**Project 2**

**CPSC 479**

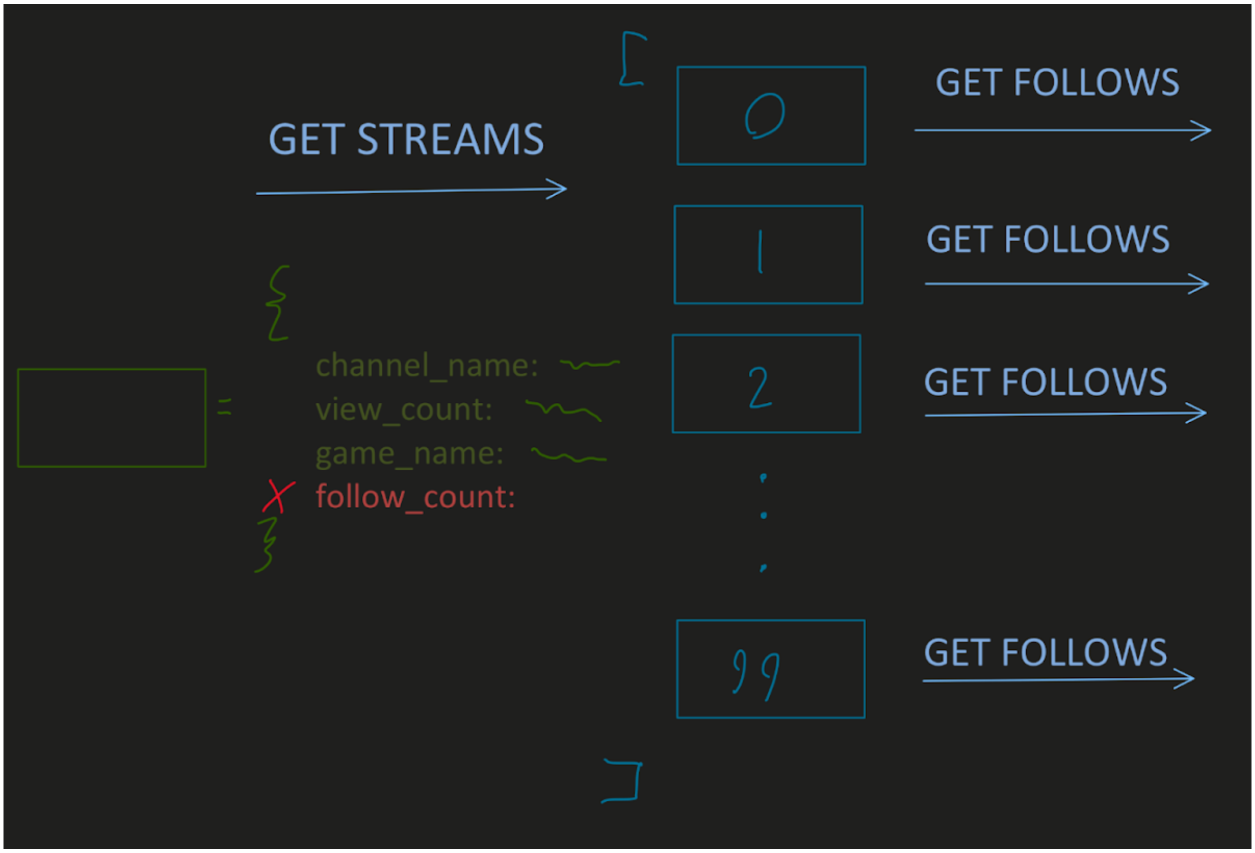
**Anthony Galustyan**

**Louis Zuckerman**

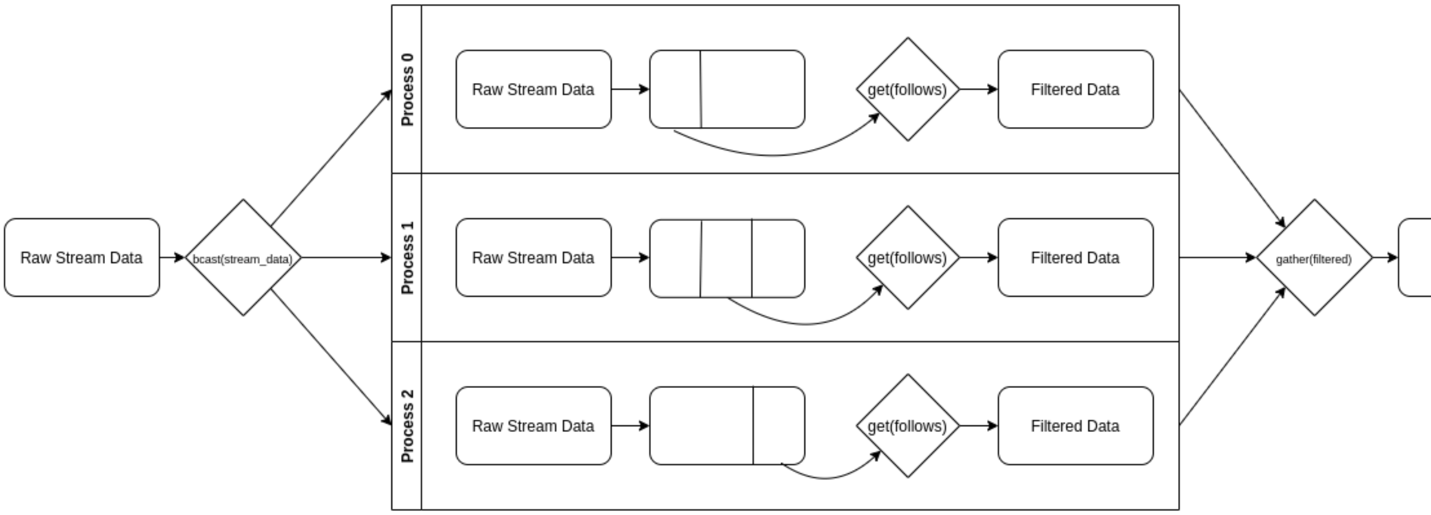
## Summary

Our implementation of the data science project features two sections of parallelization and one section of data analysis. The main purpose of our project is to pull data from the Twitch API and analyze that data. First, we parallelize our HTTP GET requests to the Twitch API servers. This allows us to increase our request rate by N processes. This was needed because getting follow information required many extra GET requests which were very slow. Then, we implement parallel sorting with merge sort, once it has a nearly sorted list we conduct an insertion sort on process 0. Following that, we conduct some correlation data analysis using pandas and numpy.

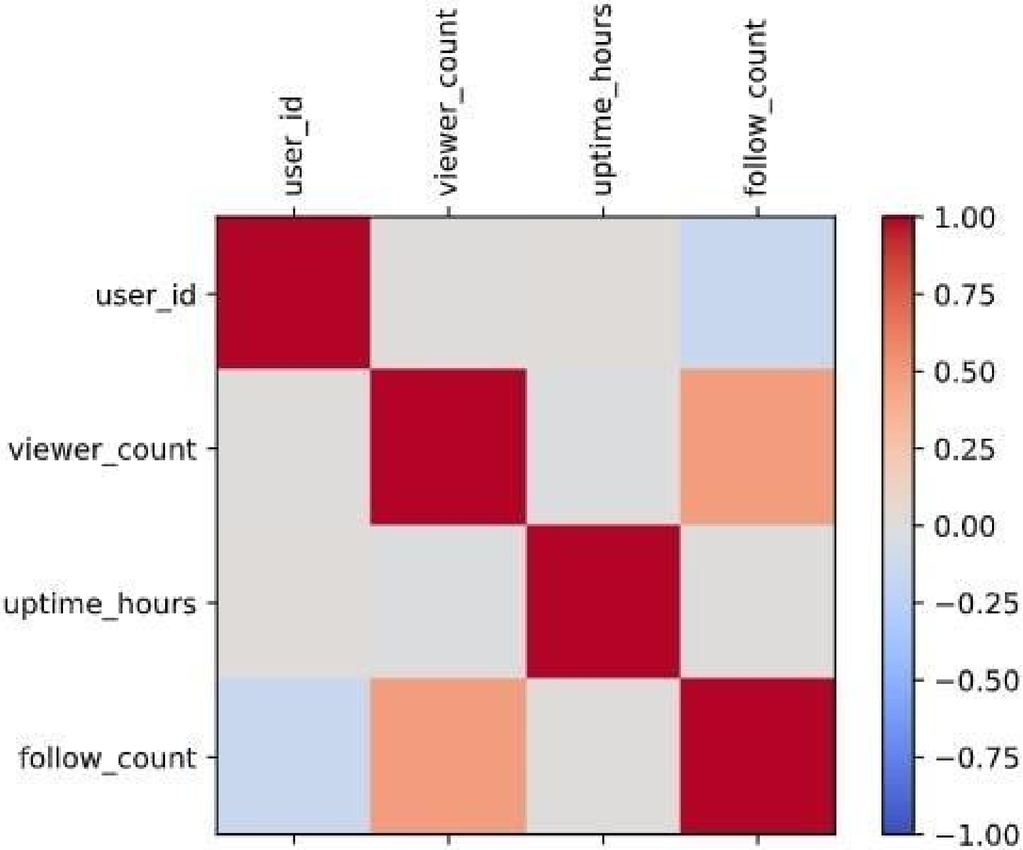
This diagram explains why we needed to parallelize the Get Follows requests

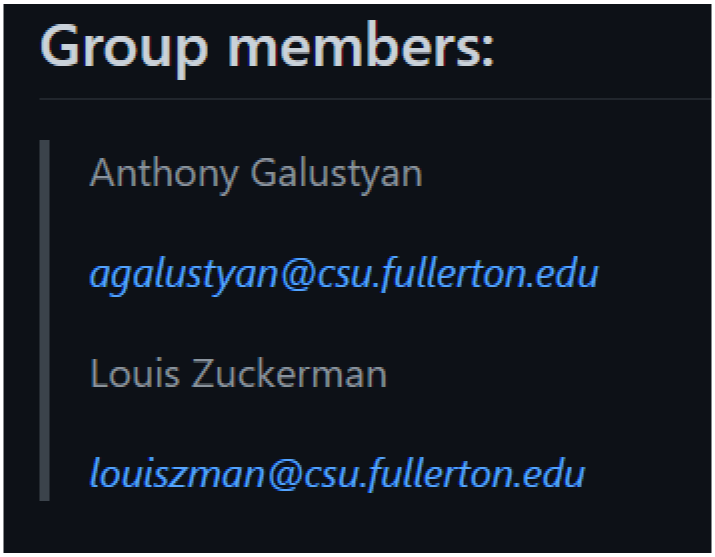


Implementation of parallel GET requests



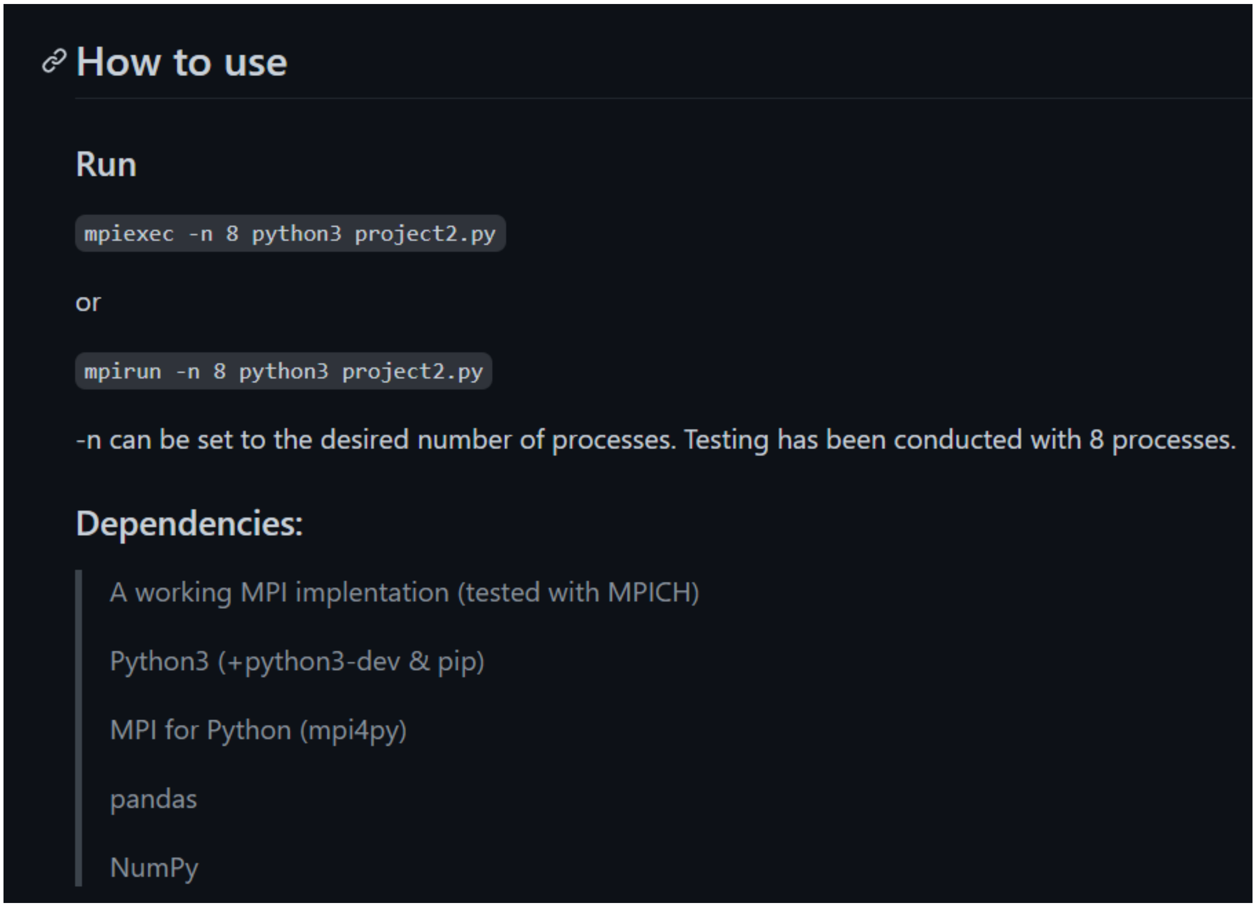
Correlation Chart





# Group members:

> Anthony Galustyan
>
> *agalustyan@csu.fullerton.edu*
>
> Louis Zuckerman
>
> *louiszman@csu.fullerton.edu*



## How to use

### Run

```
mpiexec -n 8 python3 project2.py
```

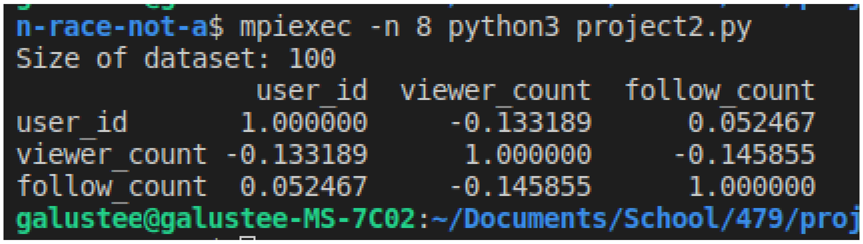
or

```
mpirun -n 8 python3 project2.py
```

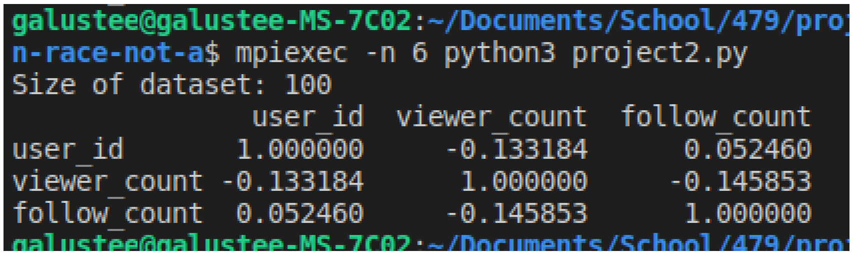
-n can be set to the desired number of processes. Testing has been conducted with 8 processes.

### Dependencies:

> A working MPI implentation (tested with MPICH)
>
> Python3 (+python3-dev & pip)
>
> MPI for Python (mpi4py)
>
> pandas
>
> NumPy



```
n-race-not-a$ mpiexec -n 8 python3 project2.py
Size of dataset: 100
                user_id  viewer_count  follow_count
user_id        1.000000     -0.133189      0.052467
viewer_count -0.133189      1.000000     -0.145855
follow_count  0.052467     -0.145855      1.000000
galustee@galustee-MS-7C02:~/Documents/School/479/proj
```



```
galustee@galustee-MS-7C02:~/Documents/School/479/proj
n-race-not-a$ mpiexec -n 6 python3 project2.py
Size of dataset: 100
                user_id  viewer_count  follow_count
user_id        1.000000     -0.133184      0.052460
viewer_count -0.133184      1.000000     -0.145853
follow_count  0.052460     -0.145853      1.000000
galustee@galustee-MS-7C02:~/Documents/School/479/proj
```

## Psuedocode

```
SET comm TO MPI.COMM_WORLD
SET rank TO comm.Get_rank()
SET size TO comm.Get_size()

# what we sort our data by
SET sort_category TO 'uptime_hours'

DEFINE FUNCTION mergeSort(arr):
    IF len(arr) > 1:
        SET mid TO len(arr)//2
        SET left TO arr[:mid]
        SET right TO arr[mid:]
        mergeSort(left)
        mergeSort(right)
        SET i TO 0
        SET j TO 0
        SET k TO 0

        WHILE i < len(left) and j < len(right):
            IF left[i][sort_category] > right[j][sort_category]:
                SET arr[k] TO left[i]
                i += 1
            ELSE:
                SET arr[k] TO right[j]
                j += 1
                k += 1

        WHILE i < len(left):
            SET arr[k] TO left[i]
            i += 1
            k += 1

        WHILE j < len(right):
            SET arr[k] TO right[j]
            j += 1
            k += 1

DEFINE FUNCTION insertionSort(arr):
    FOR i IN range(1, len(arr)):
        SET val TO arr[i]
        SET pos TO i

        WHILE pos > 0 and arr[pos - 1][sort_category] < val[sort_category]:
            SET arr[pos] TO arr[pos - 1]
            SET pos TO pos -1
        SET arr[pos] TO val

# 1 - 100
SET streamcount TO 100
SET multiplier TO 40
SET N TO streamcount * multiplier
SET partition TO N / size



SET streamlist TO []
SET raw_list TO []

SET headers TO {'client-id': os.environ['CLIENT_ID'], 'Authorization': 'Bearer ' + os.environ['API_AUTH']}

# sequential requests FOR top N streams
IF rank EQUALS 0:
    pagination=""
    FOR i IN range(0, multiplier):
        SET top_streams TO requests.get(f'https://api.twitch.tv/helix/streams?first={streamcount}&after={pagination}', headers=headers)
        raw_list.extend(top_streams.json()['data'])
        SET pagination TO top_streams.json()['pagination']['cursor']


# sends results of those requests to every process
SET raw_list TO comm.bcast(raw_list, root=0)
# slices based on rank
FOR stream IN raw_list[int(rank*partition):int(rank*partition+partition)]:
    SET stream_data TO {}


    # calculates how long a stream has been online
    SET start_time TO dateutil.parser.parse(stream['started_at'])
    SET current_time TO datetime.datetime.now(timezone.utc).replace(microsecond=0)
    SET uptime TO current_time - start_time
    SET stream_data['uptime'] TO str(uptime)



# converts HH:MM:SS uptime format to hours
    SET split_time TO re.split(',| |:', str(uptime))
    IF len(split_time) EQUALS 3:
        SET stream_data['uptime_hours'] TO int(split_time[0]) + int(split_time[1])/60 + int(split_time[2])/3600


    ELSEIF len(split_time) > 3:
        SET stream_data['uptime_hours'] TO int(split_time[0])*24 + int(split_time[3]) + int(split_time[4])/60 + int(split_time[5])/3600


    ELSE:
        SET stream_data['uptime_hours'] TO 0


    # sets stream info
    SET stream_data['game_name'] TO stream['game_name']
    SET stream_data['user_id'] TO stream['user_id']
    SET stream_data['user_login'] TO stream['user_login']
    SET stream_data['viewer_count'] TO stream['viewer_count']
    SET stream_data['title'] TO stream['title']


    # gets follower count of stream
    SET user_id TO stream_data['user_id']
    SET follow_count TO requests.get(f'https://api.twitch.tv/helix/users/follows?to_id={user_id}&first=1', headers=headers)
    SET stream_data['follow_count'] TO follow_count.json()['total']

    # makes sure we don't exceed rate limit
    IF int(follow_count.headers['Ratelimit-Remaining']) < 100:
        OUTPUT(f" Current rate limit: {follow_count.headers['Ratelimit-Remaining']}")
        OUTPUT(f"Waiting...")

        time.sleep(10)

    # local list of channel info
    streamlist.append(stream_data)

# gathers all local lists into a master list at process 0
# this is a list of lists
SET streamlist TO comm.gather(streamlist, root=0)
comm.Barrier()

# lists are scattered back out across processes
SET split_streamlist TO comm.scatter(streamlist, root=0)

# each process merge sorts their list
mergeSort(split_streamlist)

# reduce brings all the values IN the sorted lists together into one list
SET sorted_streamlist TO comm.reduce(split_streamlist, root TO 0)

# sorted_streamlist is now *nearly* sorted, so insertion sort works well here
IF rank EQUALS 0:
    insertionSort(sorted_streamlist)

IF rank EQUALS 0:
    # flattens the streamlist, which is still a list of lists
    # this maintains the original order of the get streams call
    SET viewcount_sorted TO [val FOR sublist IN streamlist FOR val IN sublist]
    OUTPUT(f"Size of dataset: {len(viewcount_sorted)}")
    with open('viewcount_data.json', 'w', encoding='utf-8') as f:
        json.dump(viewcount_sorted, f, ensure_ascii=False, indent=4)

    with open('sorted_data.json', 'w', encoding='utf-8') as f:
        json.dump(sorted_streamlist, f, ensure_ascii=False, indent=4)

IF rank EQUALS 0:
    SET json_output TO pd.read_json('viewcount_data.json')
    SET csv_output TO json_output.to_csv('raw_data.csv', index TO None, header=True)

    SET path TO 'raw_data.csv'
    SET dataset TO pd.read_csv(path, header TO 1)

    SET dataset.columns TO ['game_name', 'user_id', 'user_login', 'viewer_count', 'title', 'uptime', 'uptime_hours', 'follow_count']

#OUTPUT(dataset)
# determine IF there is correlation between viewer_count and uptime with target being follow_count // covariance
    SET numeric_dataset TO dataset[['user_id', 'viewer_count', 'uptime_hours', 'follow_count']]
    SET viewer TO dataset['viewer_count']
    SET uptime TO dataset['uptime_hours']
    SET follow_count TO dataset['follow_count']

    SET correlation TO numeric_dataset.corr()
    SET figure TO plt.figure()
    SET axis TO figure.add_subplot(111)
    SET caxis TO axis.matshow(correlation,cmap='coolwarm', vmin=-1, vmax=1)
    figure.colorbar(caxis)

    SET ticks TO np.arange(0,len(numeric_dataset.columns),1)
    axis.set_xticks(ticks)
    plt.xticks(rotation=90)
    axis.set_yticks(ticks)
    axis.set_xticklabels(numeric_dataset.columns)
    axis.set_yticklabels(numeric_dataset.columns)
    plt.show()

    OUTPUT(correlation)
```