

The Dapper Squirrels

Using Spark GraphX

▾ Spark and GraphFrames Set Up

```
# install java
!apt-get install openjdk-8-jdk-headless -qq > /dev/null

# install spark (change the version number if needed)
!wget -q https://archive.apache.org/dist/spark/spark-3.2.0/spark-3.2.0-bin-hadoop3.2.

# unzip the spark file to the current folder
!tar xf spark-3.2.0-bin-hadoop3.2.tgz

# set your spark folder to your system path environment.
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.2.0-bin-hadoop3.2"

# install findspark using pip
!pip install -q findspark

# install pyspark
!pip3 install pyspark==3.2.0

# install graphframes
!pip3 install graphframes
```

```
Collecting pyspark==3.2.0
  Downloading pyspark-3.2.0.tar.gz (281.3 MB)
    |████████████████████████████████████████| 281.3 MB 30 kB/s
Collecting py4j==0.10.9.2
  Downloading py4j-0.10.9.2-py2.py3-none-any.whl (198 kB)
    |████████████████████████████████████████| 198 kB 41.1 MB/s
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.2.0-py2.py3-none-any.whl size=28
  Stored in directory: /root/.cache/pip/wheels/0b/de/d2/9be5d59d7331c6c2a7c1b6d1
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.2 pyspark-3.2.0
Collecting graphframes
  Downloading graphframes-0.6-py2.py3-none-any.whl (18 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (
Collecting nose
  Downloading nose-1.3.7-py3-none-any.whl (154 kB)
```

```

|████████████████████████████████████████| 154 kB 5.4 MB/s
Installing collected packages: nose, graphframes
Successfully installed graphframes-0.6 nose-1.3.7

```

Download the graphframes jar file from: [Graphframe jar file](#):

Upload it in the Google Colab Files folder. Can be found in the left pane of this window.

```

!cp -v /content/graphframes-0.8.2-spark3.2-s_2.12.jar $SPARK_HOME/jars/

'/content/graphframes-0.8.2-spark3.2-s_2.12.jar' -> '/content/spark-3.2.0-bin-ha

```

```

#import the packages
from pyspark import *
from pyspark.sql import *
from graphframes import *
import findspark
import pandas as pd
import psycopg2
import networkx as nx
import matplotlib.pyplot as plt

```

```
findspark.init()
```

```

# Start a Spark session
spark = SparkSession.builder.master("local[*]").getOrCreate()

```

```

/usr/local/lib/python3.7/dist-packages/psycopg2/__init__.py:144: UserWarning: Th
"""

```

```

# access the postgresql server
conn = psycopg2.connect(
    host="codd04.research.northwestern.edu",
    port = "5433",
    database="postgres",
    user="cpdbstudent",
    password="DataSci4AI")

```

```
cursor = conn.cursor()
```

▼ Question 1

Making nodes of officers and victims by their income, race, locations, and even unsupervised machine learning models to learn the cluster and see if there is a potential connection between officers and victims.

▼ 1.1 Learn the Connection from Race

Following query creates nodes and edges to answer the questions.

- **nodes:** race
- **edges:** src(officer id), dist(race) and relationship(complainat)

▶ 1.1.1 Graph Visualization Sample

In this section, we plot the visualized graph of the connection of the officer and the victim by race with part of the data.

```
[ ] ↪ 10 cells hidden
```

▶ 1.1.2 Graph Analysis on Race

Similarly, like the graph visualization, but we use all data now.

```
[ ] ↪ 10 cells hidden
```

1.1.3 Conclusion

We can find that the black community has a high volume of complaints. However, since the black people population is not extremely high in Chicago, we can assume there is a bias that may lead to over-policing.

We are not interested in the bias, this section is only used for proving our main theme, "Is there over-policing in low socio-economic status neighborhoods?" from a different aspect. There is more discussion in the following sections.

▼ 1.2 Learn the Connection from Location

Following query creates nodes and edges to answer the questions.

- **nodes:** community
- **edges:** src(officer name), dist(communitiy name) and relationship(CRd/TRRd)

▶ 1.2.1 Graph Visualization Sample

In this section, we plot the visualized graph of the connection of the officer and the victim by the location with part of the data.

```
[ ] ↪ 9 cells hidden
```

▼ 1.2.2 Graph Analysis on Location

Similarly, like the graph visualization, but we use all data now.

► Build GraphFrame

```
[ ] ↪ 4 cells hidden
```

► Graph Analysis

```
[ ] ↪ 8 cells hidden
```

1.2.3 Conclusion

We can conclude that communities like Austin, West Englewood, and Loop have a high volume of complaint report to officers, and Austin, Humboldt Park, and West Garfield Park have a large amount of TRRs. From this result we can find in the high-income community, people are more likely to complain about the behavior of the police. People from low-income communities receive more "threats" of tactical response. One possible explanation is that people who live in high-income communities have time to report the misbehavior of over-policing officers. But in the low-income community, people have no power to against the over-policing. Anyway, a high amount of reports of tactical response shows that there is potential over-policing behavior in those areas. Combining with the result we find in Checkpoint 1, a community like West Garfield Park is a low-income area. Therefore, we can assume that there is over-policing in the socio-economy status community.

▼ Question 2

Network dynamics of co-accused in each cohort can be interesting. The analytics can be done with the following:

1. Make use of Triangle Count Algorithms for each cohort.
2. Make use of the Page Rank Algorithm to find the most connected officer in all cohorts.
3. How many CRs that officers have and how many co-accused for each cohort.
4. Compare the top k largest cohort of police officers in high and low socio-economy status.

Following query creates nodes and edges to answer the questions for co-accused allegations.

1. Who among the officers have the most triangle counts?

2. Who have the most page rank score?
3. Are there any communities in the officers?
4. What are the allegation reports number for those officers inside a cluster?
5. What are the top large cohort of police officers in high and low socio-economy status?

This is how we define the nodes and edges for the graph.

- **nodes:** id, officer name and allegation count
- **edges:** src(officer1 id), dst(officer2 id) and relationship(allegation count)


These queries are to draw co-accused officers from allegation database. Basic logic is to join the allegation table with itself on the condition of the same allegation id and unequal officerid.

Nodes can be generated with data_officer table or allegation id by counting the number of allegation id. Here we chose data_officer table by removing Nan or 0s on allegation_count.

Note: These queries are copied and modified from the GraphX demo class, which shares similar analysis goal as us.

▼ 2.1 Community finding by Label propagation

► Build GraphFrame

 ↳ 14 cells hidden

▼ 2.2 Triangle Count analysis

Triangle counting algorithm is to count the triangle like relationship among 3 nodes which have connected in pairs. We want to find out those outstanding nodes in the graph which have a lot more triangle counts.

```
tc_cpdb = cpdb.triangleCount()
tc_cpdb.select("id", "count").show()
```

```
+-----+-----+
|   id | count |
+-----+-----+
| 33748 |     0 |
| 33751 |     0 |
| 33724 |     0 |
| 33798 |     0 |
| 33755 |     0 |
```

```

| 33746 | 0 |
| 33749 | 0 |
| 33737 | 0 |
| 33725 | 0 |
| 33738 | 0 |
| 33728 | 0 |
| 33752 | 0 |
| 33711 | 0 |
| 33723 | 0 |
| 33750 | 0 |
| 32312 | 37 |
| 32358 | 109 |
| 33753 | 0 |
| 33758 | 0 |
| 33709 | 0 |

```

```
+-----+-----+
```

only showing top 20 rows

In this part, we sorted all the nodes according to their triangle counts. We can see over 20 nodes appearing in over 18,000 triangle relationships, which indicates strong community leader potential like officer 6315 and 3033.

```
tc_cpdb.sort(['count'],ascending=False).show()
```

```

+-----+-----+-----+-----+
|count|  id| officer_name|allegation_count|
+-----+-----+-----+-----+
|32118| 6315| Terence Davis| 38|
|32117| 3033| Raimondo Brown| 17|
|32073| 3744| Derek Campbell| 8|
|27855|18042| Donald Mc Coy| 22|
|27823| 441| Fernando Alonzo| 16|
|23900|21530| Michael Overstreet| 56|
|23518|27349| Charles Stanton| 11|
|23499| 5180| Stephen Conner| 9|
|23487| 5667| Jerry Crawley| 30|
|23477|16747| Evetta Lundin| 7|
|23475| 8844| Thomas Flynn| 19|
|23472|23654| Lloyd Reid| 4|
|23472|14750| William Kissane| 23|
|20185|19856| Ronald Muhammad| 11|
|19322| 8138| Glenn Evans| 132|
|18773|29882| Fred Waller| 49|
|18648|28273| James Taylor| 36|
|18602|28459| Curtis Thomas| 36|
|18539| 5577| Michael Cox| 20|
|18502|30841| Teresa Williams| 37|

```

```
+-----+-----+-----+-----+
```

only showing top 20 rows

▼ 2.3 Page Rank analysis to find key nodes

Page rank algorithm is developed to find out important nodes inside a graph by iterations of calculations of the possibilities to get to the node by starting randomly.

```
pr_cpdb = cpdb.pageRank(resetProbability=0.15, tol=0.01)
#look at the pagerank score for every vertex
pr_cpdb.vertices.orderBy('pagerank', ascending=False).show()
```

id	officer_name	allegation_count	pagerank
32442	John Zinchuk	23	127.52903862900281
32440	Mark Zawila	34	90.32581504596747
32425	Perry Williams	27	75.93393690155354
32350	Robert Spiegel	20	72.52408784740014
32410	Joseph Watson	29	71.8959609008098
32430	Michael Wrobel	22	70.6024730642657
32074	Ronald Jenkins	46	70.26504490198167
32284	Mark Reno	76	68.44254003101547
32351	Boonserm Srisuth	25	66.23218732944623
32433	Kenneth Yakes	29	63.74966193544296
32419	Eric Wier	18	60.25243358901534
32384	Edwin Utreras	47	59.71305480353141
32435	Mohammad Yusuf	22	59.31175673367685
32413	Carl Weatherspoon	69	58.047513284732524
32337	Louis Silva	21	57.93147265165182
32431	Albert Wyroba	15	57.773544505418506
32289	John Rivera	44	56.566183401162725
32401	Joshua Wallace	45	55.97258828063104
32375	James Triantafillo	31	50.60713162542214
32436	Edmund Zablocki	28	48.62194138740303

only showing top 20 rows

From the above calculations, we can identify officers with significant impact in the graph. For example, officer 32442 and 32440 are major part in the clique and may be the "bad apple" in the organization.

▼ 2.4 The Corelation Between Police Cohort and CRs/TRRs

In this section, each police are counted for the time they had the same allegation with other police officers. The counted number will then be compared with the CRs and TRRs they gave and received to find the correlation between them. The goal of the correlation is to find whether police officers are more likely to misconduct when working as a group.

► Build GraphFrame

[] ↪ 3 cells hidden

► Graph Analysis

[] ↪ 10 cells hidden

Conclusion

The correlation between group allegation and complaint reports is positively correlated, and the group allegation is less correlated to tactical response reports. It is possible that when police officers are co-accused, they are more likely to have actual misconduct activity. It is because if they gave more tactical responses than receive complaints, or if they have a fairly equal amount of tactical responses and complaints, they would be less likely to have misconduct.

