Economic Connectedness

February 5, 2023

1 Replication Study Economic Connectedness

Dimitris Tsirmpas Replicating the results of the social capital study, found here.

The datasets used can be downloaded here.

1.1 Q1: The Geography of Social Capital in the United States

We begin by importing the social capital by county dataset. We note that the `County` column is not a number, so we import it as a string.

[1]:
```
import pandas as pd
import numpy as np

social_capital_df = pd.read_csv("data/social_capital_county.csv",␣
↪dtype={"county":str})
social_capital_df
```

[1]:
```
     county          county_name  num_below_p50    pop2018  ec_county  \
0      1001      Autauga, Alabama     5922.39210    55200.0    0.72077
1      1003      Baldwin, Alabama    15458.39600   208107.0    0.74313
2      1005      Barbour, Alabama     4863.97360    25782.0    0.41366
3      1007         Bibb, Alabama     3061.49340    22527.0    0.63152
4      1009       Blount, Alabama     6740.91160    57645.0    0.72562
...     ...                  ...            ...        ...        ...
3084  56037   Sweetwater, Wyoming     2402.96900    44117.0    0.96235
3085  56039        Teton, Wyoming      783.24982    23059.0    1.07623
3086  56041        Uinta, Wyoming     2174.06180    20609.0    0.95452
3087  56043     Washakie, Wyoming      872.51544     8129.0    0.90667
3088  56045       Weston, Wyoming      635.28436     7100.0    0.97840

      ec_se_county  child_ec_county  child_ec_se_county  ec_grp_mem_county  \
0          0.00831          1.11754             0.02467            0.77223
1          0.00661          0.83064             0.01629            0.76215
2          0.00978          0.58541             0.02707            0.35927
3          0.01175          0.72265             0.03027            0.68094
4          0.00985          0.76096             0.02466            0.79584
...            ...              ...                 ...                ...
3084       0.01280          1.14781             0.02794            1.13449
```

3085	0.01744	1.23113	0.04692	1.13296
3086	0.01404	1.04595	0.03455	0.92831
3087	0.01928	0.90794	0.04962	0.78223
3088	0.02036	1.09118	0.05823	0.93135

	ec_high_county	...	child_exposure_county	child_high_exposure_county	\
0	1.21372	...	1.14816	1.19944	
1	1.28302	...	0.84588	1.00797	
2	0.91897	...	0.63306	0.71967	
3	1.06378	...	0.71433	0.72395	
4	1.10569	...	0.74821	0.79375	
...	...	...	...	...	
3084	1.32399	...	1.12164	1.12907	
3085	1.63551	...	1.32874	1.35341	
3086	1.32040	...	1.05446	1.06284	
3087	1.29208	...	0.88480	0.88589	
3088	1.28553	...	1.03325	1.05526	

	bias_grp_mem_county	bias_grp_mem_high_county	child_bias_county	\
0	0.05526	-0.22748	0.02668	
1	0.02950	-0.21519	0.01802	
2	0.13457	-0.34086	0.07528	
3	0.04108	-0.27727	-0.01165	
4	0.00217	-0.24946	-0.01704	
...	...	...	...	
3084	0.09519	-0.12030	-0.02333	
3085	0.14337	-0.11958	0.07346	
3086	0.13816	-0.12194	0.00808	
3087	0.06667	-0.20435	-0.02615	
3088	0.02279	-0.17229	-0.05606	

	child_high_bias_county	clustering_county	support_ratio_county	\
0	-0.08229	0.10347	0.98275	
1	-0.05241	0.09624	0.98684	
2	-0.19714	0.14911	0.99911	
3	-0.15993	0.14252	0.99716	
4	-0.08745	0.11243	0.99069	
...	...	...	...	
3084	-0.08683	0.10809	0.99710	
3085	-0.07364	0.09253	0.98648	
3086	-0.06074	0.11204	0.99479	
3087	-0.06076	0.11592	0.99708	
3088	-0.04609	0.11927	0.99730	

	volunteering_rate_county	civic_organizations_county
0	0.04355	0.01518
1	0.06117	0.01526

```
2                        0.02093                        0.01474
3                        0.05294                        0.01439
4                        0.05704                        0.01724
...                          ...                            ...
3084                     0.07321                        0.01225
3085                     0.09747                        0.03223
3086                     0.06942                        0.01222
3087                     0.05843                        0.03512
3088                     0.13635                        0.02375

[3089 rows x 26 columns]
```

[2]:
```
county_df = social_capital_df.loc[:, ["county", "county_name", "ec_county"]].
↪copy()
county_df
```

[2]:
```
     county          county_name  ec_county
0      1001     Autauga, Alabama    0.72077
1      1003     Baldwin, Alabama    0.74313
2      1005     Barbour, Alabama    0.41366
3      1007        Bibb, Alabama    0.63152
4      1009      Blount, Alabama    0.72562
...     ...                  ...        ...
3084  56037  Sweetwater, Wyoming    0.96235
3085  56039       Teton, Wyoming    1.07623
3086  56041       Uinta, Wyoming    0.95452
3087  56043    Washakie, Wyoming    0.90667
3088  56045      Weston, Wyoming    0.97840

[3089 rows x 3 columns]
```

The dataframe seems complete, with no missing or added columns and its data corresponding to those in the csv file.

We already have all the data we need to plot the graph, that is the county id and its corresponding EC.

We will begin plotting the graph using Plotly. Our first obstacle is that we need to display the counties on Plotly. To do so we start by importing their coordinates from the online plotly datasets.

[3]:
```
from urllib.request import urlopen
import json

with urlopen('https://raw.githubusercontent.com/plotly/datasets/master/
↪geojson-counties-fips.json') as response:
    counties = json.load(response)
```

We now need to match the geometric data with our dataset.

According to the dataset documentation, the `County` column represents the FIPS code (*Federal*

Information Processing Standard code), which is also used by the geometric dataset to represent the counties.

As such we only need to hint to the library that it should match the data by using the `County` column as the location and the `EC_County` one as the value.

[4]:
```python
import plotly.express as px

fig = px.choropleth(county_df, geojson=counties,
                    locations='county',
                    color='ec_county',
                    color_continuous_scale="Viridis",
                    scope="usa")

fig.update_layout(margin={"r":0, "t":0, "l":0, "b":0})
fig.show()
```

Our approach worked but the map seems incomplete. To figure out why, we look up the FIPS code of our first county in the dataset here and note that it starts with a "0", unlike the one in our dataset.

Let's fix this issue by adding a "0" in front of 4-digit-long codes and reconstructing our graph.

[5]:
```python
const_len_fips = lambda fips: fips if len(fips) == 5 else "0" + fips
county_df.county = county_df.county.apply(const_len_fips)
county_df.county
```

[5]:
```
0       01001
1       01003
2       01005
3       01007
4       01009
        ...
3084    56037
3085    56039
3086    56041
3087    56043
3088    56045
Name: county, Length: 3089, dtype: object
```

[6]:
```python
fig = px.choropleth(county_df, geojson=counties,
                    locations='county',
                    color='ec_county',
                    color_continuous_scale="Viridis",
                    scope="usa")

fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

The issue is fixed, but some counties are still stubbornly left grey. We can see from the figure provided by the original research that some of them are indeed missing data.

Let's look where this "missing data" comes from.

[7]:
```
print("County NaN values: ", county_df.loc[county_df.county.isna()])
print("EC NaN values: ", county_df.loc[county_df.ec_county.isna()][["county"]])
```

```
County NaN values:  Empty DataFrame
Columns: [county, county_name, ec_county]
Index: []
EC NaN values:        county
52    01105
71    02060
82    02164
186   06003
255   08023
...     ...
2707  48433
2713  48447
2748  49009
2759  49031
2831  51091

[71 rows x 1 columns]
```

A cursory look into our spreadsheet seems to confirm that the EC column contents are indeed missing for these counties. There doesn't seem to be an issue with encoding, or ill-formatted fields. These are indeed *missing* data.

An actual issue however is that some counties are missing altogether. We can get a csv file of all US counties and merge it with our county dataframe.

[8]:
```
counties_complete_df = pd.read_csv("data/counties_FIPS.csv", dtype={"fips": str})

counties_complete_df = counties_complete_df.loc[~counties_complete_df.state.
 ↪isnull()]
counties_complete_df.fips = counties_complete_df.fips.apply(const_len_fips)

counties_complete_df["county_name"] = counties_complete_df.name + ", " +␣
 ↪counties_complete_df.state
counties_complete_df.drop(["name", "state"], axis=1, inplace=True)

counties_complete_df
```

[8]:

	fips	county_name
2	01001	Autauga County, AL
3	01003	Baldwin County, AL
4	01005	Barbour County, AL
5	01007	Bibb County, AL

```
6     01009      Blount County, AL
...     ...                    ...
3190  56037  Sweetwater County, WY
3191  56039       Teton County, WY
3192  56041       Uinta County, WY
3193  56043    Washakie County, WY
3194  56045      Weston County, WY

[3143 rows x 2 columns]
```

[9]:
```python
complete_df = county_df.merge(counties_complete_df, how="outer",␣
 ↪left_on="county", right_on="fips")

complete_df
```

[9]:
```
     county     county_name_x  ec_county   fips          county_name_y
0     01001  Autauga, Alabama    0.72077  01001     Autauga County, AL
1     01003  Baldwin, Alabama    0.74313  01003     Baldwin County, AL
2     01005  Barbour, Alabama    0.41366  01005     Barbour County, AL
3     01007     Bibb, Alabama    0.63152  01007        Bibb County, AL
4     01009   Blount, Alabama    0.72562  01009      Blount County, AL
...     ...               ...        ...    ...                    ...
3140    NaN               NaN        NaN  51730    Petersburg city, VA
3141    NaN               NaN        NaN  51775         Salem city, VA
3142    NaN               NaN        NaN  51790      Staunton city, VA
3143    NaN               NaN        NaN  51820    Waynesboro city, VA
3144    NaN               NaN        NaN  51830  Williamsburg city, VA

[3145 rows x 5 columns]
```

[10]:
```python
# Copy the county FIPS and name from the extra dataset whenever
# they are absent from our original dataset
complete_df.county = np.where(complete_df.county.isnull(),
                              complete_df.fips,
                              complete_df.county)

complete_df.county_name_x = np.where(complete_df.county_name_x.isnull(),
                                     complete_df.county_name_y,
                                     complete_df.county_name_x)

complete_df.drop(["fips", "county_name_y"], axis=1, inplace=True)
complete_df.rename(mapper={"county_name_x" : "county_name"}, axis=1,␣
 ↪inplace=True)

# check that there is indeed no NA value in the county-id fields
complete_df.loc[complete_df.county.isnull() | complete_df.county_name.isnull()]
```

[10]: Empty DataFrame
Columns: [county, county_name, ec_county]
Index: []

Finally, we will pick a sentinel value to represent the missing data and store the minimum and maximum EC before applying it to the `ec_county` column.

[11]:
```
# We store these variables so they dont't get changed after we fill the missing␣
↪values with a sentinel
min_ec = county_df.ec_county.min()
max_ec = county_df.ec_county.max()
print(f"EC min:{min_ec}, EC max:{max_ec}")
```

```
EC min:0.29469001, EC max:1.3597
```

[12]:
```
sentinel = 0
county_df.ec_county = county_df.ec_county.fillna(sentinel)
```

Now let's add the appropriate colors, labels and general polish to our graph.

We will insert our own colorscale, so we can hide the missing value color from the colorbar (and more importantly so it doesn't interfere with the color-value range).

[13]:
```
import plotly.graph_objects as go

colorscale=[[sentinel, 'gray'], [sentinel + 0.001, 'gray'], [sentinel + 0.001,␣
↪'blue'],
            [(max_ec - min_ec)/2, "white"], [1, 'red']]

fig = go.Figure(data=px.choropleth(
                    county_df,
                    geojson=counties,
                    locations='county',
                    color='ec_county',
                    color_continuous_scale=colorscale,
                    range_color=(min_ec, max_ec), #don't include sentinel in the␣
↪graph's colorscale
                    labels={
                        "county_name": "County",
                        "ec_county":"Economic Connectedness"
                    },
                    hover_data={
                        "county": False,
                        "county_name": True,
                        "ec_county": True
                    },
                    scope="usa"))

fig.update_layout(
```

```
    title_text = "Economic Connectedness by US County",
    margin={"r":0,"t":0,"l":0,"b":0}
)

fig.show()
```

1.2 Q2: Economic Connectedness and Outcomes

Let's import the Opportunity Atlas dataset. The warning simply tells us there

[14]:
```
# We can't do anything about this warning, since the problem lies with the data␣
↪themselves
# in some of the 11000 columns. We will deal with wrong types if they pop up.
opp_df = pd.read_csv("data/county_outcomes.csv")
opp_df
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_7804\3413104665.py:3: DtypeWarning:

Columns (7886) have mixed types. Specify dtype option on import or set
low_memory=False.
```

[14]:
```
      state  county  kir_natam_female_p1  kir_natam_female_p25  \
0         1       1                  NaN                   NaN
1         1       3               0.3436              0.343627
2         1       5                  NaN                   NaN
3         1       7                  NaN                   NaN
4         1       9                  NaN                   NaN
...     ...     ...                  ...                   ...
3214     72     145                  NaN                   NaN
3215     72     147                  NaN                   NaN
3216     72     149                  NaN                   NaN
3217     72     151                  NaN                   NaN
3218     72     153                  NaN                   NaN

      kir_natam_female_p50  kir_natam_female_p75  kir_natam_female_p100  \
0                      NaN                   NaN                    NaN
1                 0.343645              0.343667               0.343722
2                      NaN                   NaN                    NaN
3                      NaN                   NaN                    NaN
4                      NaN                   NaN                    NaN
...                    ...                   ...                    ...
3214                   NaN                   NaN                    NaN
3215                   NaN                   NaN                    NaN
3216                   NaN                   NaN                    NaN
3217                   NaN                   NaN                    NaN
3218                   NaN                   NaN                    NaN
```

```
      kir_natam_female_n  kir_natam_female_mean  jail_natam_female_p1  ...  \
0                    NaN                    NaN                   NaN  ...
1                   42.0               0.341199             -0.010921  ...
2                    NaN                    NaN                   NaN  ...
3                    NaN                    NaN                   NaN  ...
4                    NaN                    NaN                   NaN  ...
...                  ...                    ...                   ...  ...
3214                 NaN                    NaN                   NaN  ...
3215                 NaN                    NaN                   NaN  ...
3216                 NaN                    NaN                   NaN  ...
3217                 NaN                    NaN                   NaN  ...
3218                 NaN                    NaN                   NaN  ...

      coll_white_pooled_mean_se  comcoll_white_pooled_mean_se  \
0                      0.020800                      0.021270
1                      0.014500                      0.014719
2                      0.035349                      0.038319
3                      0.040235                      0.040666
4                      0.018691                      0.022029
...                         ...                           ...
3214                        NaN                           NaN
3215                        NaN                           NaN
3216                        NaN                           NaN
3217                        NaN                           NaN
3218                        NaN                           NaN

      somecoll_white_pooled_mean_se  hs_white_pooled_mean_se  \
0                          0.020339                 0.012137
1                          0.012726                 0.007792
2                          0.030695                 0.019642
3                          0.043610                 0.025271
4                          0.020685                 0.012227
...                             ...                      ...
3214                            NaN                      NaN
3215                            NaN                      NaN
3216                            NaN                      NaN
3217                            NaN                      NaN
3218                            NaN                      NaN

      wgflx_rk_white_pooled_mean_se  hours_wk_white_pooled_mean_se  \
0                          0.018251                       1.131328
1                          0.012266                       0.741782
2                          0.027699                       1.673795
3                          0.036064                       2.380808
4                          0.017313                       1.158366
...                             ...                            ...
3214                            NaN                            NaN
```

```
3215                                   NaN                               NaN
3216                                   NaN                               NaN
3217                                   NaN                               NaN
3218                                   NaN                               NaN

      kfr_native_white_pooled_mean_se  kir_native_white_pooled_mean_se  \
0                            0.008103                         0.008534
1                            0.005500                         0.005603
2                            0.013528                         0.013531
3                            0.016382                         0.016979
4                            0.007895                         0.008158
...                               ...                              ...
3214                              NaN                              NaN
3215                              NaN                              NaN
3216                              NaN                              NaN
3217                              NaN                              NaN
3218                              NaN                              NaN

      kir_imm_white_pooled_mean_se  kfr_imm_white_pooled_mean_se
0                         0.057445                      0.058009
1                         0.041219                      0.037302
2                              NaN                           NaN
3                              NaN                           NaN
4                              NaN                           NaN
...                            ...                           ...
3214                           NaN                           NaN
3215                           NaN                           NaN
3216                           NaN                           NaN
3217                           NaN                           NaN
3218                           NaN                           NaN

[3219 rows x 10827 columns]
```

This dataset seems to have a different implementation of the FIPS code, where instead of one unique FIPS code, a county's FIPS code is determined by the individual FIPS codes of its state and its code within that state.

Fortunately, merging the state and county FIPS codes into a unified FIPS key is simple: `Unified_FIPS = State_Code * 1000 + Individual_County_Code`.

We will replace the state and county fields with the FIPS code so we can merge our data with the social capital dataframe from the previous question.

[15]:
```
def fips_converter(state_code, county_code):
    fips = state_code.apply(lambda x: x*1000) + county_code
    fips = fips.apply(str)
    return fips
```

```
opp_df["fips"] = fips_converter(opp_df.state, opp_df.county)

# make FIPS the first column of the dataframe
opp_df.insert(0, "fips", opp_df.pop("fips"))
opp_df.drop(["county", "state"], axis=1, inplace=True)
opp_df
```

[15]:

```
       fips  kir_natam_female_p1  kir_natam_female_p25  kir_natam_female_p50  \
0      1001                  NaN                   NaN                   NaN
1      1003               0.3436              0.343627              0.343645
2      1005                  NaN                   NaN                   NaN
3      1007                  NaN                   NaN                   NaN
4      1009                  NaN                   NaN                   NaN
...     ...                  ...                   ...                   ...
3214  72145                  NaN                   NaN                   NaN
3215  72147                  NaN                   NaN                   NaN
3216  72149                  NaN                   NaN                   NaN
3217  72151                  NaN                   NaN                   NaN
3218  72153                  NaN                   NaN                   NaN

      kir_natam_female_p75  kir_natam_female_p100  kir_natam_female_n  \
0                      NaN                    NaN                 NaN
1                 0.343667               0.343722                42.0
2                      NaN                    NaN                 NaN
3                      NaN                    NaN                 NaN
4                      NaN                    NaN                 NaN
...                    ...                    ...                 ...
3214                   NaN                    NaN                 NaN
3215                   NaN                    NaN                 NaN
3216                   NaN                    NaN                 NaN
3217                   NaN                    NaN                 NaN
3218                   NaN                    NaN                 NaN

      kir_natam_female_mean  jail_natam_female_p1  jail_natam_female_p25  ...  \
0                       NaN                   NaN                    NaN  ...
1                  0.341199             -0.010921              -0.006502  ...
2                       NaN                   NaN                    NaN  ...
3                       NaN                   NaN                    NaN  ...
4                       NaN                   NaN                    NaN  ...
...                     ...                   ...                    ...  ...
3214                    NaN                   NaN                    NaN  ...
3215                    NaN                   NaN                    NaN  ...
3216                    NaN                   NaN                    NaN  ...
3217                    NaN                   NaN                    NaN  ...
3218                    NaN                   NaN                    NaN  ...

      coll_white_pooled_mean_se  comcoll_white_pooled_mean_se  \
```

```
0                        0.020800                       0.021270
1                        0.014500                       0.014719
2                        0.035349                       0.038319
3                        0.040235                       0.040666
4                        0.018691                       0.022029
...                           ...                            ...
3214                          NaN                            NaN
3215                          NaN                            NaN
3216                          NaN                            NaN
3217                          NaN                            NaN
3218                          NaN                            NaN

      somecoll_white_pooled_mean_se  hs_white_pooled_mean_se  \
0                          0.020339                 0.012137
1                          0.012726                 0.007792
2                          0.030695                 0.019642
3                          0.043610                 0.025271
4                          0.020685                 0.012227
...                             ...                      ...
3214                            NaN                      NaN
3215                            NaN                      NaN
3216                            NaN                      NaN
3217                            NaN                      NaN
3218                            NaN                      NaN

      wgflx_rk_white_pooled_mean_se  hours_wk_white_pooled_mean_se  \
0                          0.018251                       1.131328
1                          0.012266                       0.741782
2                          0.027699                       1.673795
3                          0.036064                       2.380808
4                          0.017313                       1.158366
...                             ...                            ...
3214                            NaN                            NaN
3215                            NaN                            NaN
3216                            NaN                            NaN
3217                            NaN                            NaN
3218                            NaN                            NaN

      kfr_native_white_pooled_mean_se  kir_native_white_pooled_mean_se  \
0                            0.008103                         0.008534
1                            0.005500                         0.005603
2                            0.013528                         0.013531
3                            0.016382                         0.016979
4                            0.007895                         0.008158
...                               ...                              ...
3214                              NaN                              NaN
3215                              NaN                              NaN
```

```
3216                                      NaN                                      NaN
3217                                      NaN                                      NaN
3218                                      NaN                                      NaN

      kir_imm_white_pooled_mean_se  kfr_imm_white_pooled_mean_se
0                         0.057445                      0.058009
1                         0.041219                      0.037302
2                              NaN                           NaN
3                              NaN                           NaN
4                              NaN                           NaN
...                            ...                           ...
3214                           NaN                           NaN
3215                           NaN                           NaN
3216                           NaN                           NaN
3217                           NaN                           NaN
3218                           NaN                           NaN

[3219 rows x 10826 columns]
```

Now we need to select the correct fields to keep in the dataframe. From the documentation:

`[outcome]_[race]_[gender]_p[pctile]: Mean predicted outcome for children of a given race, gender and with parents at a given percentile in the national household income distribution.`

`kir: Mean percentile rank (relative to other children born in the same year) in the national distribution of individual income (i.e. just own earnings) measured as mean earnings in 2014-2015 for the baseline sample`

And we can infer that `pooled: All possible values in the field included`

Therefore we need the column that represents the mean percentile rank of all races and all genders in the 25th percentile, which by definition is `kir_pooled_pooled_p25`.

[16]:
```
rank_df = opp_df.loc[:, ["fips", "kir_pooled_pooled_p25"]].copy()
rank_df.rename({"kir_pooled_pooled_p25": "county_rank_25"}, axis=1, inplace=True)
rank_df
```

[16]:
```
       fips  county_rank_25
0      1001        0.384716
1      1003        0.407555
2      1005        0.397180
3      1007        0.380409
4      1009        0.383874
...     ...             ...
3214  72145        0.367308
3215  72147             NaN
3216  72149        0.286394
3217  72151        0.392320
3218  72153        0.405954
```

```
[3219 rows x 2 columns]
```

Let's merge the two dataframes so we have the county code, name, EC and rank in one place.

```
[17]: ec_rank_df = social_capital_df.merge(rank_df,
                             how="inner",
                             left_on="county",
                             right_on="fips")

      ec_rank_df.drop("fips", axis=1, inplace=True)
      ec_rank_df = ec_rank_df.loc[:, ["county", "county_name", "ec_county",␣
       ↪"county_rank_25", "pop2018"]]
      ec_rank_df
```

```
[17]:      county          county_name  ec_county  county_rank_25   pop2018
      0       1001      Autauga, Alabama    0.72077        0.384716   55200.0
      1       1003      Baldwin, Alabama    0.74313        0.407555  208107.0
      2       1005      Barbour, Alabama    0.41366        0.397180   25782.0
      3       1007         Bibb, Alabama    0.63152        0.380409   22527.0
      4       1009       Blount, Alabama    0.72562        0.383874   57645.0
      ...      ...                  ...        ...             ...       ...
      3084   56037  Sweetwater, Wyoming    0.96235        0.470544   44117.0
      3085   56039       Teton, Wyoming    1.07623        0.501737   23059.0
      3086   56041       Uinta, Wyoming    0.95452        0.455938   20609.0
      3087   56043    Washakie, Wyoming    0.90667        0.450778    8129.0
      3088   56045      Weston, Wyoming    0.97840        0.470659    7100.0

      [3089 rows x 5 columns]
```

And restrict our dataset to the 200 most populous counties.

```
[18]: ec_rank_df = ec_rank_df.sort_values(by="pop2018", ascending=False).head(200)

      ec_rank_df
```

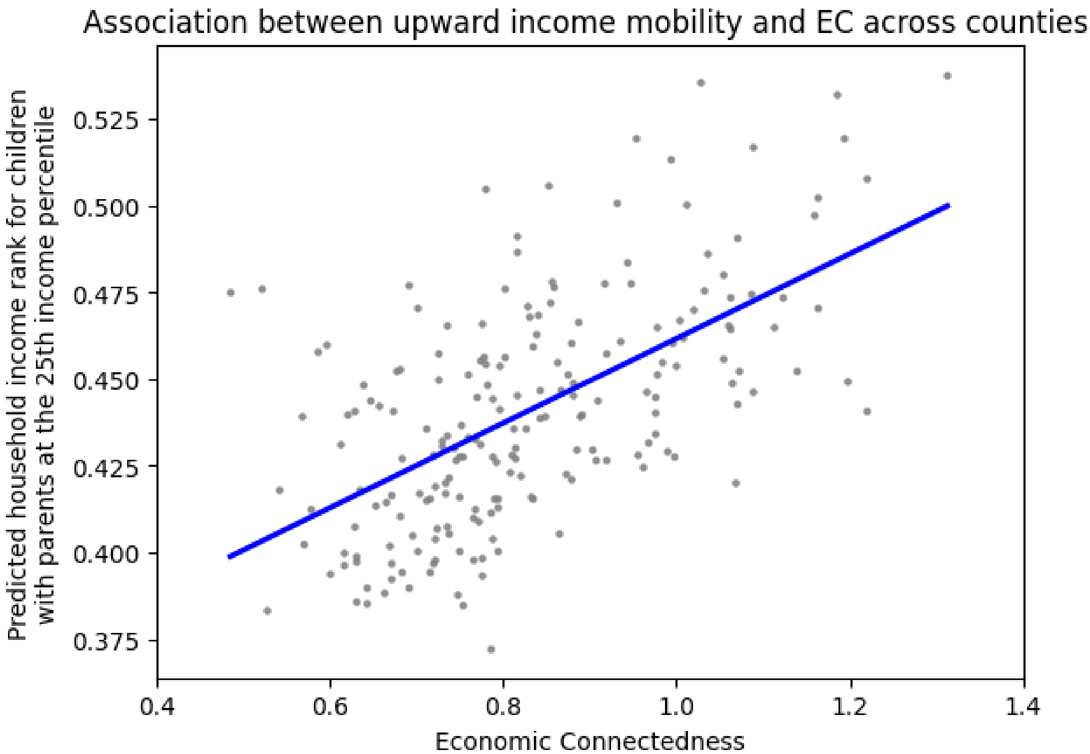
```
[18]:      county                 county_name  ec_county  county_rank_25     pop2018
      203     6037  Los Angeles, California    0.73580        0.465747  10098052.0
      605    17031          Cook, Illinois    0.75869        0.433307   5223719.0
      2598   48201           Harris, Texas    0.67668        0.452386   4602523.0
      102     4013       Maricopa, Arizona    0.74400        0.430274   4253913.0
      221     6073   San Diego, California    0.90846        0.443861   3302833.0
      ...      ...                     ...        ...             ...         ...
      2517   48039         Brazoria, Texas    0.83867        0.462893    353999.0
      357    12083         Marion, Florida    0.62977        0.397428    348371.0
      1310   27003       Anoka, Minnesota    1.03045        0.475523    347431.0
      2512   48027             Bell, Texas    0.77036        0.409022    342236.0
      2749   49011             Davis, Utah    1.13732        0.452445    340621.0
```

```
[200 rows x 5 columns]
```

Counties with no rank don't need any special handling since we will simply not put them in the graph. This is why we used an inner join.

Since we have all the information we need, it's time to start building our graph.

[19]:
```python
import seaborn as sns

ax = sns.regplot(
    x="ec_county",
    y="county_rank_25",
    data=ec_rank_df,
    color="grey",
    ci=None,
    scatter_kws ={"s": 5},
    line_kws={'color':'blue'}
    )

_ = ax.set(xlim=(0.4, 1.4),
       xlabel="Economic Connectedness",
       ylabel="Predicted household income rank for children\nwith parents at the␣
 ↪25th income percentile",
       aspect=4,
       title="Association between upward income mobility and EC across counties."
      )
```



And finally, we will include markers for 5 major counties so we can compare familiar names in our graph.

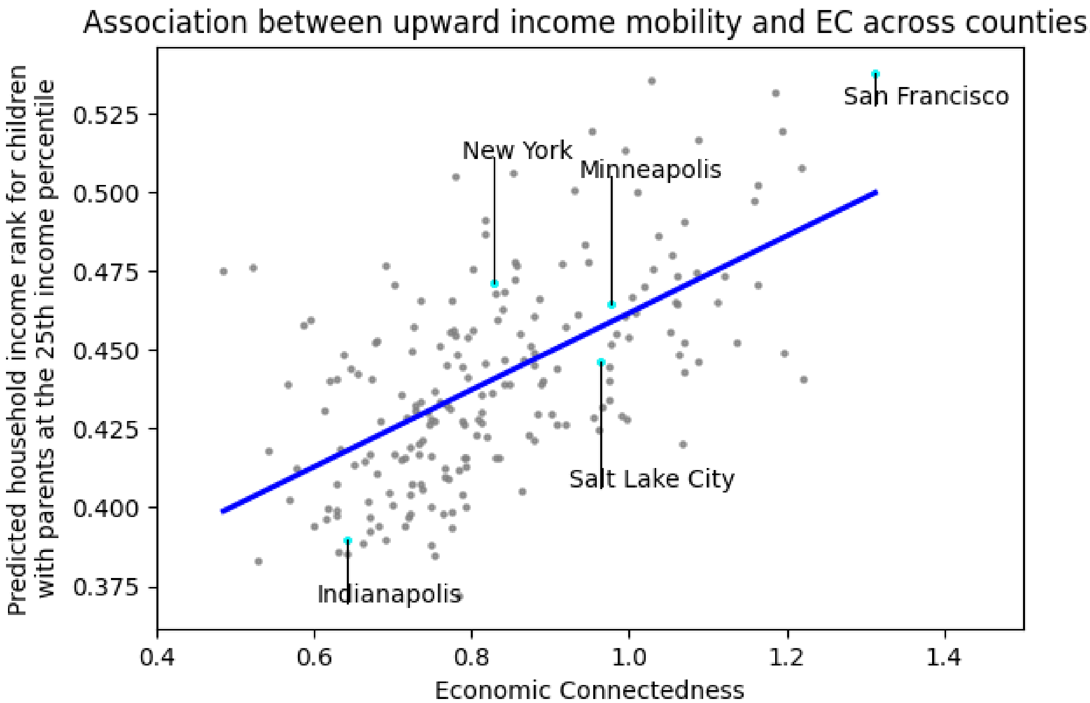
[20]:
```
special_counties_dict = {"18097": "Indianapolis",
                         "36061": "New York",
                         "49035": "Salt Lake City",
                         "27053": "Minneapolis",
                         "6075" : "San Francisco"}

query = lambda fips: fips in special_counties_dict
special_counties_df = ec_rank_df.loc[ec_rank_df.county.apply(query)]
special_counties_df
```

[20]:

	county	county_name	ec_county	county_rank_25	pop2018
1836	36061	New York, New York	0.82734	0.471015	1632480.0
1335	27053	Hennepin, Minnesota	0.97632	0.464788	1235478.0
2761	49035	Salt Lake, Utah	0.96395	0.446506	1120805.0
740	18097	Marion, Indiana	0.64282	0.389983	944523.0
222	6075	San Francisco, California	1.31244	0.537693	870044.0

[21]:
```
from matplotlib import pyplot as plt
```

```python
def add_tag(row, y_offset):
    x = row.ec_county
    y = row.county_rank_25
    text_x = x - 0.04
    text_y = y + y_offset

    # add city name
    ax.text(text_x, text_y, special_counties_dict.get(row.county))
    # add special marker
    plt.scatter(x=x, y=y, color="cyan", s=6)
    # plot line connecting the mark with the text
    plt.plot([x, x], [y, text_y], linewidth=0.8, color="black")


# We want to position each name in a different y-axis position
# in order for us to make sure it doesn't hide any data points
y_offset_dict = {"18097": -0.02,
                "36061": 0.04,
                "49035": -0.04,
                "27053": 0.04,
                "6075" : -0.01}

ax = sns.regplot(
    x="ec_county",
    y="county_rank_25",
    data=ec_rank_df,
    color="grey",
    ci=None,
    scatter_kws ={"s": 5},
    line_kws={'color':'blue'}
    )

ax.set(xlim=(0.4, 1.5),
       xlabel="Economic Connectedness",
       ylabel="Predicted household income rank for children\nwith parents at the␣
 ↪25th income percentile",
       aspect=4,
       title="Association between upward income mobility and EC across counties."
      )

for _, row in special_counties_df.iterrows():
    offset = y_offset_dict.get(row.county)
    add_tag(row, offset)
```



1.3 Q3: Upward Income Mobility, Economic Connectedness, and Median House Income

As always, we will start by importing data about the median household income by US ZIP Code. Unfortunately this data is not available on either the Social Capital or the Opportunity Atlas datasets.

After some digging we find the MCDC ZIP Code Lookup Application where we can find an Excel spreadsheet containing information about the US 2018 median family income.

We import our new dataset after converting the Excel spreadsheet to a csv file.

[22]:
```python
income_df = pd.read_csv("data/ZIP_codes_2018.csv")
income_df
```

[22]:
```
       ZIP Code      Type  State FIPS   Preferred name  \
0           501    unique          36   Holtsville, NY
1           544    unique          36   Holtsville, NY
2           601  standard          72     Adjuntas, PR
3           602  standard          72       Aguada, PR
4           603  standard          72    Aguadilla, PR
...         ...       ...         ...              ...
41271     99926    PO box           2   Metlakatla, AK
41272     99927    PO box           2  Point Baker, AK
41273     99928    PO box           2    Ward Cove, AK
```

```
41274     99929    PO box          2     Wrangell, AK
41275     99950    PO box          2    Ketchikan, AK

                                        Alternate names  Population (2018)   \
0                                    IRS Service Center                NaN
1                                    IRS Service Center                NaN
2      Colinas Del Gigante, Jard De Adjuntas, Urb San...            17,242
3      Alts De Aguada, Bo Guaniquilla, Comunidad Las ...            38,442
4      Ramey, Bda Caban, Bda Esteves, Bo Borinquen, B...            48,814
...                                                  ...               ...
41271                                                NaN             1,635
41272                                                NaN                38
41273                                                NaN               NaN
41274                                                NaN             2,484
41275                                   Edna Bay, Kasaan               NaN

       Housing units (2018)  Median family income (2018)  \
0                       NaN                          NaN
1                       NaN                          NaN
2                     7,176                      $14,433
3                    17,403                      $19,250
4                    24,311                      $19,718
...                     ...                          ...
41271                   548                      $65,313
41272                    78                          NaN
41273                   NaN                          NaN
41274                 1,450                      $71,923
41275                   NaN                          NaN

       MFI percentile (2018)   Latitude   Longitude  Land area  Water area
0                        NaN        NaN         NaN        NaN         NaN
1                        NaN        NaN         NaN        NaN         NaN
2                        0.0  18.181000  -66.750000     64.348       0.309
3                        0.0  18.362000  -67.176003     30.613       1.718
4                        0.0  18.455000  -67.120003     31.616       0.071
...                      ...        ...         ...        ...         ...
41271                   49.0  55.138000 -131.470001    132.798      82.369
41272                    NaN  56.238998 -133.457993    227.680       6.950
41273                    NaN        NaN         NaN        NaN         NaN
41274                   61.0  56.370998 -131.692993    999.999     246.117
41275                    NaN        NaN         NaN        NaN         NaN

[41276 rows x 13 columns]
```

The dataset used in Q1 and Q2 includes county-wide data instead of ZIP Code specific data, so we can't use it here.

Fortunately the Social Capital Atlas includes a separate dataset where the data is grouped by US

ZIP Codes.

[23]:
```
ec_df = pd.read_csv("data/social_capital_zip.csv", dtype={"county": str})
ec_df
```

[23]:
```
         zip county  num_below_p50  pop2018   ec_zip  ec_se_zip  nbhd_ec_zip  \
0       1001  25013     995.787468    17621  0.88157    0.02422      1.51095
1       1002  25015    1312.117077    30066  1.18348    0.02227      0.97760
2       1003  25015            NaN    11238  1.37536    0.05046          NaN
3       1005  25027     381.519745     4991  1.15543    0.03050      1.46491
4       1007  25015     915.396667    14967  1.19240    0.02046      1.17985
...      ...    ...            ...      ...      ...        ...          ...
23023  99901   2130    1192.299809    13818  0.99517    0.01776      0.88014
23024  99921   2198     365.768661     1986  0.87977    0.03071      0.74555
23025  99925   2198     154.513840      927      NaN        NaN          NaN
23026  99926   2198     311.014252     1635  0.87888    0.03618      0.81081
23027  99929   2275     313.282990     2484  1.06344    0.03122      0.88864

       ec_grp_mem_zip  ec_high_zip  ec_high_se_zip  ...  \
0             1.10210      1.47136         0.01599  ...
1             1.23333      1.62290         0.01500  ...
2             1.44359      1.65159         0.02898  ...
3             1.30756      1.47733         0.01664  ...
4             1.32294      1.56812         0.01364  ...
...               ...          ...             ...  ...
23023         0.95456      1.29659         0.01806  ...
23024         0.82996      1.18270         0.03593  ...
23025             NaN          NaN             NaN  ...
23026         0.83409      1.07167         0.04187  ...
23027         0.96641      1.32997         0.02900  ...

       exposure_grp_mem_high_zip  nbhd_exposure_zip  bias_grp_mem_zip  \
0                        1.45669            1.50590           0.02434
1                        1.53277            1.20282           0.09856
2                        1.57757                NaN           0.02482
3                        1.43769            1.46397           0.00850
4                        1.43019            1.23109          -0.01188
...                          ...                ...               ...
23023                    1.09039            0.94762           0.05710
23024                    1.04318            0.81680           0.06010
23025                        NaN                NaN               NaN
23026                    0.92952            0.80694           0.00877
23027                    1.07349            0.88926           0.01350

       bias_grp_mem_high_zip  nbhd_bias_zip  nbhd_bias_high_zip  \
0                   -0.10001       -0.00336            -0.21186
1                   -0.06421        0.18724            -0.24353
```

```
2                    -0.05143        NaN                  NaN
3                    -0.07246   -0.00064             -0.11397
4                    -0.11464    0.04162             -0.21283
...                       ...        ...                  ...
23023                -0.14293    0.07122             -0.21950
23024                -0.08759    0.08723             -0.14339
23025                     NaN        NaN                  NaN
23026                -0.07257   -0.00480             -0.09655
23027                -0.14883    0.00069             -0.24887

       clustering_zip  support_ratio_zip  volunteering_rate_zip  \
0            0.105720           0.945260                0.05650
1            0.103400           0.901630                0.14951
2            0.136500           0.769240                0.10501
3            0.105540           0.958370                0.15862
4            0.103910           0.948730                0.13053
...               ...                ...                    ...
23023        0.134730           0.997200                0.11883
23024        0.155610           0.997520                0.08404
23025        0.146579           0.992298                0.12396
23026        0.252740           1.000000                0.14291
23027        0.165580           1.000000                0.10700

       civic_organizations_zip
0                     0.010800
1                     0.036880
2                     0.080500
3                     0.021630
4                     0.016900
...                        ...
23023                 0.029990
23024                 0.032150
23025                 0.027728
23026                 0.011250
23027                 0.042480

[23028 rows x 23 columns]
```

We combine our datasets by ZIP Code…

[24]:
```
ec_income_df = ec_df.merge(income_df,
                                how="inner",
                                left_on="zip",
                                right_on="ZIP Code")

ec_income_df.rename(columns={"Median family income (2018)": "income", "ec_zip":␣
 ↪"ec"},
```

```
                    inplace=True)

# select the columns with the relevant data
ec_income_df = ec_income_df.loc[:, ["zip", "county", "ec", "income"]]
ec_income_df
```

[24]:
```
         zip  county       ec    income
0       1001   25013  0.88157   $88,797
1       1002   25015  1.18348   $98,977
2       1003   25015  1.37536       NaN
3       1005   25027  1.15543  $104,435
4       1007   25015  1.19240  $108,210
...      ...     ...      ...       ...
23023  99901    2130  0.99517   $85,295
23024  99921    2198  0.87977   $78,958
23025  99925    2198      NaN   $74,091
23026  99926    2198  0.87888   $65,313
23027  99929    2275  1.06344   $71,923

[23028 rows x 4 columns]
```

...but there's an issue. Our income dataset lists the income as a string starting with a dollar sign. We will convert its values to float before proceeding.

[25]:
```
# drop all rows that don't have all the required data for the plot
ec_income_df.dropna(how="any", inplace=True)

ec_income_df.income = ec_income_df.income.apply(str)\
                                            .apply(lambda money: money[1:].
 ↪replace(",", ""))\
                                            .apply(float)

ec_income_df.income
```

[25]:
```
0         88797.0
1         98977.0
3        104435.0
4        108210.0
6         92841.0
           ...
23022     84688.0
23023     85295.0
23024     78958.0
23026     65313.0
23027     71923.0
Name: income, Length: 18895, dtype: float64
```

At this point our dataset contains information about the EC and the rank of a given ZIP Code.

What we are missing is the 3rd dimension of our graph; the county rank for each zip code, for the 25th percentile of parents.

This information is already available to us because of the dataset we created for Q2, where we ranked each county according to the 25th income percentile of parents. We just have to merge it to our new dataset.

[26]:
```
ec_income_p25_df = ec_income_df.merge(rank_df,
                                      how="inner",
                                      left_on="county",
                                      right_on="fips")
ec_income_p25_df.drop("fips", axis=1, inplace=True)

ec_income_p25_df
```

[26]:

	zip	county	ec	income	county_rank_25
0	1001	25013	0.88157	88797.0	0.440873
1	1010	25013	0.73856	92841.0	0.440873
2	1013	25013	0.69744	50963.0	0.440873
3	1020	25013	0.72701	70974.0	0.440873
4	1022	25013	0.79394	51650.0	0.440873
...	...	...	...	...	...
18890	99840	2230	1.11489	84688.0	0.544405
18891	99901	2130	0.99517	85295.0	0.456915
18892	99921	2198	0.87977	78958.0	0.379764
18893	99926	2198	0.87888	65313.0	0.379764
18894	99929	2275	1.06344	71923.0	0.439716

[18895 rows x 5 columns]

The graph doesn't present our 3rd dimension as a continuous numerical value, but rather as distinct categories based on that value.

To replicate this we will split the income ranks into bins and then attach them to our dataset as categorical variables.

[27]:
```
ec_income_p25_df.county_rank_25 = ec_income_p25_df.county_rank_25.apply(lambda␣
↪rank: rank*100)

bins = pd.cut(ec_income_p25_df.county_rank_25,
              bins=[-float("inf"), 38, 42, 44, 48, float("inf")],
              labels=["<38", "38-42", "42-44", "44-48", ">48"])
bins
```

[27]:
```
0        44-48
1        44-48
2        44-48
3        44-48
4        44-48
```

```
        ...
18890      >48
18891    44-48
18892      <38
18893      <38
18894    42-44
Name: county_rank_25, Length: 18895, dtype: category
Categories (5, object): ['<38' < '38-42' < '42-44' < '44-48' < '>48']
```

[28]:

```
ec_income_p25_df["category"] = bins
ec_income_p25_df
```

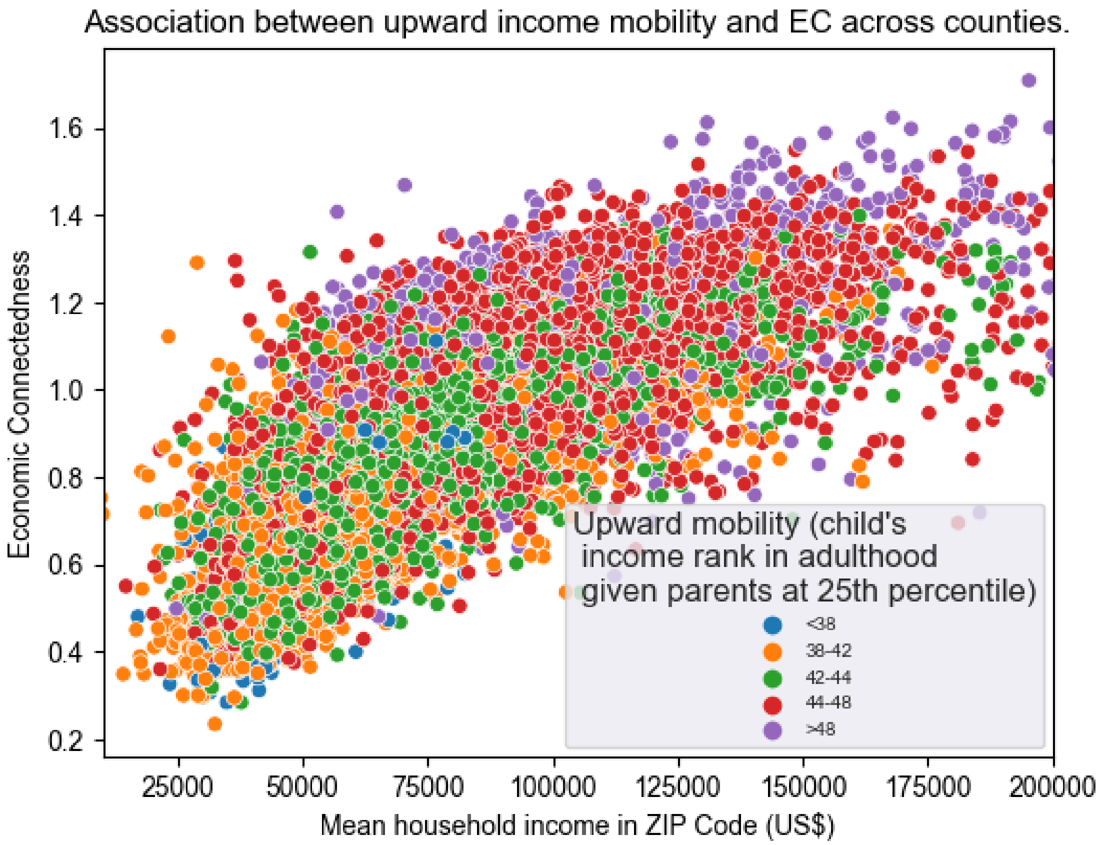
[28]:

```
         zip county       ec   income  county_rank_25 category
0       1001  25013  0.88157  88797.0       44.087276    44-48
1       1010  25013  0.73856  92841.0       44.087276    44-48
2       1013  25013  0.69744  50963.0       44.087276    44-48
3       1020  25013  0.72701  70974.0       44.087276    44-48
4       1022  25013  0.79394  51650.0       44.087276    44-48
...      ...    ...      ...      ...             ...      ...
18890  99840   2230  1.11489  84688.0       54.440475      >48
18891  99901   2130  0.99517  85295.0       45.691451    44-48
18892  99921   2198  0.87977  78958.0       37.976360      <38
18893  99926   2198  0.87888  65313.0       37.976360      <38
18894  99929   2275  1.06344  71923.0       43.971640    42-44

[18895 rows x 6 columns]
```

And finally, we can plot our graph!

[29]:

```
ax = sns.scatterplot(x="income",
                     y="ec",
                     hue="category",
                     data=ec_income_p25_df)

ax.set(xlim=(10000, 200000),
       ylabel="Economic Connectedness",
       xlabel="Mean household income in ZIP Code (US$)",
       title="Association between upward income mobility and EC across counties.
 ↪")

sns.set(rc={"figure.figsize":(10, 10)})

#configure legend
legend = plt.legend(title="Upward mobility (child's\n income rank in adulthood\n␣
 ↪given parents at 25th percentile)",
          prop={"size": 7}, loc="lower right")
```



1.4 Q4: Friending Bias and Exposure by High School

It's time to use another dataset provided by the Social Capital Dataset, that being the high school dataset.

[30]:
```
high_school_df = pd.read_csv("data/social_capital_high_school.csv")
high_school_df
```

[30]:
```
      high_school                     high_school_name    zip  county  \
0        00000044           Holy Spirit Catholic School  35405    1125
1        00000226             John Carroll Catholic HS  35209    1073
2        00000237      Holy Family Cristo Rey Catholic HS  35218    1073
3        00000714  Montgomery Catholic Preparatory School  36116    1101
4        00000758           St Paul's Episcopal School  36608    1097
...           ...                                  ...    ...     ...
17520    Y2121679     St Agnes Academy-St Dominic School  38117   47157
17521    Z0516931                         Sayre School  40507   21067
17522    Z1326859         Fort Worth Christian School  76180   48439
17523    Z1326892               Second Baptist School  77057   48201
17524    Z1328448                   The Kinkaid School  77024   48201
```

```
       students_9_to_12  ec_own_ses_hs  ec_own_ses_se_hs  ec_parent_ses_hs  \
0                   158            NaN               NaN               NaN
1                   538        1.52901           0.04220           1.43847
2                   229        0.66359           0.07105               NaN
3                   363        1.56551           0.05799               NaN
4                   409        1.62628           0.04533           1.57592
...                 ...            ...               ...               ...
17520               350            NaN               NaN               NaN
17521               258            NaN               NaN               NaN
17522               327            NaN               NaN               NaN
17523               338            NaN               NaN               NaN
17524               588            NaN               NaN               NaN

       ec_parent_ses_se_hs  ec_high_own_ses_hs  ...  ec_high_parent_ses_hs  \
0                      NaN                 NaN  ...                    NaN
1                  0.05073             1.64439  ...                1.46086
2                      NaN             0.87627  ...                    NaN
3                      NaN             1.60898  ...                    NaN
4                  0.05254             1.72722  ...                1.60072
...                    ...                 ...  ...                    ...
17520                  NaN                 NaN  ...                    NaN
17521                  NaN                 NaN  ...                    NaN
17522                  NaN                 NaN  ...                    NaN
17523                  NaN                 NaN  ...                    NaN
17524                  NaN                 NaN  ...                    NaN

       ec_high_parent_ses_se_hs  exposure_own_ses_hs  exposure_parent_ses_hs  \
0                           NaN                  NaN                     NaN
1                       0.04742              1.50707                 1.44259
2                           NaN              0.65517                     NaN
3                           NaN              1.49000                     NaN
4                       0.04730              1.62275                 1.57514
...                         ...                  ...                     ...
17520                       NaN                  NaN                     NaN
17521                       NaN                  NaN                     NaN
17522                       NaN                  NaN                     NaN
17523                       NaN                  NaN                     NaN
17524                       NaN                  NaN                     NaN

       bias_own_ses_hs  bias_parent_ses_hs  bias_high_own_ses_hs  \
0                  NaN                 NaN                   NaN
1             -0.01456             0.00285              -0.09112
2             -0.01286                 NaN              -0.33747
3             -0.05068                 NaN              -0.07985
4             -0.00217            -0.00050              -0.06438
...                ...                 ...                   ...
```

```
17520                NaN                  NaN                     NaN
17521                NaN                  NaN                     NaN
17522                NaN                  NaN                     NaN
17523                NaN                  NaN                     NaN
17524                NaN                  NaN                     NaN

       bias_high_parent_ses_hs  clustering_hs  volunteering_rate_hs
0                          NaN       0.693142              0.086807
1                     -0.01266       0.604580              0.069540
2                          NaN       0.686860              0.051010
3                          NaN       0.673730              0.042280
4                     -0.01624       0.623290              0.060610
...                        ...            ...                   ...
17520                      NaN       0.644070              0.077204
17521                      NaN       0.740327              0.092056
17522                      NaN       0.680769              0.053181
17523                      NaN       0.692155              0.050045
17524                      NaN       0.643250              0.047230

[17525 rows x 21 columns]
```

The graph requires 2 columns: The "Friending bias among low-parental SES students" and the "Share of high-parental-SES-Students".

We will go through the dataset's documentation to locate these two quantities in our dataset.

Friendling Bias From the Figure's comments:

`Friending bias is defined as one minus the mean ratio of the share of high-school friends with high parental SES to the share of high-school peers with high parental SES, averaging over students with low parental SES`

The mathematical definition for a given high school h with N students would be: $friending\_bias(h) = 1 - \frac{\sum_{student \in h} \frac{share\_friends(student)}{share\_peers(student)}}{N}$, where:

$share\_friends(x)$ is the share of high-parental-SES friends for a given student x, $share\_peers(x)$ is the share of high-parental-SES peers for a given student x.

(Keep in mind we are talking about the friending bias for a given high school, not a single student. This is why the definition includes the average, which is absent in equation (4) of the original paper).

From the documentation:

`exposure_parent_ses_hs: Mean exposure to high-parental-SES individuals by high school for low-parental-SES individuals: two times the average share of highparental-SES individuals within three birth cohorts, averaged over low-parental-SES users.`

Which means $ec\_high\_parent\_ses\_hs(h) = 2 * friending\_bias(h) \iff friending\_bias(h) = \frac{ec\_high\_parent\_ses\_hs(h)}{2}$

Share of high-parental-SES Students From the documentation:

`bias_parent_ses_hs: ec_parent_ses hs divided by exposure_parent_ses_hs, all subtracted from one`

This seems to be the definition we are looking for. It doesn't hurt that, according to the documentation, this is the only other variable used by the Figure we are trying to recreate.

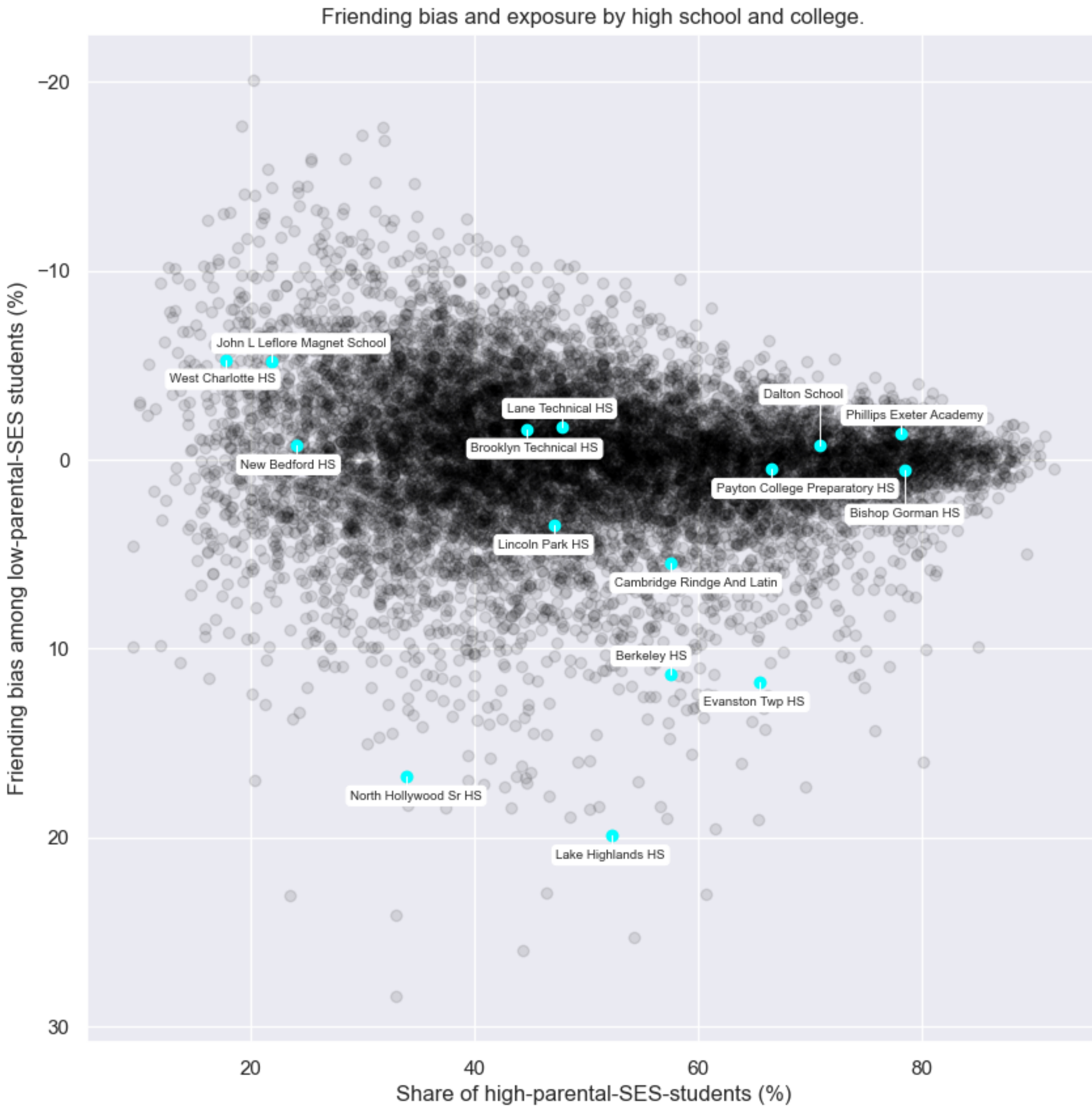
As an aside, we should also multiply the columns by 100 since on the graph they represent % probabilities.

[31]:
```python
columns = ["high_school", "high_school_name", "exposure_parent_ses_hs",␣
↪"bias_parent_ses_hs"]
friend_bias_df = high_school_df.loc[:, columns]
friend_bias_df.dropna(how="any", inplace=True)

friend_bias_df.exposure_parent_ses_hs = friend_bias_df.exposure_parent_ses_hs *␣
↪100 / 2
friend_bias_df.bias_parent_ses_hs = friend_bias_df.bias_parent_ses_hs * 100

friend_bias_df
```

[31]:
```
      high_school             high_school_name  exposure_parent_ses_hs  \
1        00000226      John Carroll Catholic HS               72.129500
4        00000758    St Paul's Episcopal School               78.757000
9        00000973        Mars Hill Bible School               62.216995
33       00002788     Alabama Christian Academy               73.492000
37       00030942         Salpointe Catholic HS               72.346505
...           ...                          ...                     ...
17501    X1932087            Bob Jones Academy               68.537005
17510    Y0537452                   Jesuit HS               72.664500
17511    Y0538208           Archbishop Shaw HS               58.547995
17512    Y0539347                 Catholic HS               78.136500
17519    Y2120358  St Michael The Archangel HS               77.722500

       bias_parent_ses_hs
1                   0.285
4                  -0.050
9                  -0.565
33                 -2.436
37                  1.845
...                   ...
17501              -2.587
17510               1.170
17511              -1.223
17512               0.796
17519               0.519

[11608 rows x 4 columns]
```

Now that we have prepared the columns, we just need to make a plot and highlight the high schools from the graph.

[32]:
```
special_hs = ["00941729", "060474000432", "170993000942",
              "170993001185", "170993003989", "171449001804",
              "250327000436", "360009101928", "370297001285",
              "483702004138", "250843001336", "062271003230",
              "010237000962", "00846981", "00852124"]

# The titles of some of these high schools collide in the graph,
# therefore we will put them over instead of under their markers
y_offset_map = {name : 1.2 if name not in
                ["170993000942", "010237000962", "060474000432",
                 "00852124"] else -0.8
                for name in special_hs}
y_offset_map["00941729"] = -2.5
y_offset_map["00846981"] = 2.5

def add_mark(x, y, title, y_offset):
    x = row[2]
    y = row[3]
    text_y = y + y_offset

    plt.scatter(x=x, y=y, color="cyan")
    props = dict(boxstyle='round', facecolor='white', alpha=1)
    # plot line connecting the mark with the text
    plt.plot([x, x], [y, text_y], linewidth=0.8, color="white")
    plt.text(x - 5, text_y, row[1], fontsize=7, bbox=props)


plt.gca().invert_yaxis()

ax = plt.scatter(x=friend_bias_df.exposure_parent_ses_hs,
            y=friend_bias_df.bias_parent_ses_hs,
            alpha=0.1,
            color="black")

query = lambda hs_code: hs_code in special_hs
for _ , row in friend_bias_df.loc[friend_bias_df.high_school.apply(query), :].
↪iterrows():
    add_mark(row[2], row[3], row[1], y_offset_map.get(row[0]))

plt.title("Friending bias and exposure by high school and college.")
plt.ylabel("Friending bias among low-parental-SES students (%)")
plt.xlabel("Share of high-parental-SES-students (%)")
plt.show()
```



1.5 Q5: Friending Bias vs. Racial Diversity

1.5.1 US Colleges Racial Diversity

First of all we need to acquire data about the racial composition of US universities. The dataset cited in the original research (2013 Integrated Post-Secondary Education Data System (IPEDS)) can be found here.

[33]:
```
uni_race_df = pd.read_csv("data/college_race.csv")

uni_race_df
```

```
[33]:        unitid  year fips_ipeds                          inst_name  slevel  \
      0      100760  2009    Alabama  Central Alabama Community College  2-year
      1      100760  2010    Alabama  Central Alabama Community College  2-year
      2      100760  2011    Alabama  Central Alabama Community College  2-year
      3      100760  2012    Alabama  Central Alabama Community College  2-year
      4      100760  2013    Alabama  Central Alabama Community College  2-year
      ...       ...   ...        ...                                ...     ...
      16339  240718  2010    Wyoming                    Wyotech-Laramie  2-year
      16340  240718  2011    Wyoming                    Wyotech-Laramie  2-year
      16341  240718  2012    Wyoming                    Wyotech-Laramie  2-year
      16342  240718  2013    Wyoming                    Wyotech-Laramie  2-year
      16343  240718  2014    Wyoming                    Wyotech-Laramie  2-year

                        twocat  public  forprofit  total_enrollment  col_white  ...  \
      0         Public 2-year       1          0            2288.0   70.49825  ...
      1         Public 2-year       1          0            2341.0   68.68860  ...
      2         Public 2-year       1          0            2338.0   67.32250  ...
      3         Public 2-year       1          0            1898.0   65.43730  ...
      4         Public 2-year       1          0            1779.0   67.84710  ...
      ...                  ...     ...        ...               ...        ...  ...
      16339  For-Profit 2-year       0          1            1879.0   75.41245  ...
      16340  For-Profit 2-year       0          1            1600.0   75.50000  ...
      16341  For-Profit 2-year       0          1            1808.0   72.12389  ...
      16342  For-Profit 2-year       0          1            1462.0   69.69904  ...
      16343  For-Profit 2-year       0          1            1085.0   69.86176  ...

             dif_asian  col_amind  mkt_amind  dif_amind  col_pacis  mkt_pacis  \
      0            NaN   0.305944        NaN        NaN        NaN        NaN
      1      -0.250846   0.512601   0.384398   0.128203   0.000000   0.064605
      2       0.003510   0.513259   0.157998   0.355261   0.042772   0.000000
      3       0.263435   0.368809   0.215630   0.153179   0.000000   0.000000
      4       0.008670   0.224845   0.278752  -0.053907   0.112423   0.000000
      ...          ...        ...        ...        ...        ...        ...
      16339  -1.160573   4.364023   0.634790   3.729234   0.000000   0.070074
      16340  -2.339696   5.250000   0.471543   4.778456   0.187500   0.131212
      16341  -2.935405   4.590708   0.558227   4.032482   0.221239   0.114090
      16342  -3.295658   3.898769   0.535388   3.363381   0.547196   0.073008
      16343  -3.388090   3.686636   0.230638   3.455997   1.105991   0.048556

             dif_pacis  col_twora  mkt_twora  dif_twora
      0            NaN   0.000000        NaN        NaN
      1      -0.064605   0.000000   0.877009  -0.877009
      2       0.042772   0.000000   0.867378  -0.867378
      3       0.000000   0.000000   0.753769  -0.753769
      4       0.112423   0.056211   0.754717  -0.698506
      ...          ...        ...        ...        ...
      16339  -0.070074   0.000000   2.275350  -2.275350
```

```
16340  0.056288  0.437500  2.201902  -1.764403
16341  0.107149  1.272124  2.477386  -1.205262
16342  0.474188  2.667579  2.202393   0.465186
16343  1.057435  3.225806  2.512746   0.713061

[16344 rows x 30 columns]
```

The dataset contains information about the enrollment of different races into the US's public and private universities. We need to extract the universities' county's FIPS code, and their Herfindahl-Hirschman Index from them.

To calculate the HHI index we need to subtract 1 from the sum of all the columns representing race fractions each raised to the power of 2.

[34]:
```
p_2 = lambda arr : np.power(arr / 100, 2)
uni_race_df = uni_race_df.loc[uni_race_df.year == 2013].copy()

uni_race_df["hhi"] = 1 - (p_2(uni_race_df.col_white) + p_2(uni_race_df.
 ↪col_hispa) +\
                          p_2(uni_race_df.col_black) + p_2(uni_race_df.
 ↪col_asian) +\
                          p_2(uni_race_df.col_amind) + p_2(uni_race_df.
 ↪col_pacis) +\
                          p_2(uni_race_df.col_twora))
uni_race_df.hhi
```

[34]:
```
4        0.452768
13       0.585278
26       0.515541
35       0.446589
44       0.493447
           ...
16306    0.345499
16315    0.295868
16324    0.218290
16333    0.300614
16342    0.486521
Name: hhi, Length: 1972, dtype: float64
```

Now we import the college social capital dataset and merge it with the race composition one.

[35]:
```
college_capital_df = pd.read_csv("data/social_capital_college.csv")
college_capital_df
```

[35]:
```
      college                   college_name    zip  county  \
0      100200         Alabama A & M University  35762    1089
1      100300             Faulkner University  36109    1101
2      100400         University of Montevallo  35115    1117
3      100500         Alabama State University  36104    1101
```

```
4      100700        Central Alabama Community College  35010    1123
...       ...                                      ...    ...     ...
2581  4254400  Arkansas State University-Mountain Home  72653    5005
2582  4263400           Florida Polytechnic University  33805   12105
2583  4263600               Northeast Lakeview College  78145   48029
2584  4281700                          Compton College  90221    6037
2585  4283700           Oregon Coast Community College  97366   41041

      mean_students_per_cohort  ec_own_ses_college  ec_own_ses_se_college  \
0                   943.666667             0.85678                0.02233
1                   227.666667             1.30964                0.04869
2                   494.000000             1.42378                0.03040
3                          NaN             0.77916                0.01937
4                          NaN             0.72742                0.03504
...                        ...                 ...                    ...
2581                       NaN             0.88695                0.04674
2582                       NaN                 NaN                    NaN
2583                       NaN             1.28254                0.05277
2584                       NaN             0.71178                0.06780
2585                       NaN             0.69457                0.07714

      ec_parent_ses_college  ec_parent_ses_se_college  \
0                   0.67629                   0.03241
1                   1.26671                   0.05812
2                   1.15413                   0.03638
3                   0.67090                   0.03038
4                   0.77238                   0.04497
...                     ...                       ...
2581                0.52927                   0.05098
2582                1.20327                   0.09919
2583                1.17784                   0.06483
2584                    NaN                       NaN
2585                    NaN                       NaN

      ec_high_own_ses_college  ...  ec_high_parent_ses_se_college  \
0                     1.12202  ...                        0.03498
1                     1.54639  ...                        0.05134
2                     1.57365  ...                        0.03395
3                     1.04811  ...                        0.03201
4                     0.98888  ...                        0.04984
...                       ...  ...                            ...
2581                  1.00103  ...                        0.05764
2582                      NaN  ...                        0.09509
2583                  1.41132  ...                        0.06000
2584                  0.81637  ...                            NaN
2585                  0.80913  ...                            NaN
```

```
      exposure_own_ses_college  exposure_parent_ses_college  \
0                      0.84662                      0.65090
1                      1.23776                      1.20183
2                      1.41664                      1.17101
3                      0.75162                      0.65297
4                      0.76579                      0.76786
...                        ...                          ...
2581                   0.89316                      0.49553
2582                       NaN                      1.19730
2583                   1.36033                      1.17411
2584                   0.72474                          NaN
2585                   0.71267                          NaN

      bias_own_ses_college  bias_parent_ses_college  \
0                 -0.01200                 -0.03900
1                 -0.05807                 -0.05398
2                 -0.00504                  0.01442
3                 -0.03664                 -0.02747
4                  0.05010                 -0.00589
...                    ...                      ...
2581               0.00695                 -0.06810
2582                   NaN                 -0.00499
2583               0.05718                 -0.00318
2584               0.01789                      NaN
2585               0.02539                      NaN

      bias_high_own_ses_college  bias_high_parent_ses_college  \
0                      -0.32529                      -0.14036
1                      -0.24935                      -0.12001
2                      -0.11083                      -0.05979
3                      -0.39448                      -0.12802
4                      -0.29133                      -0.13139
...                         ...                           ...
2581                   -0.12077                      -0.15805
2582                        NaN                      -0.03957
2583                   -0.03748                      -0.06948
2584                   -0.12643                           NaN
2585                   -0.13536                           NaN

      clustering_college  support_ratio_college  volunteering_rate_college
0                0.24470                0.99483                    0.03256
1                0.40754                0.99481                    0.03336
2                0.30921                0.99683                    0.09566
3                0.23222                0.99485                    0.02150
4                0.34104                0.99271                    0.02922
...                  ...                    ...                        ...
2581             0.32144                0.99446                    0.06755
```

```
2582             0.48909              0.99920                0.04523
2583             0.24113              0.90760                0.03251
2584             0.21260              0.82709                0.02312
2585             0.34485              0.97277                0.12013

[2586 rows x 22 columns]
```

Unfortunately, the social capital dataset's college codes are rounded to 100 for some incomprehensible reason. As such we will try to merge on the colleges' names since they are (as far as we can see from the data) fairly standardized.

[36]:
```python
race_capital_df = uni_race_df.merge(college_capital_df,
                  how="inner",
                  right_on="college_name",
                  left_on="inst_name")

columns = ["college", "college_name", "hhi", "bias_parent_ses_college",␣
 ↪"mean_students_per_cohort"]
race_capital_df = race_capital_df.loc[:, columns]
race_capital_df
```

[36]:
```
     college                                   college_name       hhi  \
0     100700              Central Alabama Community College  0.452768
1    1218200          Chattahoochee Valley Community College  0.585278
2     101500            Enterprise State Community College  0.515541
3     101700               Gadsden State Community College  0.493447
4     787100  George C Wallace State Community College-Hance...  0.230032
..       ...                                            ...       ...
950   728900                        Central Wyoming College  0.461718
951   392900                        Eastern Wyoming College  0.236304
952   925900                 Laramie County Community College  0.345499
953   393100                              Northwest College  0.295868
954   393300              Western Wyoming Community College  0.300614

     bias_parent_ses_college  mean_students_per_cohort
0                   -0.00589                       NaN
1                    0.05260                263.000000
2                   -0.03047                289.000000
3                   -0.00233                819.333333
4                    0.00578                923.000000
..                       ...                       ...
950                 -0.01448                238.000000
951                 -0.05346                205.000000
952                  0.00592                489.666667
953                  0.01465                337.000000
954                 -0.01671                430.000000

[955 rows x 5 columns]
```

To make sure our merge is valid we can check the intersection between the college names of the two datasets:

[37]:
```
np.intersect1d(college_capital_df.college_name, uni_race_df.inst_name).shape
```

[37]:
```
(926,)
```

... and find out that there are 29 duplicates in the merged set.

We will clear the duplicate and NA values in the mean_students_per_cohort column in order to have valid data for our weighted HHI computation.

We also need the mean of the y-axis, which in this case is the `bias_parent_ses_college`.

[38]:
```
race_capital_df.drop_duplicates(subset="college_name", inplace=True)
race_capital_df.shape
```

[38]:
```
(926, 5)
```

[39]:
```
# Drop non-computable rows
race_capital_df.dropna(subset="mean_students_per_cohort", inplace=True)
# Make the bias a % column
race_capital_df.bias_parent_ses_college = race_capital_df.
 ↪bias_parent_ses_college * 100
```

The following cells are dedicated to computing the figure's x-axis value, the weighted racial diversity (HHI) index.

The weighted HHI in mathematical terms for a given college c is defined as: $Weighted\_HHI(c) = \frac{HHI(c) * mean\_pop(c)}{\sum\limits_{other\_c \in perc(c)} mean\_pop(other\_c)}$,

where $perc(c)$ is the set of all colleges in c's percentile and $mean\_pop(c)$ is the mean students per cohort for college c.

Keep in mind of course that the figure's x-axis plots the dataset's *percentiles* and not each college individually.

For that reason we will tag each college with its percentile rank.

[40]:
```
bins = pd.cut(race_capital_df.hhi, bins=20, labels=np.arange(0, 100, 5)) #␣
 ↪divide into 20 percentile bins
race_capital_df["percentile"] = bins
race_capital_df
```

[40]:
```
    college                                  college_name       hhi  \
1   1218200           Chattahoochee Valley Community College  0.585278
2    101500               Enterprise State Community College  0.515541
3    101700                   Gadsden State Community College  0.493447
4    787100  George C Wallace State Community College-Hance...  0.230032
6    102200                 Jefferson State Community College  0.475184
..      ...                                           ...       ...
```

```
950   728900                           Central Wyoming College  0.461718
951   392900                           Eastern Wyoming College  0.236304
952   925900                  Laramie County Community College  0.345499
953   393100                                 Northwest College  0.295868
954   393300                 Western Wyoming Community College  0.300614

     bias_parent_ses_college  mean_students_per_cohort percentile
1                      5.260                263.000000         60
2                     -3.047                289.000000         55
3                     -0.233                819.333333         50
4                      0.578                923.000000         15
6                      3.767               1268.666667         50
..                       ...                       ...        ...
950                   -1.448                238.000000         45
951                   -5.346                205.000000         20
952                    0.592                489.666667         30
953                    1.465                337.000000         25
954                   -1.671                430.000000         25

[781 rows x 6 columns]
```

And compute the two terms of the fraction above.

First let's start from the numerator, $HHI(c) * mean\_pop(c)$, named henceforth as `multiplied_hhi`.

[41]:
```
race_capital_df["multiplied_hhi"] = race_capital_df.hhi * race_capital_df.
 ↪mean_students_per_cohort
race_capital_df
```

[41]:
```
     college                                college_name       hhi  \
1    1218200           Chattahoochee Valley Community College  0.585278
2     101500              Enterprise State Community College  0.515541
3     101700                  Gadsden State Community College  0.493447
4     787100  George C Wallace State Community College-Hance...  0.230032
6     102200                Jefferson State Community College  0.475184
..       ...                                             ...       ...
950   728900                           Central Wyoming College  0.461718
951   392900                           Eastern Wyoming College  0.236304
952   925900                  Laramie County Community College  0.345499
953   393100                                 Northwest College  0.295868
954   393300                 Western Wyoming Community College  0.300614

     bias_parent_ses_college  mean_students_per_cohort percentile  \
1                      5.260                263.000000         60
2                     -3.047                289.000000         55
3                     -0.233                819.333333         50
4                      0.578                923.000000         15
6                      3.767               1268.666667         50
```

```
..                    ...                    ...        ...
950                -1.448             238.000000         45
951                -5.346             205.000000         20
952                 0.592             489.666667         30
953                 1.465             337.000000         25
954                -1.671             430.000000         25

     multiplied_hhi
1        153.928002
2        148.991251
3        404.297394
4        212.319450
6        602.849841
..              ...
950      109.888977
951       48.442399
952      169.179301
953       99.707351
954      129.263853

[781 rows x 7 columns]
```

And the normalization term in the denominator $\sum_{other\_c \in perc(c)} mean\_pop(other\_c)$.

For this term we need to create a temporary grouped dataset to hold the percentiles' sums and merge it back to our original one.

This way, we will have the columns for the numerator and the denominator for each college and computing the weighted HHI will become trivial.

[42]:
```python
category_mean_df = race_capital_df.groupby("percentile").sum(numeric_only=True)\
                        .loc[:, ["mean_students_per_cohort"]].reset_index()
category_mean_df
```

[42]:

	percentile	mean_students_per_cohort
0	0	3022.166667
1	5	39334.833333
2	10	5684.000000
3	15	37676.000000
4	20	70857.666667
5	25	102501.000000
6	30	195517.000000
7	35	101086.333333
8	40	67978.333333
9	45	70503.500000
10	50	84063.833333
11	55	88848.166667
12	60	211268.666667

```
13         65            229312.666667
14         70            147703.333333
15         75            148439.833333
16         80            219314.333333
17         85             94230.666667
18         90              2610.166667
19         95              1769.666667
```

[43]:
```
race_capital_df = race_capital_df.merge(category_mean_df,
                                        how="inner",
                                        on="percentile")
race_capital_df = race_capital_df.loc[:, ["percentile",␣
 ↪"bias_parent_ses_college", "multiplied_hhi"]]
race_capital_df
```

[43]:
```
    percentile  bias_parent_ses_college  multiplied_hhi
0           60                    5.260      153.928002
1           60                    7.844      676.846685
2           60                    1.897      267.004819
3           60                    2.437      280.394461
4           60                    1.480     7983.889198
..         ...                      ...             ...
776          0                   -4.250       29.638785
777          0                   -8.808       16.959042
778          0                   -5.774       33.726974
779          0                   -0.705       41.014739
780          0                   -0.782       37.240564

[781 rows x 3 columns]
```

[44]:
```
race_capital_df = race_capital_df.groupby("percentile")\
                                 .agg({"multiplied_hhi": "sum",␣
 ↪"bias_parent_ses_college": "mean"})\
                                 .reset_index()
race_capital_df
```

[44]:
```
   percentile  multiplied_hhi  bias_parent_ses_college
0           0      263.856309                -3.356875
1           5     5411.406282                 0.318786
2          10     1015.317625                 1.275111
3          15     7652.497475                 0.384381
4          20    18533.193110                 0.615600
5          25    30088.520109                 0.685030
6          30    65915.314963                 1.575897
7          35    38087.373506                 2.048433
8          40    28470.520858                 0.818596
9          45    31992.378963                 2.931465
```

```
10          50   41705.437630                 2.351824
11          55   47276.884667                 3.398089
12          60  120630.411517                 3.968127
13          65  139521.016295                 3.631423
14          70   96651.668230                 5.191339
15          75  102064.403152                 5.745143
16          80  160285.045191                 5.193390
17          85   72783.564511                 3.953400
18          90    2115.909185                 4.913667
19          95    1494.164685                 3.187000
```

Above we see a dataframe holding the sum of the multiplied HHI, and the mean bias for each percentile rank.

Let's finish the computation by dividing with our denominator. We are merging with the temporary dataframe `category_mean_df` which holds the sum of the mean_students_per_cohort *for each* percentile.

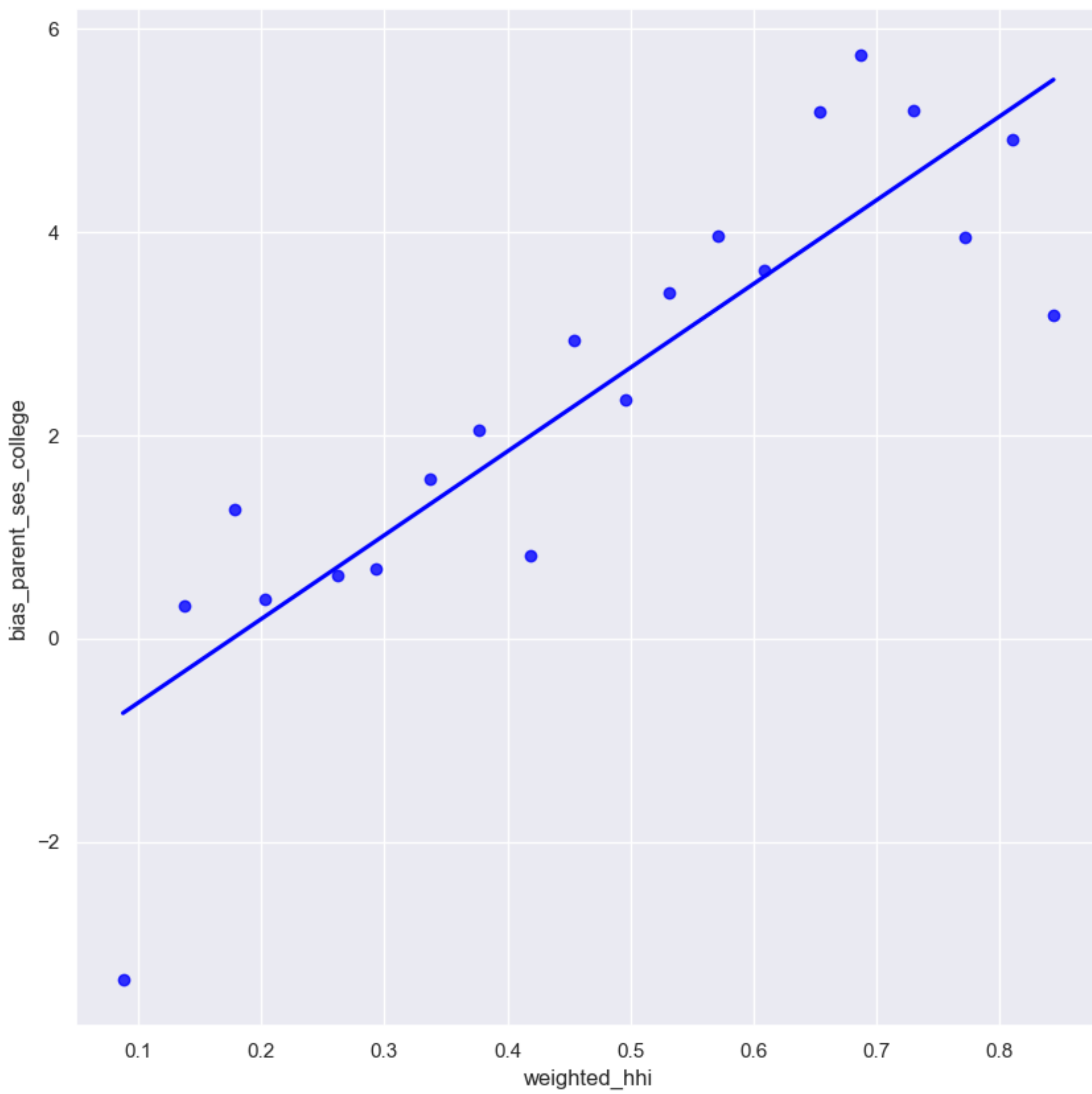
[45]:
```
race_capital_df = race_capital_df.merge(category_mean_df,
                                        how="inner",
                                        on="percentile")
race_capital_df["weighted_hhi"] = race_capital_df.multiplied_hhi /␣
 ↪race_capital_df.mean_students_per_cohort
race_capital_df
```

[45]:
```
    percentile  multiplied_hhi  bias_parent_ses_college  \
0            0      263.856309                -3.356875
1            5     5411.406282                 0.318786
2           10     1015.317625                 1.275111
3           15     7652.497475                 0.384381
4           20    18533.193110                 0.615600
5           25    30088.520109                 0.685030
6           30    65915.314963                 1.575897
7           35    38087.373506                 2.048433
8           40    28470.520858                 0.818596
9           45    31992.378963                 2.931465
10          50    41705.437630                 2.351824
11          55    47276.884667                 3.398089
12          60   120630.411517                 3.968127
13          65   139521.016295                 3.631423
14          70    96651.668230                 5.191339
15          75   102064.403152                 5.745143
16          80   160285.045191                 5.193390
17          85    72783.564511                 3.953400
18          90     2115.909185                 4.913667
19          95     1494.164685                 3.187000

    mean_students_per_cohort  weighted_hhi
```

0	3022.166667	0.087307
1	39334.833333	0.137573
2	5684.000000	0.178627
3	37676.000000	0.203113
4	70857.666667	0.261555
5	102501.000000	0.293544
6	195517.000000	0.337133
7	101086.333333	0.376781
8	67978.333333	0.418818
9	70503.500000	0.453770
10	84063.833333	0.496116
11	88848.166667	0.532109
12	211268.666667	0.570981
13	229312.666667	0.608431
14	147703.333333	0.654363
15	148439.833333	0.687581
16	219314.333333	0.730846
17	94230.666667	0.772398
18	2610.166667	0.810641
19	1769.666667	0.844320

And with that, our weighted HHI computation is complete. We build a plot with the x axis as the weighted HHI and the y axis as the mean of the friending bias.

[46]:
```
sns.regplot(
    x="weighted_hhi",
    y="bias_parent_ses_college",
    data=race_capital_df,
    ci=None,
    color="blue")

sns.set(rc={"figure.figsize":(8,10)})
```



We will improve on this graph later. For now it's enough to know that our numbers seem roughly correct. Of course we aren't expecting exact matches, since the data used by the Social Capital survey are published with noise for privacy reasons.

1.5.2 Neighborhood Racial Diversity

The original study sourced the data for this segment from the 2018 American Community Survey (ACS). Looking up the US central census database we find this dataset which contains data about the racial composition of every US ZIP Code.

[47]:
```
# We will ignore the warning for the same reasons as before, keeping in mind the␣
 ↪possibility
# of malformed data or columns as we proceed.
neighborhood_df = pd.read_csv("data/neighborhood_data.csv", header=1)
```

```
neighborhood_df
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_7804\1374905736.py:3: DtypeWarning:

Columns (4,5,8,9,12,13,19,20,31,32,35,36,39,40,43,44,47,48,60,61,75,76,83,84,95,
96,100,101,103,104,107,108,115,116,119,120,123,124,131,132,251,252,279,280,284,2
85,304,305,362,363,364,365,366,367,368,369,374,375,376,377,378,379,380,381,382,3
83,384,385,386,387,388,389,390,391,392,393,394,395,396,397,398,399,400,401,402,4
03,404,405,406,407,408,409,410,411,412,413,414,415,416,417,418,419,420,421,422,4
23,424,425,430,431,432,433,434,435,436,437,438,439,440,441,442,443,444,445,446,4
47,448,449,450,451,452,453,458,459,460,461,462,463,464,465,490,491,492,493,494,4
95,496,497,498,499,500,501,502,503,504,505,506,507,508,509,510,511,512,513,514,5
15,516,517,518,519,520,521,522,523,524,525,526,527,528,529,530,531,532,533,534,5
35,536,537,538,539,540,541,542,543,544,545,546,547,548,549,550,551,552,553,554,5
55,556,557,558,559,560,561,562,563,564,565,566,567,568,569,570,571,572,573,574,5
75,576,577,578,579,580,581,582,583,584,585,586,587,588,589,590,591,592,593,594,5
95,596,597,598,599,600,601,602,603,604,605,610,611,612,613,614,615,616,617,618,6
19,620,621,622,623,624,625,626,627,628,629,630,631,632,633,638,639,640,641,642,6
43,644,645,646,647,648,649,650,651,652,653,654,655,656,657,658,659,660,661,662,6
63,664,665,666,667,668,669,670,671,672,673,674,675,676,677,678,679,680,681,682,6
83,684,685,686,687,688,689,690,691,692,693,694,695,696,697,706,707,708,709,710,7
11,712,713) have mixed types. Specify dtype option on import or set
low_memory=False.
```

[47]:
```
             Geography Geographic Area Name  \
0       8600000US00601          ZCTA5 00601
1       8600000US00602          ZCTA5 00602
2       8600000US00603          ZCTA5 00603
3       8600000US00606          ZCTA5 00606
4       8600000US00610          ZCTA5 00610
...                ...                  ...
33115   8600000US99923          ZCTA5 99923
33116   8600000US99925          ZCTA5 99925
33117   8600000US99926          ZCTA5 99926
33118   8600000US99927          ZCTA5 99927
33119   8600000US99929          ZCTA5 99929

       Estimate!!SEX AND AGE!!Total population  \
0                                         17242
1                                         38442
2                                         48814
3                                          6437
4                                         27073
...                                         ...
33115                                        15
33116                                       927
```

```
33117                                          1635
33118                                            38
33119                                          2484

       Annotation of Estimate!!SEX AND AGE!!Total population  \
0                                                     NaN
1                                                     NaN
2                                                     NaN
3                                                     NaN
4                                                     NaN
...                                                   ...
33115                                                 NaN
33116                                                 NaN
33117                                                 NaN
33118                                                 NaN
33119                                                 NaN

      Margin of Error!!SEX AND AGE!!Total population  \
0                                                314
1                                                150
2                                                749
3                                                304
4                                                205
...                                              ...
33115                                             22
33116                                             98
33117                                            122
33118                                             30
33119                                          *****

      Annotation of Margin of Error!!SEX AND AGE!!Total population  \
0                                                     NaN
1                                                     NaN
2                                                     NaN
3                                                     NaN
4                                                     NaN
...                                                   ...
33115                                                 NaN
33116                                                 NaN
33117                                                 NaN
33118                                                 NaN
33119                                               *****

       Estimate!!SEX AND AGE!!Total population!!Male  \
0                                               8426
1                                              18842
2                                              23939
```

```
3                                                 3212
4                                                13112
...                                                ...
33115                                                0
33116                                              526
33117                                              882
33118                                               20
33119                                             1302

       Annotation of Estimate!!SEX AND AGE!!Total population!!Male  \
0                                                    NaN
1                                                    NaN
2                                                    NaN
3                                                    NaN
4                                                    NaN
...                                                  ...
33115                                                NaN
33116                                                NaN
33117                                                NaN
33118                                                NaN
33119                                                NaN

      Margin of Error!!SEX AND AGE!!Total population!!Male  \
0                                                    159
1                                                     60
2                                                    366
3                                                    187
4                                                     82
...                                                  ...
33115                                                  9
33116                                                 64
33117                                                 68
33118                                                 18
33119                                                 76

      Annotation of Margin of Error!!SEX AND AGE!!Total population!!Male  ...  \
0                                                    NaN                  ...
1                                                    NaN                  ...
2                                                    NaN                  ...
3                                                    NaN                  ...
4                                                    NaN                  ...
...                                                  ...                  ...
33115                                                NaN                  ...
33116                                                NaN                  ...
33117                                                NaN                  ...
33118                                                NaN                  ...
33119                                                NaN                  ...
```

```
       Percent Annotation of Estimate!!CITIZEN, VOTING AGE POPULATION!!Citizen,
18 and over population  \
0                                                    NaN
1                                                    NaN
2                                                    NaN
3                                                    NaN
4                                                    NaN
...                                                  ...
33115                                                NaN
33116                                                NaN
33117                                                NaN
33118                                                NaN
33119                                                NaN

       Percent Estimate!!CITIZEN, VOTING AGE POPULATION!!Citizen, 18 and over
population!!Male  \
0                                                   48.1
1                                                   48.7
2                                                   48.1
3                                                   49.3
4                                                   47.6
...                                                  ...
33115                                                0.0
33116                                               55.5
33117                                               53.6
33118                                               52.6
33119                                               51.3

      Percent Margin of Error!!CITIZEN, VOTING AGE POPULATION!!Citizen, 18 and
over population!!Male  \
0                                                    0.6
1                                                    0.2
2                                                    0.6
3                                                    1.5
4                                                    0.2
...                                                  ...
33115                                               60.1
33116                                                3.7
33117                                                2.9
33118                                               26.3
33119                                                2.9

      Percent Annotation of Margin of Error!!CITIZEN, VOTING AGE
POPULATION!!Citizen, 18 and over population!!Male  \
0                                                    NaN
1                                                    NaN
```

```
2                                                         NaN
3                                                         NaN
4                                                         NaN
...                                                       ...
33115                                                     NaN
33116                                                     NaN
33117                                                     NaN
33118                                                     NaN
33119                                                     NaN

      Percent Annotation of Estimate!!CITIZEN, VOTING AGE POPULATION!!Citizen,
18 and over population!!Male  \
0                                                         NaN
1                                                         NaN
2                                                         NaN
3                                                         NaN
4                                                         NaN
...                                                       ...
33115                                                     NaN
33116                                                     NaN
33117                                                     NaN
33118                                                     NaN
33119                                                     NaN

      Percent Estimate!!CITIZEN, VOTING AGE POPULATION!!Citizen, 18 and over
population!!Female  \
0                                                        51.9
1                                                        51.3
2                                                        51.9
3                                                        50.7
4                                                        52.4
...                                                       ...
33115                                                   100.0
33116                                                    44.5
33117                                                    46.4
33118                                                    47.4
33119                                                    48.7

      Percent Annotation of Estimate!!CITIZEN, VOTING AGE POPULATION!!Citizen,
18 and over population!!Female  \
0                                                         NaN
1                                                         NaN
2                                                         NaN
3                                                         NaN
4                                                         NaN
...                                                       ...
33115                                                     NaN
```

```
33116                                                  NaN
33117                                                  NaN
33118                                                  NaN
33119                                                  NaN

      Percent Margin of Error!!CITIZEN, VOTING AGE POPULATION!!Citizen, 18 and
over population!!Female  \
0                                                      0.6
1                                                      0.2
2                                                      0.6
3                                                      1.5
4                                                      0.2
...                                                    ...
33115                                                 60.1
33116                                                  3.7
33117                                                  2.9
33118                                                 26.3
33119                                                  2.9

       Percent Annotation of Margin of Error!!CITIZEN, VOTING AGE
POPULATION!!Citizen, 18 and over population!!Female  \
0                                                      NaN
1                                                      NaN
2                                                      NaN
3                                                      NaN
4                                                      NaN
...                                                    ...
33115                                                  NaN
33116                                                  NaN
33117                                                  NaN
33118                                                  NaN
33119                                                  NaN

      Unnamed: 714
0              NaN
1              NaN
2              NaN
3              NaN
4              NaN
...            ...
33115          NaN
33116          NaN
33117          NaN
33118          NaN
33119          NaN

[33120 rows x 715 columns]
```

After some digging through the database's columns we find the columns containing the populations for each race for each ZIP code.

For the sake of our sanity we will only compute the statistics for people identifying as one race. A cursory look through our dataset suggests bi-racial people are rare enough that they *should not* have any significant impact on our results.

[48]:
```
name_map = {
    "Estimate!!SEX AND AGE!!Total population": "Population",
    "Estimate!!RACE!!Total population!!One race!!White": "White",
    "Estimate!!RACE!!Total population!!One race!!Black or African American":␣
 ↪"Black",
    "Estimate!!RACE!!Total population!!One race!!Asian": "Asian",
    "Estimate!!RACE!!Total population!!One race!!Some other race": "Other",
    "Estimate!!RACE!!Total population!!One race!!Native Hawaiian and Other␣
 ↪Pacific Islander": "Pacific",
    "Estimate!!RACE!!Total population!!One race!!American Indian and Alaska␣
 ↪Native": "Alaskan"
}

columns = ["Geographic Area Name"]
columns.extend(name_map.keys())

neighborhood_df = neighborhood_df.loc[:, columns]
neighborhood_df.rename(columns=name_map, inplace=True)
neighborhood_df
```

[48]:
```
      Geographic Area Name  Population  White  Black  Asian  Other  Pacific  \
0              ZCTA5 00601       17242  13026    145      3   3929        0
1              ZCTA5 00602       38442  30529   1070      0   1540        0
2              ZCTA5 00603       48814  37330   1930    364   8007       10
3              ZCTA5 00606        6437   2627    149      0   3518        0
4              ZCTA5 00610       27073  20451    696      0   2871        1
...                    ...         ...    ...    ...    ...    ...      ...
33115          ZCTA5 99923          15     15      0      0      0        0
33116          ZCTA5 99925         927    446      3      0      0        7
33117          ZCTA5 99926        1635    239      0     22     17        0
33118          ZCTA5 99927          38     38      0      0      0        0
33119          ZCTA5 99929        2484   1670      2     69     30        9

       Alaskan
0           25
1            0
2          115
3           18
4            0
...        ...
33115        0
```

```
33116      366
33117     1210
33118        0
33119      496

[33120 rows x 8 columns]
```

[49]:
```python
neighborhood_df.rename(columns={"Geographic Area Name": "zip"}, inplace=True)

get_zip = lambda x: x[6:]
neighborhood_df.zip = neighborhood_df.zip.apply(get_zip).apply(int)

neighborhood_df.zip
```

[49]:
```
0          601
1          602
2          603
3          606
4          610
         ...
33115    99923
33116    99925
33117    99926
33118    99927
33119    99929
Name: zip, Length: 33120, dtype: int64
```

We follow the exact same procedure as before to compute each neighborhood's HHI. We will then merge it with our previously loaded "economic connectedness by zip dataset" used in Q3.

[50]:
```python
p_2 = lambda x: np.power(x / neighborhood_df.Population, 2)
neighborhood_df["hhi"] = 1 - (p_2(neighborhood_df.White) + p_2(neighborhood_df.
 ↪Black) +\
                              p_2(neighborhood_df.Asian) + p_2(neighborhood_df.
 ↪Other) +\
                              p_2(neighborhood_df.Pacific) + p_2(neighborhood_df.
 ↪Alaskan))

neighborhood_df = neighborhood_df.loc[:, ["zip", "hhi"]]
neighborhood_df
```

[50]:
```
         zip       hhi
0        601  0.377249
1        602  0.366934
2        603  0.386643
3        606  0.534210
4        610  0.417461
...      ...       ...
```

```
33115  99923  0.000000
33116  99925  0.612570
33117  99926  0.430652
33118  99927  0.000000
33119  99929  0.507207

[33120 rows x 2 columns]
```

[51]:
```
neighborhood_df = neighborhood_df.merge(ec_df,
                       how="inner",
                       on="zip")

neighborhood_df.dropna(inplace=True)
neighborhood_df.bias_grp_mem_zip = neighborhood_df.bias_grp_mem_zip * 100 #make␣
 ↪it a % column

neighborhood_df = neighborhood_df.loc[:, ["zip", "hhi", "num_below_p50",␣
 ↪"bias_grp_mem_zip"]]
neighborhood_df
```

[51]:
```
         zip       hhi  num_below_p50  bias_grp_mem_zip
0       1001  0.143371     995.787468          2.434000
1       1002  0.409802    1312.117077          9.856000
3       1005  0.094008     381.519745          0.850000
4       1007  0.101516     915.396667         -1.188000
8       1013  0.295885    2616.550354         13.699999
...      ...       ...            ...               ...
23021  99835  0.559399     790.157898          0.953000
23023  99901  0.526235    1192.299809          5.710000
23024  99921  0.512764     365.768661          6.010000
23026  99926  0.430652     311.014252          0.877000
23027  99929  0.507207     313.282990          1.350000

[14269 rows x 4 columns]
```

And the weighted HHI exactly as demonstrated above:

[52]:
```
bins = pd.cut(neighborhood_df.hhi, bins=20, labels=np.arange(0, 100, 5)) #␣
 ↪divide into 20 percentile bins
neighborhood_df["percentile"] = bins
neighborhood_df
```

[52]:
```
         zip       hhi  num_below_p50  bias_grp_mem_zip percentile
0       1001  0.143371     995.787468          2.434000         15
1       1002  0.409802    1312.117077          9.856000         45
3       1005  0.094008     381.519745          0.850000         10
4       1007  0.101516     915.396667         -1.188000         10
8       1013  0.295885    2616.550354         13.699999         30
```

```
...      ...       ...          ...              ...         ...
23021   99835   0.559399     790.157898         0.953000      60
23023   99901   0.526235    1192.299809         5.710000      55
23024   99921   0.512764     365.768661         6.010000      55
23026   99926   0.430652     311.014252         0.877000      45
23027   99929   0.507207     313.282990         1.350000      55

[14269 rows x 5 columns]
```

Calculate the numerator $HHI(c) * mean\_pop(c)$:

[53]:
```
neighborhood_df["multiplied_hhi"] = neighborhood_df.hhi * neighborhood_df.
 ↪num_below_p50
neighborhood_df = neighborhood_df.loc[:, ["percentile", "multiplied_hhi",␣
 ↪"bias_grp_mem_zip", "num_below_p50"]]

neighborhood_df
```

[53]:
```
       percentile  multiplied_hhi  bias_grp_mem_zip  num_below_p50
0              15      142.767122          2.434000     995.787468
1              45      537.708355          9.856000    1312.117077
3              10       35.866017          0.850000     381.519745
4              10       92.927450         -1.188000     915.396667
8              30      774.198610         13.699999    2616.550354
...           ...             ...               ...            ...
23021          60      442.013334          0.953000     790.157898
23023          55      627.429525          5.710000    1192.299809
23024          55      187.552843          6.010000     365.768661
23026          45      133.939062          0.877000     311.014252
23027          55      158.899171          1.350000     313.282990

[14269 rows x 4 columns]
```

Calculate the denominator $\sum_{other\_c \in perc(c)} mean\_pop(other\_c)$ (alongside the mean friending bias):

[54]:
```
category_mean_df = neighborhood_df.groupby("percentile")\
                                  .agg({"multiplied_hhi": "sum",
                                        "num_below_p50": "sum",
                                        "bias_grp_mem_zip": "mean"})\
                                  .loc[:, ["multiplied_hhi", "num_below_p50",␣
 ↪"bias_grp_mem_zip"]]\
                                  .reset_index()
category_mean_df
```

[54]:
```
   percentile  multiplied_hhi  num_below_p50  bias_grp_mem_zip
0           0    1.529810e+04   5.291259e+05          2.923441
1           5    1.137555e+05   1.652997e+06          3.901205
```

```
2           10    1.985498e+05   1.797302e+06          4.349621
3           15    2.727948e+05   1.761248e+06          5.039688
4           20    3.661530e+05   1.825688e+06          5.785545
5           25    4.225426e+05   1.728120e+06          5.715311
6           30    4.878999e+05   1.681081e+06          6.359059
7           35    6.176948e+05   1.844567e+06          7.449996
8           40    7.358131e+05   1.934090e+06          7.454627
9           45    8.977249e+05   2.110080e+06          8.324649
10          50    1.001965e+06   2.135085e+06          8.772321
11          55    1.246022e+06   2.423439e+06         10.316320
12          60    1.584614e+06   2.845912e+06         11.724285
13          65    1.352807e+06   2.244159e+06         10.799742
14          70    1.208695e+06   1.867079e+06         11.346929
15          75    9.889712e+05   1.432801e+06         11.317608
16          80    4.639941e+05   6.305277e+05          9.772231
17          85    1.238934e+05   1.605946e+05          7.133885
18          90    2.417612e+04   2.951411e+04          3.048667
19          95    8.674691e+03   1.010231e+04          0.003000
```

Calculate the final `weighted_hhi` values:

[55]:
```
category_mean_df["weighted_hhi"] = category_mean_df.multiplied_hhi /␣
 ↪category_mean_df.num_below_p50
category_mean_df.weighted_hhi
```

[55]:
```
0     0.028912
1     0.068818
2     0.110471
3     0.154887
4     0.200556
5     0.244510
6     0.290230
7     0.334872
8     0.380444
9     0.425446
10    0.469286
11    0.514154
12    0.556803
13    0.602812
14    0.647372
15    0.690236
16    0.735882
17    0.771467
18    0.819138
19    0.858684
Name: weighted_hhi, dtype: float64
```

And now *finally* we can plot our two graphs together:

[56]:
```
from matplotlib.lines import Line2D

ax = sns.regplot(
    x="weighted_hhi",
    y="bias_grp_mem_zip",
    data=category_mean_df,
    ci=None,
    color="orange")

sns.regplot(
    x="weighted_hhi",
    y="bias_parent_ses_college",
    data=race_capital_df,
    ci=None,
    color="blue",
    marker="D")

legend_elements = [Line2D([0], [0], color="blue", lw=4, label='College'),
Line2D([0], [0], color='orange', lw=4, label='Neighborhood')]

ax.set(xlabel="Racial Diversity (Herfindahl-Hirschman Index) in Group",
       ylabel="Friending Bias among Low-SES Individuals",
       aspect=0.03)

ax.legend(handles=legend_elements, loc="lower right")

sns.set(rc={"figure.figsize":(8,10)})
```