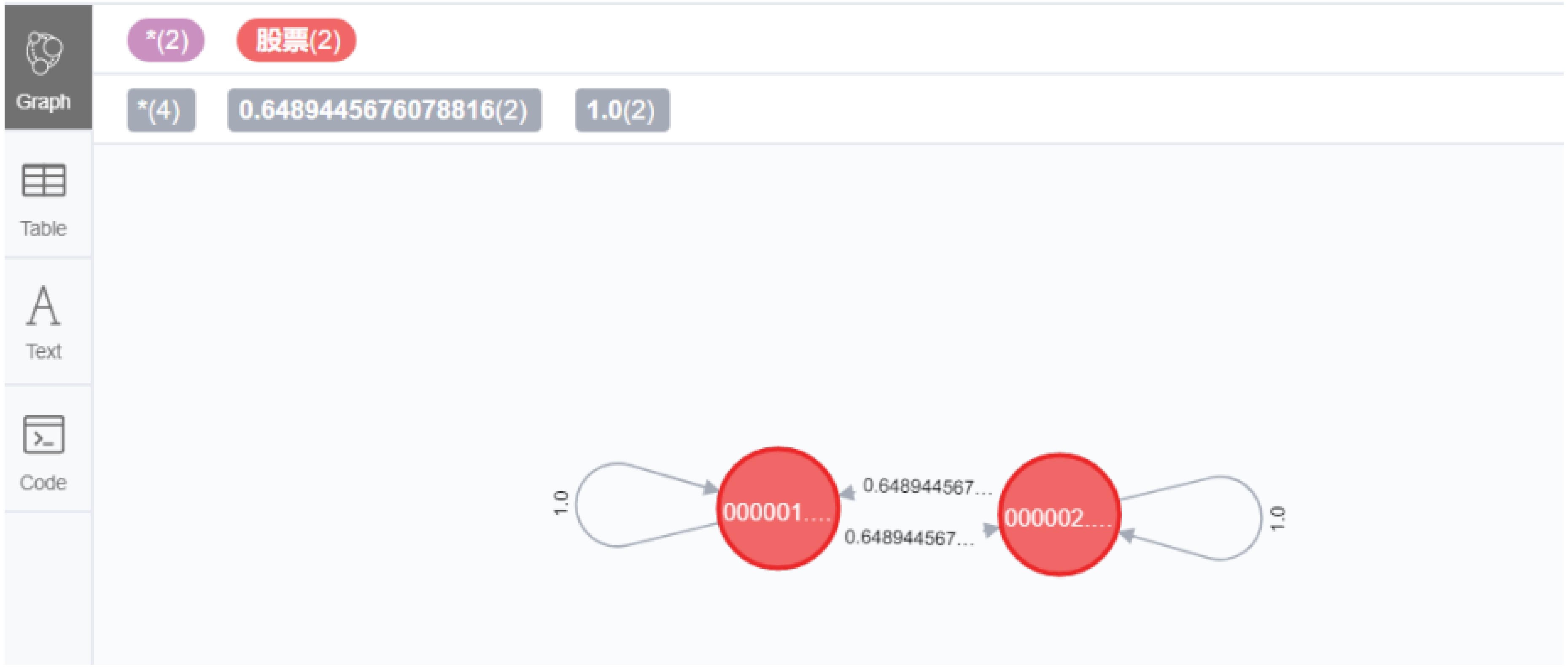
*(2033) 概念属于(71) 0.3606999502926148(1) 0.2354384933817208(1) 0.186165300298377 88(2) 0.2844899707785667(1) 0.2279101118288274(1) 0.1640345064692 5333(1) 0.3975394520591 6585(1)

Displaying 294 nodes, 2033 relationships.

数据可视化查询

查询“平安银行”与“万科A”的对数收益的相关系数

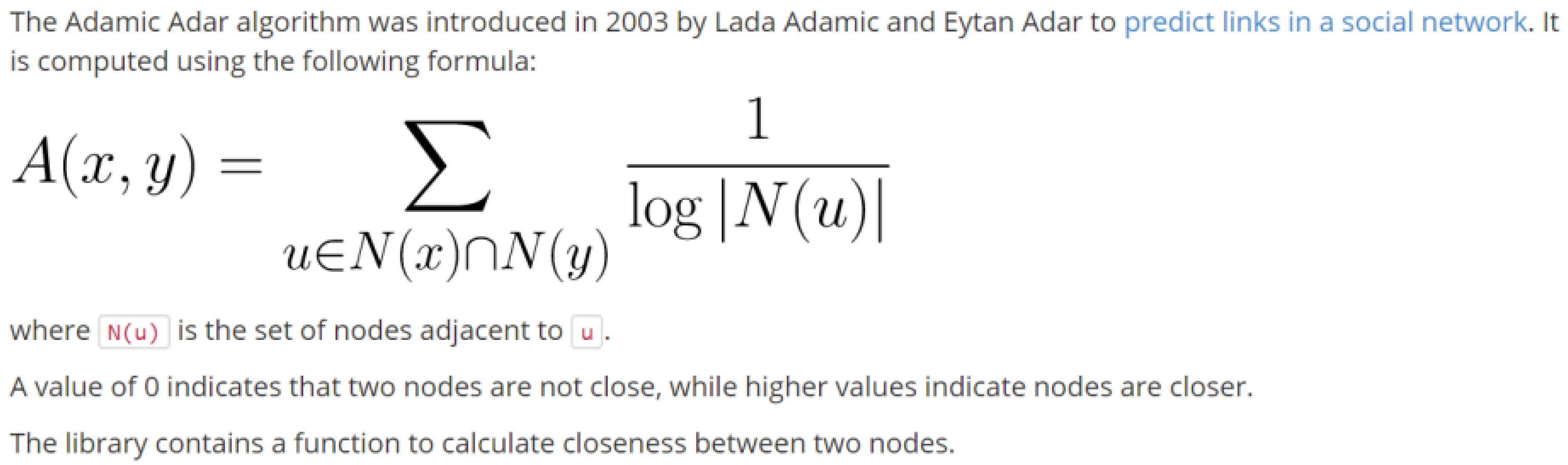
```
$ match p=(m)-[]->(n) where m.股票名称="平安银行" and n.股票名称="万科A" return p;
```



链路预测算法

使用neo4j附带的图算法，其中链路预测部分主要基于判断相邻的两个节点之间的亲密程度作为评判标准

1. The Adamic Adar algorithm



The Adamic Adar algorithm was introduced in 2003 by Lada Adamic and Eytan Adar to predict links in a social network. It is computed using the following formula:

$$A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log |N(u)|}$$

where `N(u)` is the set of nodes adjacent to `u`.

A value of 0 indicates that two nodes are not close, while higher values indicate nodes are closer.

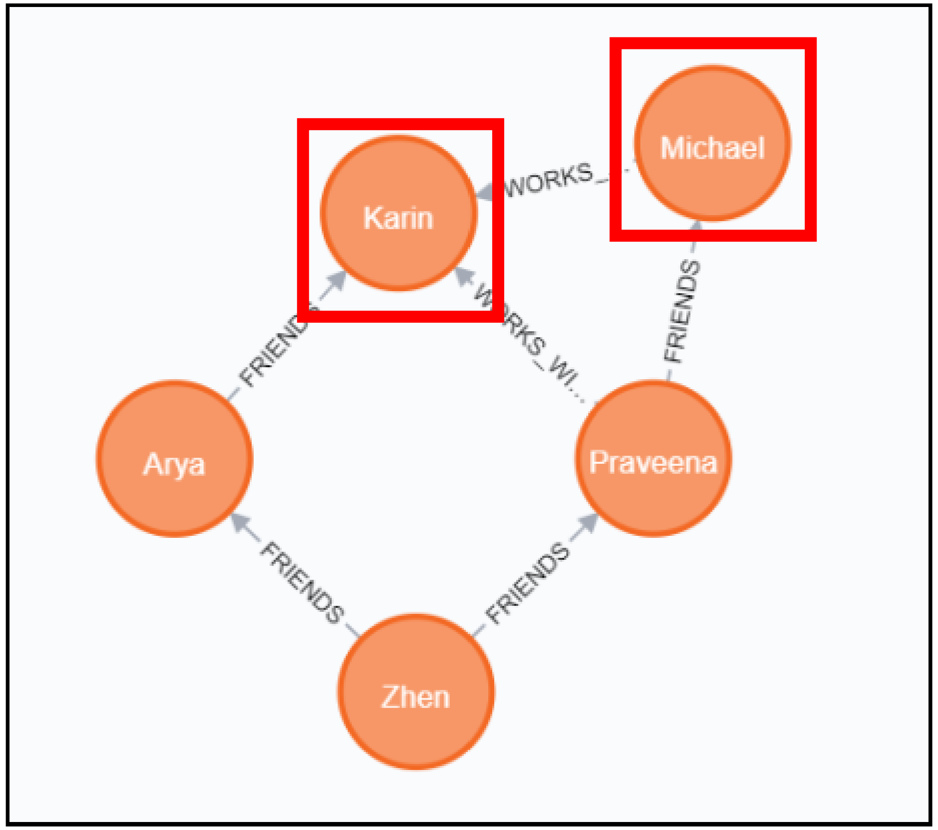
The library contains a function to calculate closeness between two nodes.

链路预测算法

1. The Adamic Adar algorithm

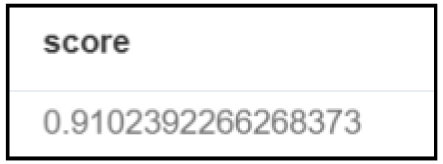
Sample

```
MERGE (zhen:Person {name: "Zhen"})
MERGE (praveena:Person {name: "Praveena"})
MERGE (michael:Person {name: "Michael"})
MERGE (arya:Person {name: "Arya"})
MERGE (karin:Person {name: "Karin"})

MERGE (zhen)-[:FRIENDS]-(arya)
MERGE (zhen)-[:FRIENDS]-(praveena)
MERGE (praveena)-[:WORKS_WITH]-(karin)
MERGE (praveena)-[:FRIENDS]-(michael)
MERGE (michael)-[:WORKS_WITH]-(karin)
MERGE (arya)-[:FRIENDS]-(karin)
```



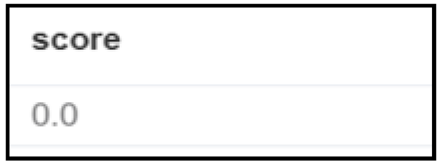
```
MATCH (p1:Person {name: 'Michael'})
MATCH (p2:Person {name: 'Karin'})
RETURN algo.linkprediction.adamicAdar(p1, p2) AS score
```

1/log(3)



```
score
0.9102392266268373
```

```
MATCH (p1:Person {name: 'Michael'})
MATCH (p2:Person {name: 'Karin'})
RETURN algo.linkprediction.adamicAdar(p1, p2, {relationshipQuery:
"FRIENDS"}) AS score
```



```
score
0.0
```

链路预测算法

（1）Adamic Adar	$A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log \lvert N(u) \rvert}$
（2）Common Neighbors	$CN(x,y) = \lvert N(x) \cap N(y) \rvert$
（3）Preferential Attachment	$PA(x,y) = \lvert N(x) \rvert * \lvert N(y) \rvert$
（4）Resource Allocation	$RA(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\lvert N(u) \rvert}$
（5）Same Community	
（6）Total Neighbors	$TN(x,y) = \lvert N(x) \cup N(y) \rvert$

其他算法

中心度算法(Centralities)：

（1）PageRank (页面排名)

（2）ArticleRank

（3）Betweenness Centrality (中介中心度)

（4）Closeness Centrality (接近中心度)

（5）Harmonic Centrality

社区检测算法(Community detection)：

（1）Louvain (鲁汶算法)

（2）Label Propagation (标签传播)

（3）Connected Components (连通组件)

（4）Strongly Connected Components (强连通组件)

（5）Triangle Counting / Clustering Coefficient (三角计数/聚类系数)

路径搜索算法(Path finding)：

（1）Minimum Weight Spanning Tree (最小权重生成树)

（2）Shortest Path (最短路径)

（3）Single Source Shortest Path (单源最短路径)

（4）All Pairs Shortest Path (全顶点对最短路径)

（5）A*

（6）Yen’s K-shortest paths

（7）Random Walk (随机漫步)