MSB Final Project: The Genetics of Music

Kabir Gupta

May 21, 2021

Introduction

One of Tinbergen's four causes of behavior is development, or ontogeny: the question of how a behavior came to an organism in the first place.[1] While this subject can delve deep into Mendellian inheritance and meiosis, or different forms of learning and different classifications of innate behaviors, the central and highest-level question asked during the study of development is that of nature versus nurture: is the trait genetically inherited or environmentally acquired? Of course, traits are often not one or the other but will rather fall on a spectrum between the two; the purpose of this project is to study one such trait in hopes of determining whether it is influenced more by genes or environment. This trait is musical taste: what kinds of music do people like to listen to?

It is surprisingly common, in the researcher's own (anecdotal) experience, to find two friends who have vastly different music tastes, and the same applies to family members. This leads into the question of who makes more of an impact on your music tastes — friends will often attest to have virtually identical music taste, but the same could apply to family members; it only seems to vary from person to person and family to family. By gathering a large sample of data, however, it could be determined whether there is actually some sort of trend, or association, between a person's music tastes and their friends'/family's.

It was expected that both family members and friends would have a significant impact on a person's music choices, although one might not necessarily outweigh the other: that is, there may not be a significant difference between the family's impact and the friends'. This is because people have very strong reasons to like similar music to both a family member and a friend, so that one relationship type should not be significantly closer than the other.

[1]Special thanks to Lincoln Auster and Thomas Morford for code review, bug fixes and formatting advice.

Materials and Methods

A Google Forms survey was used to gather data for this study, and a Python script was used to analyze it. Each respondent was asked to select the musical genres that they enjoy out of 6 options: pop, jazz/blues, rock, country/folk, rap/hip hop, and classical. The respondent was then asked to decide whether they like or dislike each of 12 musical clips, out of which each genre was represented by two clips: Uptown Funk and Dead Girl in the Pool (pop), What a Wonderful World and The Thrill is Gone (jazz and blues), Bohemian Rhapsody and Hurt (rock), Old Town Road and I Walk the Line (country and folk), Rap God and Gangsta's Paradise (rap and hip hop), and finally Eine Kleine Nachtmusik and Duel of the Fates (classical).

For the purposes of this study, it was useful to be able to identify whether there were some genres that the respondent did not necessarily dislike, even if it was not something they thought they actively listen to. So, while the first part of the survey asked for what they thought they like, the second part of the survey was in an attempt to see what they might actually like. However, it's possible that somebody who generally likes a certain genre did not like the specific two clips that they had been asked about from that genre. So, the question about categories needed to be weighted more heavily than the questions about specific songs. Further, an overall system was needed for condensing the 2-part data that would be gathered — genre and music preference, both on nominal scales — into one number that could be used in statistical testing. Hence, the following scoring system was created, as seen in Figure 1.

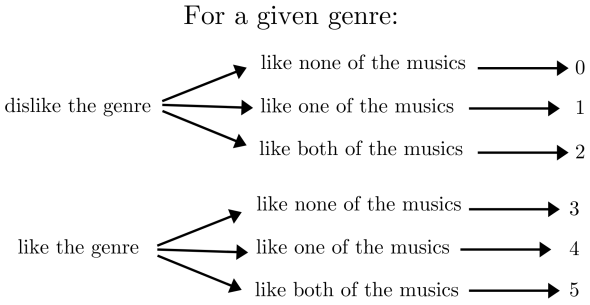

Figure 1: The procedure used to generate ordinal-scale scores for each respondent, for each genre.

This means that if a respondent says they enjoy a particular genre but doesn't like any of the musics, they can still walk away with a score of 3, whereas if they liked both musics but said they disliked the genre in general, they would only get a 2. These rankings are on

an ordinal scale, from 0 to 5 (where there are no defined intervals between ranks).

Each respondent was requested to provide the names of two friends and two family members. The family members were sent a slightly different survey, which was identical to the one for students except that it did not ask them for the names of their friends or family members. At the end of data collection, each respondent was matched up to their two family members and two friends to perform statistical analysis. If not all of the family members and friends listed had responded to the survey, only the ones whose responses were present were tested against the student.

Data

There were two main variables being compared in this study: relationship type (family member or friend), on a nominal (binary) scale; and genre rankings that demonstrate the degree to which someone enjoys a particular genre, on an ordinal scale.

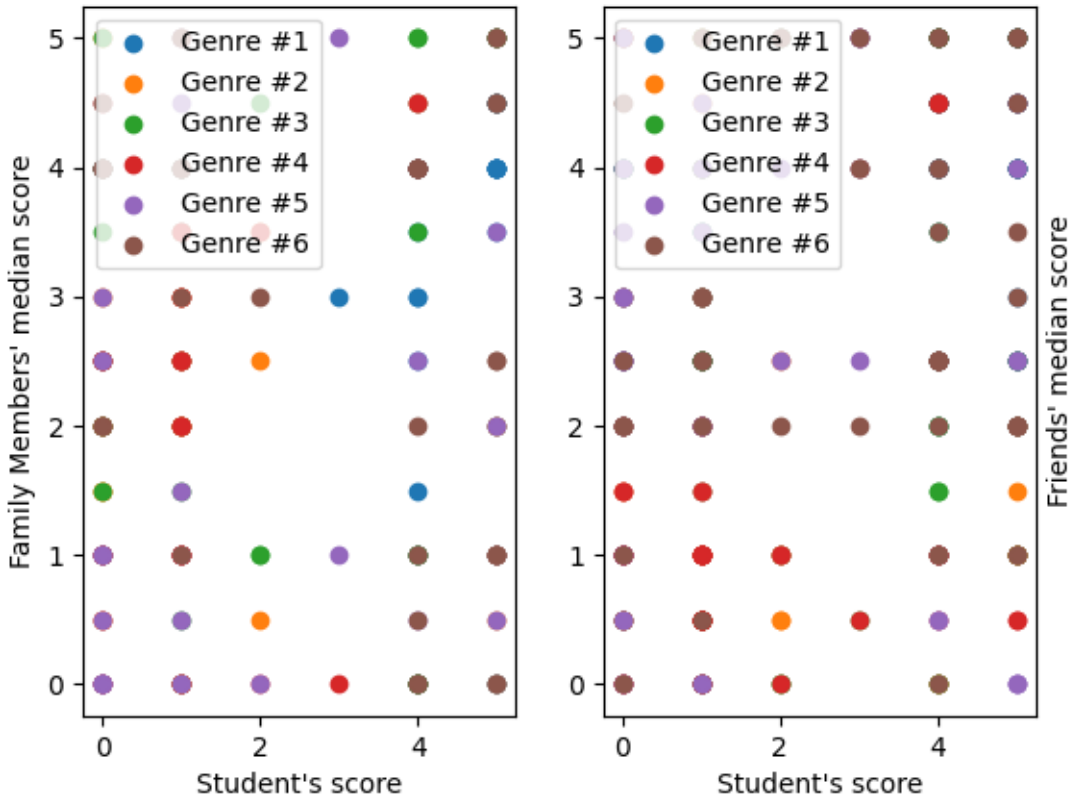

Figure 2: Scatter plots displaying the data gathered. At left, each student's score is plotted against the median of their family members' scores for each genre; at right, each student's score is plotted against the median of their friends' scores for each genre.

The scatter plots above display the results of data collection, spread across genres. The raw data gathered during this study is included in Appendix I.

A Wilcoxon test was used for the primary statistical analysis. Out of 49 tests conducted

for a difference between the students' rankings and their family members', 9 were statistically significant at a 90% confidence level (a=0.10), and 5 were significant at a 95% confidence level (a=0.05). For the friends, 14 tests came out significant at the 90% confidence level, and 7 at the 95% confidence level, out of a total of 79 tests. Overall, 18.4% of tests conducted for family members came back significant, compared to 17.7% of tests for friends.

The follow-up chi-square test for association had an insignificant result, meaning that the null hypothesis must be accepted: there is no association between relationship type and similarity of rankings. The test statistic χ^2 was calculated to be 3.389771795679717 with 9 degrees of freedom; p=0.9468207928306335.

An SRCC was also calculated for each relationship type for each genre. For family members, the correlation coefficients were r = 0.48, 0.05, -0.10, 0.13, 0.21, and -0.15 respectively. For friends, they were r = 0.21, 0.21, 0.07, 0.04, -0.03, and 0.06 respectively. Three of the coefficients overall (two on the family side and one on the friends side) were negative; all the rest were positive. Almost all of the correlation coefficients were quite weak (below 0.1); however, the correlation coefficient for Genre 1 (Pop) for family members could be classified as of moderate strength. Apart from the aforementioned coefficient, which had a p-value p=0.00679 (highly statistically significant), all the other coefficients were insignificant.

Finally, a t-test for two correlated samples was conducted between the family members' and friends' SRCCs to test for a significant difference in the correlation coefficients. The test statistic reported was t = 0.14. The critical value for the t-distribution at a 95% confidence level with 5 degrees of freedom is 2.57, and the test statistic found does not exceed this. The p-value was only p = 0.90, so the result was insignificant. Thus, we fail to reject the null hypothesis that family members and friends both influence people's musical choices to approximately the same extent.

Statistical Analysis

The most preliminary technique used in analyzing the collected data was assigning each individual an ordinal-scale ranking for each genre (in order to have two easily comparable populations). The specifications of how this was done have been provided in the Methods section. Once each student, family member and friend was assigned a set of 6 ranks (as there were 6 genres involved), a comparison of medians was conducted between the data sets. Because the data was on an ordinal scale, the statistical test had to be nonparametric. Further, the data was paired because each student was related to their corresponding family member or friend in some way. Consequently, a Wilcoxon test was used to test for a significant difference between each student's rankings and their family members', and between each student's rankings and their friends'. The results of these tests are summarized in the previous section.

The p-values generated by the Wilcoxon tests were sorted into an aggregated frequency table (class borders 0-0.1, 0.1-0.2, 0.2-0.3, and so on up to 1.0). This was done to each set of tests (family and friends), allowing for the construction of the contingency tables below (Tables 1 and 2). A chi-square test for association was conducted to find if there was an association between to what extent two people differ in music choice, and how the two people are related (family or friend).

	0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
Family	9	11	3	5	6	1	3	6	2	3
Friend	14	16	8	4	10	5	4	9	5	4

Table 1: Contingency table of observed values for χ^2

	0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
Family	8.80	10.34	4.21	3.45	6.13	2.30	2.68	5.74	2.68	2.6
Friend	14.20	16.66	6.79	5.55	9.88	3.70	4.32	9.26	4.32	4.32

Table 2: Contingency table of expected values for χ^2

Although the chi-square test returned an insignificant result, it was still worth calcualting a Spearman Rank Correlation Coefficient to get a sense of how strong the correlation between each pair of two people was for each genre. This gave a set of twelve SRCCs, one for each genre and for each relationship type (friend/family). The SRCC comparison tables were of the format as in Table 3 (only a part of the first one is included here, as they were all

similarly structured.)

x: student's ranking	y: family member's ranking
0	4
1	0
1	0
...	...

Table 3: SRCC table of values for students vs family members for Genre 1, Pop (truncated after row 3).

The twelve correlation coefficients that were found (six for each relationship type) have been reported in the previous section. One last statistical test was conducted, the t-test for two correlated (paired) samples. A significant result here would have meant that one relationship type (be that family member or friend) has significantly higher correlations between two people than the other. The two samples consisted of the set of all SRCCs for family members, and the set of all SRCCs for friends (thus n=6). So, each sample was matched because every SRCC for the family members applied to the same genre as the corresponding SRCC for the friends. Further, a parametric test could be used because correlation coefficients lie on a ratio scale (with an absolute zero) and there was no reason to suspect major skew in the data. The results of the two-sample t-test were reported in the previous section as well.

Discussion

Conducting the Wilcoxon tests for each student revealed that there were some cases where there was a significant difference between family members, and likewise for friends. However, when all of those cases were tallied up in a chi-square test, there did not appear to be a significant association between relationship type and music choices. While calculating the SRCCs supported this conclusion for the most part, one of the coefficients did suggest that family members may exert a special influence on students' choices when it comes to pop music based on the correlation coefficient alone. However, when a t-test was run between the SRCCs to check for a significant difference, the result was clear: neither relationship has greater power over a person's musical choices than the other.

The data gathered thus supports the original hypothesis that an environmental influence would not outweigh a genetical influence on a person's music choices, and vice versa. However, there are two ways these results could have been made more reliable:

- A larger sample size always helps increase the power of statistical analysis, and can thus catch smaller differences in a population that a small sample size cannot. The sample size for this study was 76 students and 65 family members; while that's not a tiny sample size, this study would definitely have been more powerful if (a) more people participated, but even more so (b) for every student who participated, both family members and both friends were able to fill out the respective surveys.

- The twelve songs selected to "represent" each genre may not have been very representative of each genre. For instance, "Hurt" is a heavy metal song that even an avid rock listener might not find as appealing as a pure rock song. "Eine Kleine Nachtmusik," while popular, is overused to the point of many classical enthusiasts not enjoying it as much as some Brahms or Shostakovich; and "Duel of the Fates" is a vocalized theme from the *Star Wars* soundtrack, so that it can technically be categorized as classical but when compared to Bach or Beethoven seems vastly different. These oversights were due to the researcher's own relative lack of musical knowledge (which has slightly improved over the course of this study).

A redo of this study could have the potential to be more successful if more time was spent to take as large a sample size as possible — for instance, if everybody in one school filled it out, it might only provide 500 people, but at least each person's friend would have (hopefully) filled it out too until it cycled back so that every person can be tested against four other people — and to use music clips that were much more representative of the genre (and likely to be liked by somebody who listens to the genre often).

Another possible flaw in this research was that it assumes that family members are the "genetic" factor, and friends are the "environmental". While it's true that there's usually no genetic relationship between two friends, the problem is that one's family is not only who they share the greatest amount of their DNA with, but also who they have usually grown up around and been brought up with. This means that a family member might encapsulate both the genetic and the environmental factor. Thus, this study cannot really conclude whether music has a large genetic part, because it is quite possible that the overlaps found between students and their family members were due to a shared environment — in some cases, people may even share much more of their environment with family members than with friends. To rectify this, a completely different study might need to be carried out, maybe one connecting genetically related people who have not been in the same environment (cousins? siblings who were separated at birth? long-lost parents or children? clones?) might be more powerful in revealing whether there is more of a genetic and an environmental factor at play.

References

A Google Forms survey was used in this study, and the data was linked into a Google Sheet spreadsheet. The following twelve songs were used during the study; the relevant portions were clipped out (10 seconds) for survey-takers to listen to.

1. "Uptown Funk", Mark Ronson, 00:58 to 1:08. https://youtu.be/OPf0YbXqDm0
2. "Dead Girl in the Pool", girl in red, 00:34 to 00:44. https://youtu.be/Pzq4TEU-wHo
3. "What a Wonderful World", Louis Armstrong, 00:17 to 00:27. https://youtu.be/VqhCQZaH4Vs
4. "The Thrill is Gone", B.B. King, 00:34 to 00:44. https://youtu.be/oica5jG7FpU
5. "Bohemian Rhapsody", Queen, 00:02 to 00:12. https://youtu.be/fJ9rUzIMcZQ
6. "Hurt", Nine Inch Nails, 4:01 to 4:11. https://youtu.be/0voTktdpIiI
7. "Old Town Road", Lil Nas X, 00:39 to 00:49. https://youtu.be/r7qovpFAGrQ
8. "I Walk the Line", Johnny Cash, 00:25 to 00:35. https://youtu.be/J5126CibNsk
9. "Rap God", Eminem, 00:25 to 00:35. https://youtu.be/XbGs_qK2PQA
10. "Gangsta's Paradise", Coolio, 00:57 to 1:07. https://youtu.be/fPO76Jlnz6c
11. "Eine Kleine Nachtmusik", Wolfgang Amadeus Mozart, 00:05 to 00:15. https://youtu.be/oy2zDJPIgwc
12. "Duel of the Fates", John Williams, 1:03 to 11:13. https://youtu.be/C2XUJ5PWg-8

For the Python script, the SciPy Statistics library (`scipy==1.6.3`) was used to perform statistical analyses. The Pandas library (`pandas==1.2.4`) was used to import data from a CSV (exported from Google Sheets) into the Python script. Finally, the Matplotlib library (`matplotlib==3.4.2`) was used to draw scatter plots from the data collected.

Appendix I

Here is the raw data collected over the course of study (after being anonymized).

Timestamp	Check all of the following which you enjoy listening to:	Music #1 Opinio
3/30/2021 18:25:03	Pop, Rock, Classical	Like
3/30/2021 19:04:10	Pop, Rock, Classical	Dislike
3/30/2021 19:08:20	Pop, Jazz/Blues, Rap/Hip Hop, Classical	Like
3/30/2021 19:16:50	Pop, Rap/Hip Hop, Classical	Like
3/30/2021 19:38:17	Pop, Rap/Hip Hop, Classical	Like
3/30/2021 19:43:52	Pop, Classical	Dislike
3/30/2021 19:47:44	Pop, Country/Folk, Rap/Hip Hop, Classical	Like
3/30/2021 19:57:52	Pop, Rap/Hip Hop	Dislike
3/30/2021 20:39:10	Pop, Rock, Classical	Like
3/30/2021 20:49:59	Rock	Dislike
3/31/2021 1:00:09	Pop, Jazz/Blues, Rock, Country/Folk, Rap/Hip Hop, Classical	Like
3/31/2021 6:47:05	Rock, Classical	Like
3/31/2021 7:56:24	Pop, Jazz/Blues, Country/Folk, Classical	Like
3/31/2021 11:29:15	Pop, Rock, Rap/Hip Hop	Like
4/20/2021 19:44:09	Pop, Jazz/Blues, Classical	Like
3/31/2021 12:45:53	Pop	Like
3/31/2021 12:54:02	Jazz/Blues, Classical	Like
3/31/2021 14:19:59	Pop, Rock, Classical	Like
3/31/2021 15:45:07	Classical	Dislike
3/31/2021 17:19:58	Pop, Rock	Like
3/31/2021 17:34:50	Pop, Country/Folk, Rap/Hip Hop, Classical	Like
3/31/2021 23:14:02	Pop, Rap/Hip Hop	Like
4/1/2021 10:59:53	Pop, Rock, Classical	Like
4/3/2021 11:18:50	Pop, Jazz/Blues, Rock, Rap/Hip Hop	Like
4/3/2021 15:43:11	Pop	Like
4/5/2021 21:37:17	Pop, Rock, Country/Folk	Like
4/7/2021 14:44:31	Pop, Jazz/Blues, Rap/Hip Hop, Classical	Like
4/8/2021 21:51:49	Pop, Rock, Rap/Hip Hop	Like
4/9/2021 0:23:57	Pop, Rock	Like
4/9/2021 0:42:24	Jazz/Blues, Rock, Country/Folk, Classical	Like
4/9/2021 16:03:19	Pop, Jazz/Blues, Rock, Country/Folk, Rap/Hip Hop, Classical	Like
4/11/2021 18:49:19	Pop, Country/Folk, Rap/Hip Hop, Classical	Like
4/12/2021 16:24:40	Pop, Jazz/Blues, Country/Folk	Like
4/13/2021 20:33:15	Pop	Like
4/15/2021 16:35:21	Pop, Rap/Hip Hop, Classical	Like
4/17/2021 20:02:48	Pop	Like
4/18/2021 13:16:53	Pop, Classical	Like
4/18/2021 20:17:28	Pop, Country/Folk, Classical	Like
4/19/2021 11:25:16	Classical	Like
4/19/2021 15:54:41	Pop, Jazz/Blues, Rock, Country/Folk, Rap/Hip Hop, Classical	Like
4/20/2021 22:22:37	Pop, Rap/Hip Hop	Like
4/23/2021 11:48:10	Pop, Jazz/Blues, Rock	Like
4/25/2021 14:12:21	Pop, Jazz/Blues, Rock, Country/Folk, Rap/Hip Hop, Classical	Like
4/26/2021 18:07:40	Pop, Rock, Rap/Hip Hop	Dislike
4/27/2021 20:09:21	Pop, Jazz/Blues, Classical	Like
5/4/2021 14:03:26	Pop, Country/Folk	Like
5/4/2021 23:40:11	Pop, Jazz/Blues, Rock, Country/Folk, Rap/Hip Hop, Classical	Like
5/5/2021 15:05:22	Pop, Jazz/Blues, Rock, Classical	Like

Timestamp	Check all of the following which you enjoy listening to:	Music #1 Opinio
3/30/2021 19:09:14	Pop	Like
3/30/2021 19:11:45	Jazz/Blues, Rock, Country/Folk, Classical	Dislike
3/30/2021 19:15:42	Pop, Rock, Country/Folk	Like
3/30/2021 19:46:28	Pop, Jazz/Blues, Rock, Country/Folk, Rap/Hip Hop	Like
3/30/2021 19:48:57	Classical	Dislike
3/30/2021 20:01:45	Pop	Dislike
3/30/2021 20:44:13	Pop, Rock	Like
3/30/2021 20:49:18	Pop, Rap/Hip Hop, Classical	Like
3/30/2021 21:02:21	Pop, Rock, Country/Folk, Rap/Hip Hop	Dislike
3/30/2021 21:56:23	Rock, Classical	Like
3/31/2021 11:36:00	Pop, Rap/Hip Hop	Like
3/31/2021 13:01:20	Rock, Country/Folk, Classical	Like
3/31/2021 13:59:28	Pop, Country/Folk, Rap/Hip Hop	Like
3/31/2021 16:55:03	Rock, Rap/Hip Hop	Dislike
3/31/2021 17:37:39	Pop, Jazz/Blues, Rock, Rap/Hip Hop, Classical	Like
3/31/2021 17:45:21	Pop, Rock, Country/Folk, Rap/Hip Hop, Classical	Like
3/31/2021 20:34:23	Pop, Jazz/Blues, Rock	Like
3/31/2021 20:35:02	Pop, Rock, Country/Folk, Classical	Like
3/31/2021 22:32:00	Jazz/Blues, Rock, Classical	Like
4/1/2021 10:53:08	Rock, Classical	Dislike
4/4/2021 12:39:59	Country/Folk, Classical	Dislike
4/9/2021 0:55:09	Pop, Classical	Dislike
4/9/2021 10:10:21	Pop, Jazz/Blues, Rock, Classical	Like
4/9/2021 20:48:44	Pop, Rock, Country/Folk, Rap/Hip Hop	Dislike
4/12/2021 16:47:10	Pop, Jazz/Blues, Country/Folk, Classical	Like
4/18/2021 13:17:33	Pop	Like
4/18/2021 20:26:35	Country/Folk	Dislike
4/19/2021 14:29:22	Pop, Jazz/Blues, Rock, Country/Folk, Classical	Dislike
4/27/2021 21:17:28	Pop, Country/Folk, Classical	Like
4/28/2021 19:58:16	Jazz/Blues, Rock, Classical	Dislike
5/4/2021 10:37:49	Pop, Country/Folk, Classical	Like
5/5/2021 15:12:23	Rock, Country/Folk, Classical	Like
5/5/2021 19:27:38	Pop, Country/Folk	Like
5/5/2021 23:40:43	Pop, Jazz/Blues, Rock, Rap/Hip Hop, Classical	Like
5/6/2021 9:16:58	Pop, Classical	Like
5/9/2021 22:04:19	Pop, Country/Folk, Classical	Dislike
5/9/2021 22:05:54	Pop, Country/Folk	Like
5/9/2021 22:40:12	Classical	Dislike
5/9/2021 23:28:42	Pop, Rock	Like
5/10/2021 6:44:47	Rock, Classical	Like
5/10/2021 12:22:06	Rock, Country/Folk	Dislike
5/10/2021 12:22:06	Jazz/Blues, Rock, Classical	Dislike
5/10/2021 7:58:09	Pop, Rock, Rap/Hip Hop, Classical	Like
5/10/2021 8:32:01	Jazz/Blues	Like
5/10/2021 8:33:11	Pop, Rock, Rap/Hip Hop	Like
5/10/2021 16:28:12	Pop, Rock, Country/Folk, Classical	Like
5/10/2021 18:25:39	Pop, Jazz/Blues, Rock, Classical	Like
5/10/2021 18:38:21	Jazz/Blues, Rock, Country/Folk, Rap/Hip Hop, Classical	Like

Appendix II

The following scripts were used to analyze the data gathered.

Main program:

```
#!/usr/bin/env python3


def findmatches(student, studs, fams):
    # assign family members and friends for a given student
    fam0 = ""  # an empty string by default
    fam1 = ""
    friend0 = ""
    friend1 = ""
    for friend in studs:
        if student.friends[0] == friend.name:
            friend0 = friend
        if student.friends[1] == friend.name:
            friend1 = friend
    for family in fams:
        if student.fam[0] == family.name:
            fam0 = family
        if student.fam[1] == family.name:
            fam1 = family
    return (fam0, fam1, friend0, friend1)


def wc(studs, fams):
    # initialize arrays to hold the results of the WC tests
    fam_results = []
    friend_results = []

    for index in range(len(studs)):
        student = studs[index]
        fam0, fam1, friend0, friend1 = findmatches(student, studs, fams)

        if fam0 != "":  # fam0 exists
            # run tests between student and family member 0
            if student.ranks == fam0.ranks:
                print(
                    "{} {} and {} {} (family member) have the same rankings. p=0".format(
                        student.name[1], student.name[0], fam0.name[1], fam0.name[0]
                    )
```

```
                )
                p = 0
            else:
                w, p = wilcoxon(student.ranks, fam0.ranks)
                print(
                    "The Wilcoxon test statistic between {} {} and {} {} (family member) is W={} (p={}).".format(
                        student.name[1],
                        student.name[0],
                        fam0.name[1],
                        fam0.name[0],
                        w,
                        p,
                    )
                )
            fam_results.append(p)

        if fam1 != "":  # fam1 exists
            # run tests between student and family member 1
            if student.ranks == fam1.ranks:
                print(
                    "{} {} and {} {} (family member) have the same rankings. p=0".format(
                        student.name[1], student.name[0], fam1.name[1], fam1.name[0]
                    )
                )
                p = 0
            else:
                w, p = wilcoxon(student.ranks, fam1.ranks)
                print(
                    "The Wilcoxon test statistic between {} {} and {} {} (family member) is W={} (p={}).".format(
                        student.name[1],
                        student.name[0],
                        fam1.name[1],
                        fam1.name[0],
                        w,
                        p,
                    )
                )
            fam_results.append(p)

        if friend0 != "":  # friend0 exists
            # run tests between student and friend 0
            if student.ranks == friend0.ranks:
                print(
                    "{} {} and {} {} (friend) have the same rankings. p=0".format(
                        student.name[1],
                        student.name[0],
                        friend0.name[1],
                        friend0.name[0],
                    )
```

```
                )
                p = 0
            else:
                w, p = wilcoxon(student.ranks, friend0.ranks)
                print(
                    "The Wilcoxon test statistic between {} {} and {} {} (friend) is W={} (p={}).".format(
                        student.name[1],
                        student.name[0],
                        friend0.name[1],
                        friend0.name[0],
                        w,
                        p,
                    )
                )
            friend_results.append(p)

        if friend1 != "":  # friend1 exists
            # run tests between student and friend 1
            if student.ranks == friend1.ranks:
                print(
                    "{} {} and {} {} (friend) have the same rankings. p=0".format(
                        student.name[1],
                        student.name[0],
                        friend1.name[1],
                        friend1.name[0],
                    )
                )
                p = 0
            else:
                w, p = wilcoxon(student.ranks, friend1.ranks)
                print(
                    "The Wilcoxon test statistic between {} {} and {} {} (friend) is W={} (p={}).".format(
                        student.name[1],
                        student.name[0],
                        friend1.name[1],
                        friend1.name[0],
                        w,
                        p,
                    )
                )

            friend_results.append(p)

    return (fam_results, friend_results)


def chi2(group1, group2):
    # conduct chi-square test for association
    g1 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    g2 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    for x in range(10):
```

```
        for p in group1:
            if p >= 0.1 * x and p < 0.1 * (x + 1):
                g1[x] += 1
        for p in group2:
            if p > 0.1 * x and p < 0.1 * (x + 1):
                g2[x] += 1
    for p in group1:
        if p == 1:
            g1[9] += 1
    for p in group2:
        if p == 1:
            g2[9] += 1

    chi2observed = [g1, g2]
    print(
        "Here are the observed frequencies of each type of result for family members and friends, respectively:"
    )
    print(chi2observed)
    X2, p, df, ex = chi2_contingency(chi2observed)
    print(
        "Here are the expected frequencies of each type of result for family members and friends, respectively:"
    )
    print(ex)
    print(
        "The result of the Chi-Square Test for Association is test statistic X^2 = {} with {} degrees of freedom, which gives us a p-value of {}.".format(
            X2, df, p
        )
    )
    return p


def srcc(students, families):
    famccs = []
    friendccs = []
    fig, (famplot, friendplot) = plt.subplots(1, 2)
    for genre in range(6):  #
        studentranksa = []  # for fam
        studentranksb = []  # for friend
        famranks = []
        friendranks = []

        for student in students:
            fam0, fam1, friend0, friend1 = findmatches(student, students, families)

            studentranksa.append(student.ranks[genre])
            if fam0 != "" and fam1 != "":  # both of them were found
                famranks.append(
                    (fam0.ranks[genre] + fam1.ranks[genre]) / 2
                )  # median not mean
```

```
            elif fam0 != "":  # only fam0 was found
                famranks.append(fam0.ranks[genre])  # pick the only one we
 have
            elif fam1 != "":  # only fam1 was found
                famranks.append(fam1.ranks[genre])
            else:  # no family members were found
                studentranksa.remove(
                    student.ranks[genre]
                )  # we can't use this student's data

            studentranksb.append(student.ranks[genre])
            if friend0 != "" and friend1 != "":  # repeat same for the
friends
                friendranks.append(
                    (friend0.ranks[genre] + friend1.ranks[genre]) / 2
                )  # median again
            elif friend0 != "":
                friendranks.append(friend0.ranks[genre])
            elif friend1 != "":
                friendranks.append(friend1.ranks[genre])
            else:
                studentranksb.remove(student.ranks[genre])

        print("The student ranks (x1) for Genre #%i were: " % (genre + 1),
 end="")
        print(studentranksa)
        print("The family ranks (y1) for Genre #%i were: " % (genre + 1),
end="")
        print(famranks)
        print("The student ranks (x2) for Genre #%i were: " % (genre + 1),
 end="")
        print(studentranksb)
        print("The friend ranks (y2) for Genre #%i were: " % (genre + 1),
end="")
        print(friendranks)

        gen_name = "Genre #%i" % (genre + 1)

        famplot.scatter(
            studentranksa, famranks, label=gen_name
        )  # add stud and fam to scatter plot
        friendplot.scatter(
            studentranksb, friendranks, label=gen_name
        )  # add stud and friend to scatter plot

        r, p = spearmanr(studentranksa, famranks)
        print(
            "The correlation coefficient between students and family
members for Genre #%i was %2.5f (p=%2.5f)"
            % ((genre + 1), r, p)
        )
        famccs.append(r)
        r, p = spearmanr(studentranksb, friendranks)
        print(
```

```
            "The correlation coefficient between students and friends for Genre #%i was %2.5f (p=%2.5f)"
            % ((genre + 1), r, p)
        )
        friendccs.append(r)

    famplot.legend(loc="upper left")  # create legends
    famplot.set_xlabel("Student's score")
    famplot.set_ylabel("Family Members' median score")
    friendplot.legend(loc="upper left")
    friendplot.set_xlabel("Student's score")
    friendplot.set_ylabel("Friends' median score")
    friendplot.yaxis.set_label_position("right")  # so that it has room to be seen
    plt.savefig("scatter_plots.png")  # export to a png

    return famccs, friendccs


def main():
    # create lists to hold student and family objects:
    studs = []
    fams = []
    print(students)
    print(families)
    print(
        "CSVs were successfully imported into the pandas dataframe; proceeding to creating objects for each individual."
    )

    # add a student/family object for each row in the dataframe
    for row in range(students.shape[0]):
        studs.append(Student.at_row(row))  # create students
    for row in range(families.shape[0]):
        fams.append(Family.at_row(row))  # create family

    print(
        "Student/Family objects were successfully created; running Wilcoxon tests between each individual and their friends/family members."
    )

    # run wilcoxon tests
    genetic, environmental = wc(studs, fams)
    print()
    print("Here are the results of the Wilcoxon Tests for the family members:")
    print(genetic)
    print("Here are the results of the Wilcoxon Tests for the friends:")
    print(environmental)
    print("Here is the most common result of the tests for the family members:")
    print(multimode(genetic))
    print("Here is the most common result of the tests for the friends:")
```

```
    print(multimode(environmental))

    p = chi2(genetic, environmental)
    if p > 0.5:
        print(
            "WARNING: There is not a statistically significant association between relationship type and similarity of music choice. Following results may be very unreliable."
        )
    print()
    genecc, envcc = srcc(studs, fams)

    print("Genetic correlation coefficients, as a list: " + str(genecc))
    print("Environmental correlation coefficients, as a list: " + str(envcc))

    print(
        "The average correlation coefficient for a genetic relationship is "
        + str(mean(genecc))
    )
    print(
        "The average correlation coefficient for an environmental relationship is "
        + str(mean(envcc))
    )

    print()
    t, p = ttest_rel(genecc, envcc)  # paired t-test for two samples
    print(
        "The paired t-test for a difference between genetic and environmental correlation coefficients gave the test statistic t="
        + str(t)
        + "."
    )
    print("The p-value was " + str(p) + ".")


if __name__ == "__main__":
    from scipy.stats import (
        wilcoxon,
        chi2_contingency,
        spearmanr,
        ttest_rel,
    )  # needed for statistical testing
    from statistics import mean, multimode  # needed for descriptive statistics
    import matplotlib.pyplot as plt  # needed for graphing SRCC data
    from people import *  # needed dataframes & classes from ./people.py

    main()
```

Wrapper script (using POSIX sh):

```
#!/usr/bin/env sh
```

```
for csv in private/*.csv; do
        # delete timestamp column if present
        if [ "$(head -n1 < "$csv" | cut -d ',' -f 1)" = "Timestamp" ];
    then
                cp "$csv" "${csv}.old" #make backup
                cut -d ',' -f 2- <"${csv}.old" >"$csv"
                rm "${csv}.old" #rm backup
                printf 'Removed timestamps from %s\n' "$csv"
        fi

        # serialize names of genres to integer values
        sed -i '' 's/Pop/1/g' "$csv" && printf "replaced 'pop' with '1' in
     %s\n" "$csv"
        sed -i '' 's/Jazz\/Blues/2/g' "$csv" && printf "replaced 'jazz/
    blues' with '2' in %s\n" "$csv"
        sed -i '' 's/Rock/3/g' "$csv" && printf "replaced 'rock' with '3'
    in %s\n" "$csv"
        sed -i '' 's/Country\/Folk/4/g' "$csv" && printf "replaced '
    country/folk' with '4' in %s\n" "$csv"
        sed -i '' 's/Rap\/Hip Hop/5/g' "$csv" && printf "replaced 'rap/hip
     hop' with '5' in %s\n" "$csv"
        sed -i '' 's/Classical/6/g' "$csv" && printf "replaced 'classical'
     with '6' in %s\n" "$csv"
        sed -i '' 's/Dislike/0/g' "$csv" && printf "replaced 'dislike'
    with '0' in %s\n" "$csv"
        sed -i '' 's/Like/1/g' "$csv" && printf "replaced 'like' with '1'
    in %s\n" "$csv"
        printf 'finished working on %s\n\n' "$csv"
done
python3 main.py

printf '\nfinished running script, exiting now\n'
```

Class definitions for students and families:

```
#!/usr/bin/env python3

from pandas import read_csv  # necessary for importing csv data as a
    pandas dataframe

students = read_csv("private/Students.csv")  # initialize dataframe for
    students
families = read_csv("private/Family.csv")  # and one for families


class Student:
    def __str__(self):
        return (
            str(self.name)
            + "\n"
            + str(self.ranks)
            + "\n"
            + str(self.fam)
            + "\n"
```

```
            + str(self.friends)
        )

    @classmethod
    def at_row(cls, n: int):
        row = [student for student in students.iloc[n]]
        return cls(row)

    # Construct a Student from a row of data. This row is expected to be laid
    # out in (first name, last name, ...  musics ..., family member a first name,
    # family member a last name, family number b first name, family number b
    # last name, friend a first name, friend a last name, friend b first name,
    # friend b last name).
    def __init__(self, row):
        self.name = (
            row[0],
            row[1],
        )

        self.fam = [
            (row[15], row[16]),
            (row[17], row[18]),
        ]

        self.friends = [
            (row[19], row[20]),
            (row[21], row[22]),
        ]

        self.ranks = [-1 for n in range(1, 7)]

        # generate & store rankings
        genres = row[2]  # temporarily store genre selections
        musics = [row[n] for n in range(3, 15)]

        # Each genre is represented by a number (1=pop, 2=jazz & blues, etc).
        for genre in range(1, 7):
            if str(genre) in genres:  # liked the genre
                self.ranks[genre - 1] = (
                    3 + musics[genre * 2 - 2] + musics[genre * 2 - 1]
                )
            else:  # disliked the genre
                self.ranks[genre - 1] = musics[genre * 2 - 2] + musics[genre * 2 - 1]


class Family:  # I'll have one object of this type for each participating family member; identical to Student but lacks fam[] and friends[]
    def __str__(self):
```

```
65          return str(self.name) + "\n" + str(self.ranks)
66
67      @classmethod
68      def at_row(cls, n: int):
69          row = [family for family in families.iloc[n]]
70          return cls(row)
71
72      # Construct a Family from a row of data. This row is expected to be
    laid
73      # out in (first name, last name, genres,  musics ...).
74      def __init__(self, row):
75          self.name = (
76              row[0],
77              row[1],
78          )
79
80          self.ranks = [-1 for n in range(1, 7)]
81
82          # generate & store rankings
83          genres = row[2]  # temporarily store genre selections
84          musics = [row[n] for n in range(3, 15)]
85
86          # Each genre is represented by a number (1=pop, 2=jazz & blues,
    etc).
87          for genre in range(1, 7):
88              if str(genre) in genres:  # liked the genre
89                  self.ranks[genre - 1] = (
90                      3 + musics[genre * 2 - 2] + musics[genre * 2 - 1]
91                  )
92              else:  # disliked the genre
93                  self.ranks[genre - 1] = musics[genre * 2 - 2] + musics[
    genre * 2 - 1]
```