Code

R Notebook

Replicating my Code

To replicate this code, you will need to have my sqlite database in the same directory as this R notebook, which I can share if it is necessary. You will also need to install the R packages RSqlite and reticulate. Both can be installed with the R install.packages() command. The first can execute and fetch sql queries and data in r and the second can run python in R. That also means you need to have a valid python interpreter set up on your machine and you must set up reticulate with it. This tutorial helped me with that: https://docs.rstudio.com/tutorials/user/using-python-with-rstudio-and-reticulate/ (https://docs.rstudio.com/tutorials/user/using-python-with-rstudio-and-reticulate/). Also, some of my plots use ggplot, so if your environment did not come with ggplot2, install that. Some of these queries took several minutes to run on my laptop due to R trying to load tons of data from SQL, so be patient or use a system with a fast disk and CPU.

Hide

```
setwd("C:/Users/daniel/Documents/AnimeAnalysis")
```

Data Collection

(Skip this if you are only interested in the R data analysis) This project deals with public user profiles on myanimelist.com. These profiles contain a lot of data, but the only personal data I collect are their anime ratings on a scale of 1-10. Their usernames are not even stored in my SQL database, so I have no way of tracing the numbers back to their users. I used python HTTP requests to collect all the relevant data from myanimelist.com's free REST API and sqlite to store it. The general steps I took are: 1. Verify how many users exist on myanimelist.com and exploit their "Member Search" feature to get the usernames of all users who claim to be between 10 and 36 years old. I store all these usernames in a CSV file after removing duplicates. 2. Create several API keys for myanimelist's API, and asynchronously execute GET requests that take usernames as input and return said user's anime reviews as output. This is a type of web scraping. I was able to do about 4 GET requests per second at best after finding a rotating proxy server and rotating 3 API keys. This does not violate myanimelist.com's terms of service. My code did keep stopping due to rate limiting, so I manually retried each time until it completed. 3. Occasionally during step 2, the code stores the user data in a `user_records` table in sqlite to prevent overflowing my computer's memory. The final table ended up being over 3 gigabytes. 4. Once I had gotten data for all the usernames, verify that there is no duplicated data with SQL and delete duplicated rows 5. Get metadata for all the distinct animes that were reviewed by the 900,000 users and store in another SQL table `anime_info`. Every anime in `anime_info` has an integer `id` that corresponds to the `anime_id` column in user_records. A lot of anime has missing fields, which just become null in sqlite. The metadata includes official and English titles, start dates, end dates, images, genres, studios, the number of scoring users, and more. There ended up being only being ~12000 unique animes in my data, which is convenient because the entire table easily fits in memory. 6. Create SQL indexes on specific columns in `user_records` table to speed up queries. ## Goals Before diving into data exploration, I want to explain some of my goals in this project besides finding cool associations.

1. I will do whatever I can in R, but there are certain things like web scraping and interacting with sqlite that I think python is better for, so a few of my results will come from python, but they are based on the same data.
2. I want to practice making and optimizing complex SQL queries that are time efficient and return result sets that are memory-efficient. This is why SQL will do the majority of the data wrangling - I simply wanted to do it that way, even if it was overcomplicated. Some of the statistics I calculate with SQL are reported directly on myanimelist.com's website, but I derive them myself for fun.

3. I want to connect the phenomenons I see in the data with some personal experience I gained from watching anime for a couple years. For example, if I find a correlation in the ratings between two animes that are very similar, I plan to consider all the possible things that made them more similar than other combinations of animes, even if those things are subjective or not provable.
4. Because my sample of anime reviews comes from a website with many dishonest or inconsistent users, there is bound to be some bias and human error in the data. I want to consider how user habits may impact the significance of certain conclusions. One main goal of the project in general is to determine “what goes into” an anime review - both directly and indirectly. The users in my data are probably not perfectly representative of the general public given that they exist on a website dedicated to anime. That’s one of the facts I will ignore since I have no way to adjust to it.

Data Exploration

Genres

The first factor I want to explore are anime genres, since I believe genres are probably the #1 factor that people consider when choosing anime to watch. Over the course of anime history, various genres have rose to prominence.

Hide

```
library(RSQLite)
```

Here we do a little aggregation to reduce the size of the result set

Hide

```
con <- dbConnect(SQLite(), dbname = "animeDB2.sqlite3")
query = "Select genres,count(id) c from anime_info group by genres"
genresDF <- dbGetQuery(con,query)
head(genresDF)
```

	genres <chr>	<
1	*NA*	4
2	Action	5
3	Action;Adventure	1
4	Action;Adventure;Avant Garde;Historical;Military;Mystery;Shounen;Supernatural	
5	Action;Adventure;Cars;Comedy;Kids;Police	
6	Action;Adventure;Cars;Comedy;Sci-Fi;Shounen	

6 rows

Then we distribute the counts to each individual genre. I could have simply loaded the entire dataset into R and used the table function, but that’s probably memory-intensive and no fun.

Hide

```
genres_counter <- c()

distribute <- function (row) {
  count <- row["c"][[1]]
  values <- noquote(strsplit(row["genres"], ';')[[1]])
  for (i in 1:length(values)) {
    if (is.element(values[i], names(genres_counter))) {
      #if genre is already in genres counter
      genres_counter[values[i]] <<-
        as.numeric(genres_counter[values[i]]) + as.numeric(count)
    } else if (is.null(names(genres_counter))) {
      #if genres counter is empty
      genres_counter <<- c(count)
      names(genres_counter) <<- c(values[i])

    }
    else {
      genres_counter <<-
        c(genres_counter, count) #creates new genre at the end with value of the first cou
nt
      names(genres_counter)[length(genres_counter)] <<- values[i]
    }
  }
}
#values = rep(c("romance", "mystery", "drama"), 10)
apply(genresDF, MARGIN = 1, distribute)
```

```
NULL
```

Hide

```
genres_counter
```

```
         <NA>        Action     Adventure  Avant Garde    Historical 
        " 41"        "3395"        "2377"        "368"         "779" 
     Military       Mystery       Shounen  Supernatural          Cars 
        "522"         "669"        "1787"       "1275"          "85" 
       Comedy          Kids        Police        Sci-Fi         Mecha 
       "4615"         "912"         "216"       "2213"         "969" 
       Sports        Demons         Drama         Ecchi        Horror 
        "553"         "382"        "2104"        "524"         "382" 
      Romance       Fantasy   Super Power         Josei  Martial Arts 
       "1641"        "2567"         "526"         "87"         "353" 
         Game        Parody       Vampire         Harem        School 
        "331"         "464"         "130"        "313"        "1368" 
Psychological         Music         Space        Seinen       Gourmet 
        "333"        "1484"         "422"        "719"          "85" 
      Samurai Slice of Life        Shoujo      Suspense     Boys Love 
        "161"        "1463"         "636"        "128"          "73" 
   Girls Love     Work Life       Erotica 
         "67"        "   1"           "2" 
```

Hide

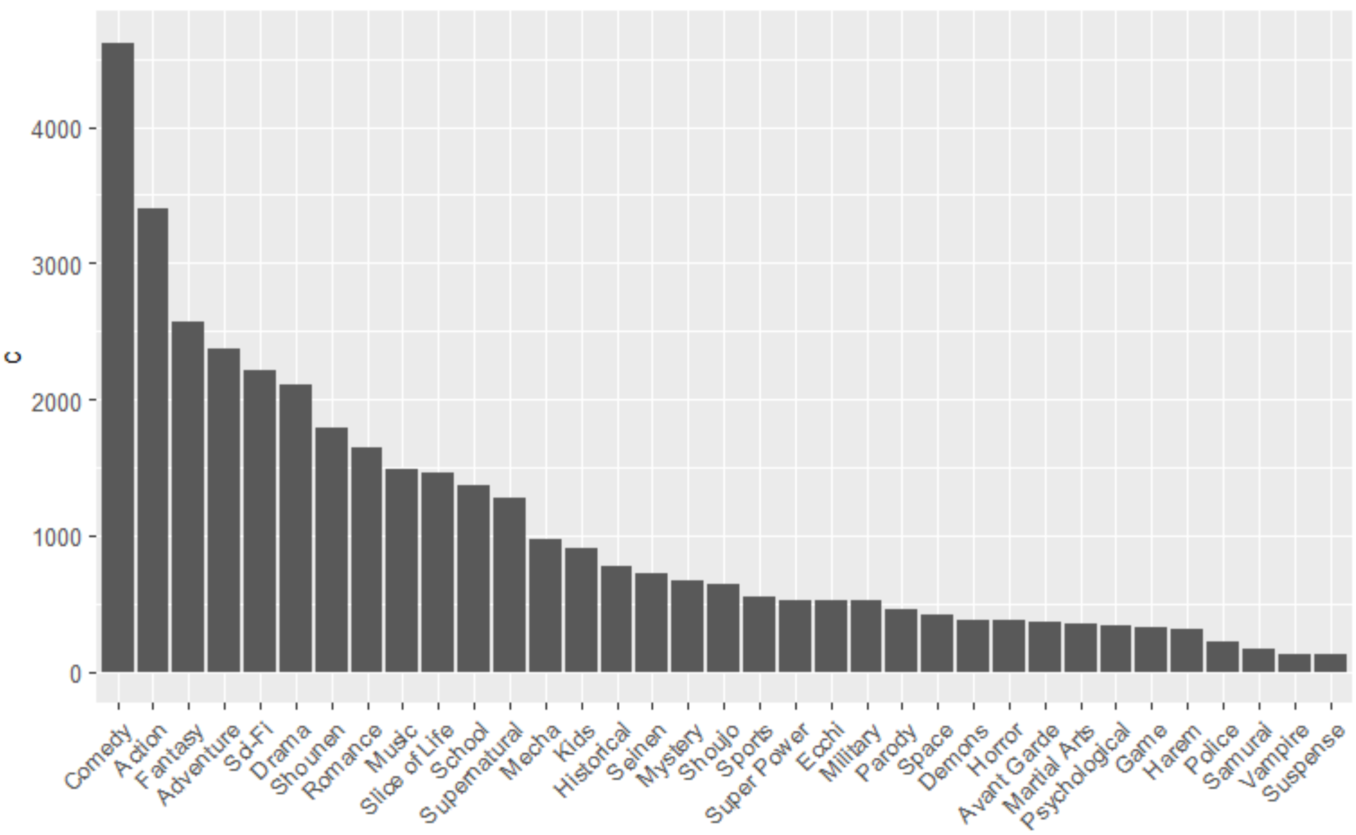
```
library(ggplot2)
genres_counter2 = subset(data.frame(c = c(as.numeric(genres_counter)), name = names(genres
_counter)),c>100)
genres_counter2 = genres_counter2[order(-genres_counter2$c),]
genres_counter2$name = factor(genres_counter2$name,levels=genres_counter2$name)
#genres_counter2 = genres_counter2[order(genres_counter2$c),]
genres_counter2
```

	c <dbl>	name <fctr>
11	4615	Comedy
2	3395	Action
22	2567	Fantasy
3	2377	Adventure
14	2213	Sci-Fi
18	2104	Drama
8	1787	Shounen
21	1641	Romance
32	1484	Music
37	1463	Slice of Life

1-10 of 35 rows Previous **1** 2 3 4 Next

Hide

```
p <- ggplot(data=genres_counter2,aes(x=name,y=c)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=45,hjust=1,vjust=1))+
  theme(axis.title.x=element_blank())
p
```



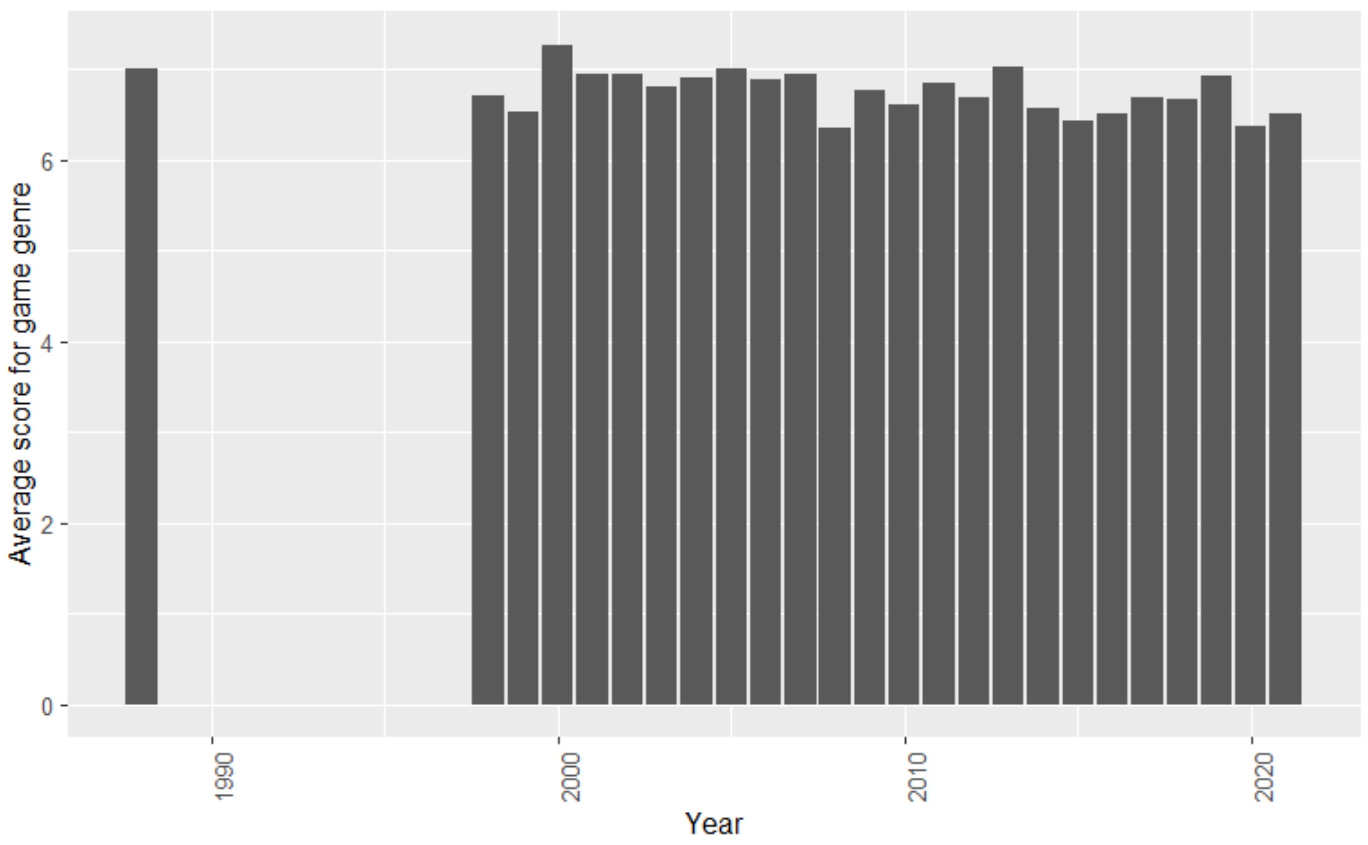
Hide

```
#barplot(genres_counter2[genres_counter2>100],las=2,cex.names = .5)
```

Here I should clarify that genres are not independent and are not mutually exclusive either. Animes typically are assigned anywhere from 1-8 genres by myanimelist completely manually (there is no official standard). It is very common to see Drama and Romance together, or Slice of Life and Romance. Likewise, Action and Adventure are a very common pair. I am curious about the developement of certain genres over time. Let's find the average score for the game genre over time. Note: certain anime's popularity will not skew this SQL query since I am averaging scores on a per-anime basis, not a per-user basis.

Hide

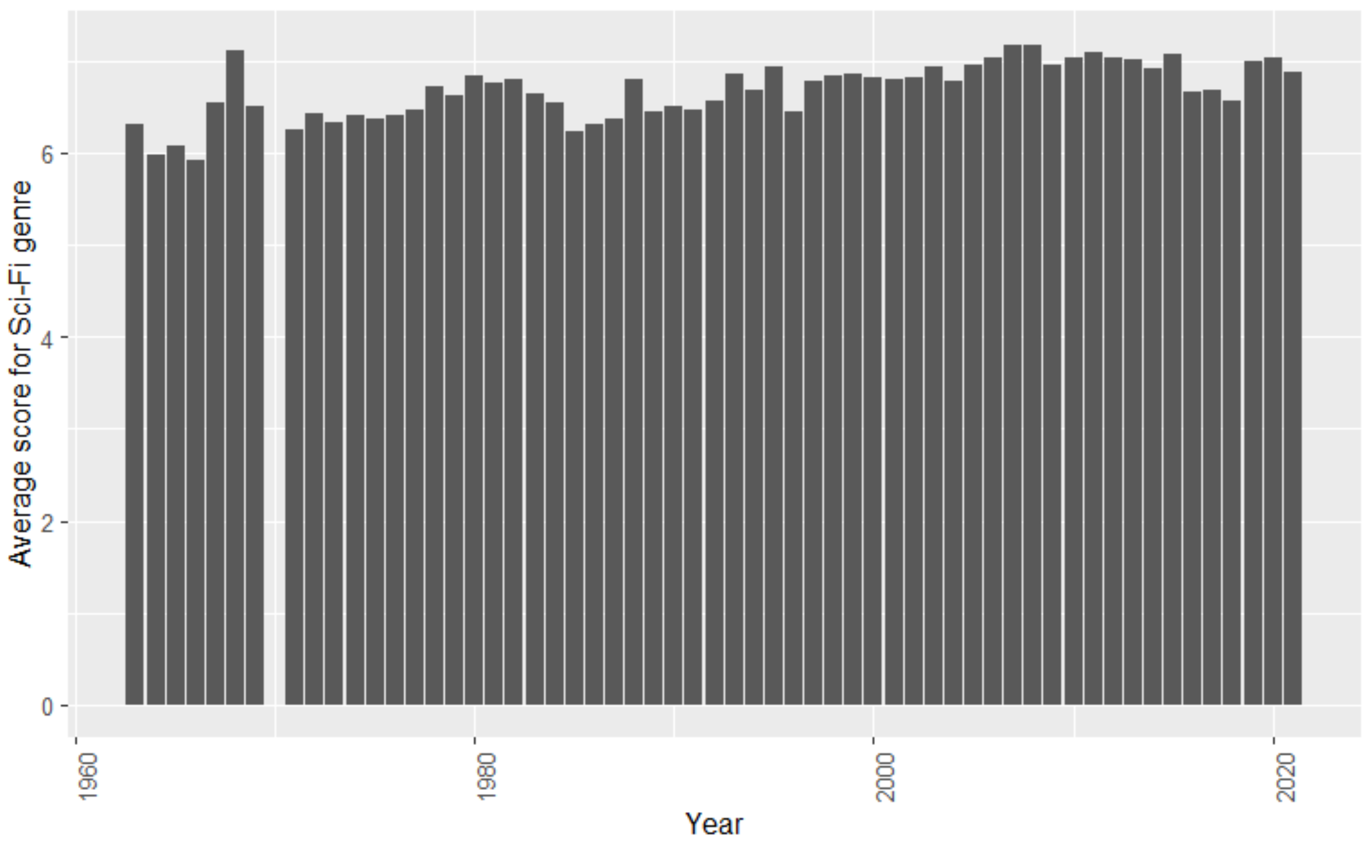
```
query = "select avg(average) y,(cast(strftime('%Y',a.start_date) as int)) x from
(select avg(score) average,anime_id from user_records u group by anime_id)  inner join ani
me_info a on anime_id = a.id and a.genres like '%Game%'
and cast(strftime('%Y',a.start_date) as int) > 0
group by strftime('%Y',a.start_date) "
gamesDF <- dbGetQuery(con,query)
p <- ggplot(data=gamesDF,aes(x=x,y=y)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=1))+
  xlab("Year") + ylab("Average score for game genre")
p
```



Hmm not very exciting. What about the Sci-Fi genre?

Hide

```
query = "select avg(average) y,(cast(strftime('%Y',a.start_date) as int)) x from
(select avg(score) average,anime_id from user_records u group by anime_id)  inner join ani
me_info a on anime_id = a.id and a.genres like '%Sci-Fi%'
and cast(strftime('%Y',a.start_date) as int) > 0
group by strftime('%Y',a.start_date) "
gamesDF <- dbGetQuery(con,query)
p <- ggplot(data=gamesDF,aes(x=x,y=y)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=1))+
  xlab("Year") + ylab("Average score for Sci-Fi genre")
p
```



Well, there certainly appears to be some trend before the year 2000, but it is hard to make a claim yet since we do not know if the quality of anime in general has increased throughout the years.

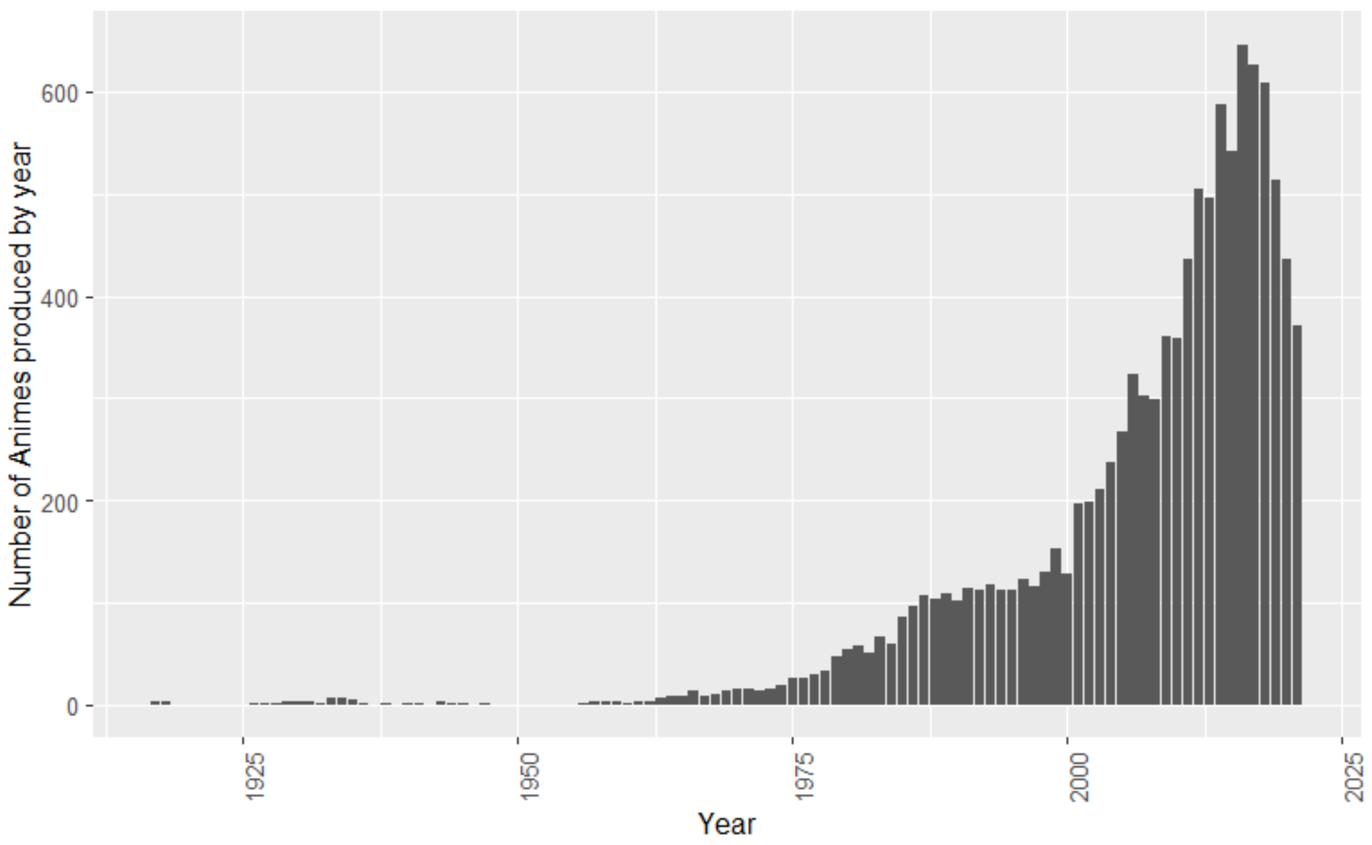
I also want to check the average rating for each genre. Surely, there are some genres that are liked more than others. However, because the myanimelist.com system often adds certain groups of genres to animes, analyzing them it probably wouldn’t do much good to compare any statistics for individual genres with my current data.

Viewership over time

Let’s see how many animes were produced each year:

Hide

```
query3 = "select cast(strftime('%Y',start_date) as int) year,count(id) count from anime_in
fo group by year having year > 0"
yearsDF <- dbGetQuery(con,query3)
p <- ggplot(data=yearsDF,aes(x=year,y=count)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=1))+
  xlab("Year") + ylab("Number of Animes produced by year")
p
```



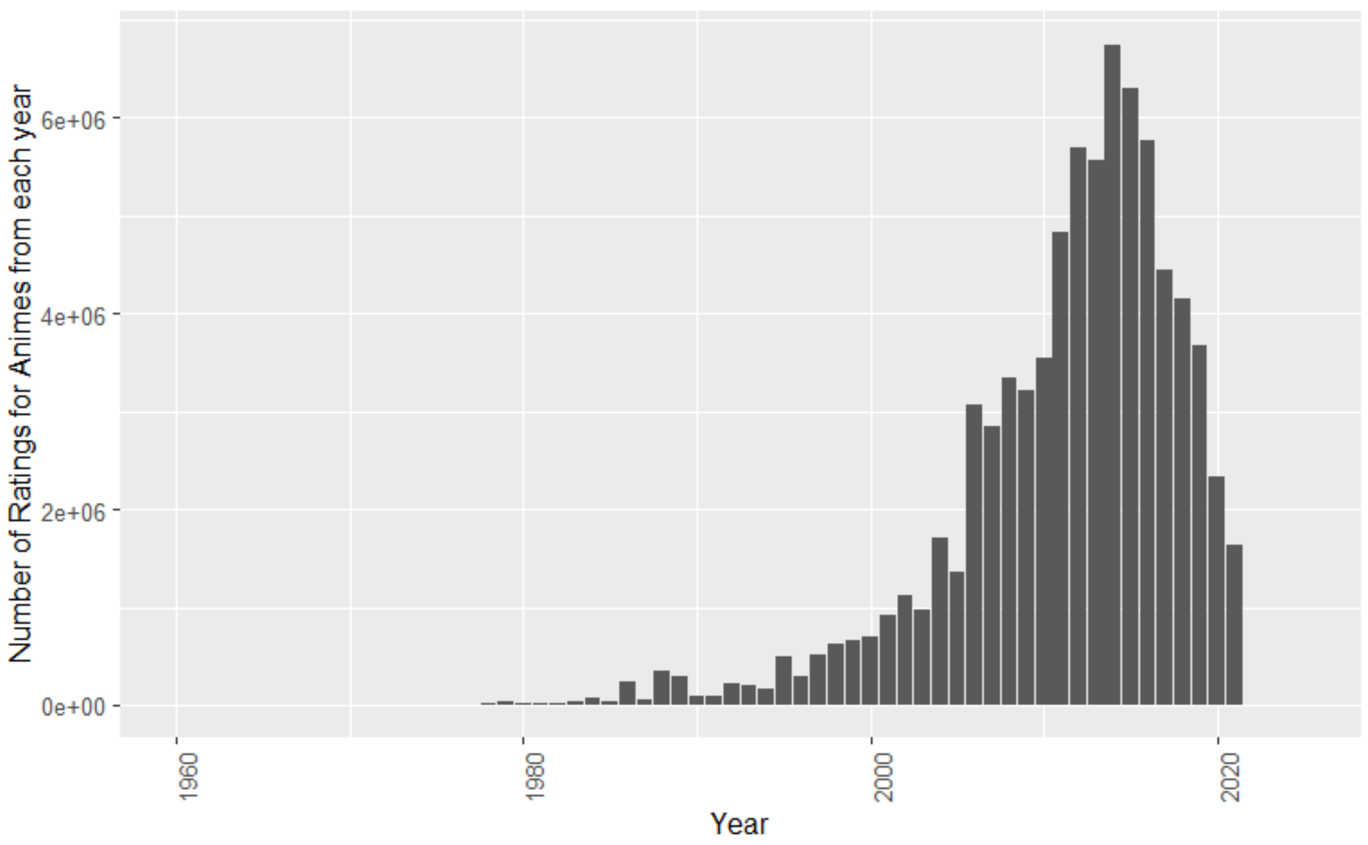
Wow, barely any anime existed before 1970, and there’s a obvious spike after 2000. Can we even trust the trends we observe before the year 1970 with so little content being produced? We should also check the level of user interaction with anime over time. In other words, we can count the total number of ratings on the site all animes published in a particular year and make a plot over time.

Hide

```
query4 = "select cast(strftime('%Y',a.start_date) as int) year,sum(count) s from
(select u.anime_id,count(u.score) count from user_records u group by u.anime_id) inner joi
n anime_info a on anime_id = a.id
group by cast(strftime('%Y',a.start_date) as int) having year > 0"
yearsDF2 <- dbGetQuery(con,query4)
p <- ggplot(data=yearsDF2,aes(x=year,y=s)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=1))+
  xlab("Year") + ylab("Number of Ratings for Animes from each year") +
  xlim(1960, 2025)

p
```

```
Warning: Removed 24 rows containing missing values (position_stack).
Warning: Removed 1 rows containing missing values (geom_bar).
```



As expected, the shape of the two plots are extremely similiar. It makes sense that on years where more animes are produced, more reviews are given to those animes. Some takeaways are how the COVID19 pandemic clearly decreased anime production starting from 2020. However, the decreased number of ratings for those years of anime is likely due to the fact that those shows are still relatively fresh and not all users have had an equal chance to view them as the older animes.

Data Analysis

Experience and Anime Eras vs ratings

With the exploration phase over, I have some questions I think I can answer with ANOVA and correlations. First, I believe that as a person watches more and more shows, they start to get disinterested and even run out of shows that suit their taste. **In other words, I suspect there is an inverse relationship between average anime rating and number of animes a user has watched.** Another factor I must keep in mind, however, is that people who have watched more anime are more likely to be have been watching it for longer (eg. they are likely older) and thus have been exposed to older anime. For example, one legendary user on myanimelist is named ‘Archeon.’ Born in 1975, he has lived twice as long as me and he claims to have completed 1338 anime (imagine almost a year’s worth of nonstop watching). He is extremely active as an anime critic, and his average rating of 6.84 is significantly lower than my 7.64, though I would even call him forgiving compared to some other critics who have seen thousands of shows. Our data exploration has shown that mean anime ratings do fluctuate throughout the years. so **is the “era” that a user watches anime in also a good predictor of the user’s ratings?** It may even be a better predictor than anime experience.

Note: I will now refer to the average release year of user’s completed animes as the user’s “era” We will make a linear regression model based on these two factors

Hide

```
query5 = "select avg(u.score) av,count(u.score) c,avg(cast(strftime('%Y',a.start_date) as
int)) y from user_records u
inner join anime_info a on a.id = u.anime_id and a.media_type='tv' and cast(strftime('%Y',
a.start_date) as int) > 0 group by u.username"
experienceDF <- dbGetQuery(con,query5)
experienceDF$y = experienceDF$y - 2000 #setting better baseline
dim(experienceDF)
```

```
[1] 679216      3
```

Hide

```
model = lm(av~c+y,data = experienceDF)
```

```
Warning: call dbDisconnect() when finished working with a connection
```

I would like to point out that SQL is extremely slow at "funneling" data from its output into R dataframes. I've experienced the same in python. For over 600,000 rows, the query itself takes under a second, but actually fetching the data into R takes about 10 minutes on my laptop. In the rest of this project, I will have even larger result sets and will likely implement random sampling. This will make the queries themselves much less efficient, but it will speed up fetch times.

Hide

```
summary(model)
```

```


Call:
lm(formula = av ~ c + y, data = experienceDF)

Residuals:
    Min      1Q  Median      3Q     Max 
-7.5448 -0.5267  0.0021  0.5697  3.8742 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  8.424e+00  3.141e-03 2682.26   <2e-16 ***
c           -2.991e-03  1.149e-05 -260.31   <2e-16 ***
y           -1.336e-02  2.847e-04  -46.92   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9052 on 679213 degrees of freedom
Multiple R-squared:  0.1092,	Adjusted R-squared:  0.1092 
F-statistic: 4.165e+04 on 2 and 679213 DF,  p-value: < 2.2e-16
```

Judging by the R squared, this model does not explain much of the variability in the user's scores, but that's expected. I don't expect the era and experience of a user to be the largest factors in their enjoyment of anime. I just wanted to see if there were significant trends.

Hide

```
round(coef(model),3)
```

```
(Intercept)           c           y 
      8.424      -0.003      -0.013 
```

So far, we know that both of the predictors have a negative slope, and the values of them suggests that for ever 333 anime a user watches, their average score is expected to go down by 1. For every 10 years a user’s “era” increases, their average rating is expected to decrease by .13. Quite fascinating results!

Hide

```
anova(model)
```

```
Analysis of Variance Table

Response: av
              Df Sum Sq Mean Sq F value    Pr(>F)    
c              1  66441   66441 81091.1 < 2.2e-16 ***
y              1   1804    1804  2201.4 < 2.2e-16 ***
Residuals 679213 556501       1                      
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The F tests in anova confirm our findings are significant. Let’s make the model predict average scores for my data, and then for an imaginary user who has watched 1200 anime and whose “era” was around 2000. The point is that this user is more experienced and older than me.

Hide

```
#keep in mind y is years after 2000
predict(model,data.frame(c=88,y=20),interval = "confidence") # me
```

```
       fit      lwr      upr
1 7.893558 7.887972 7.899143
```

Hide

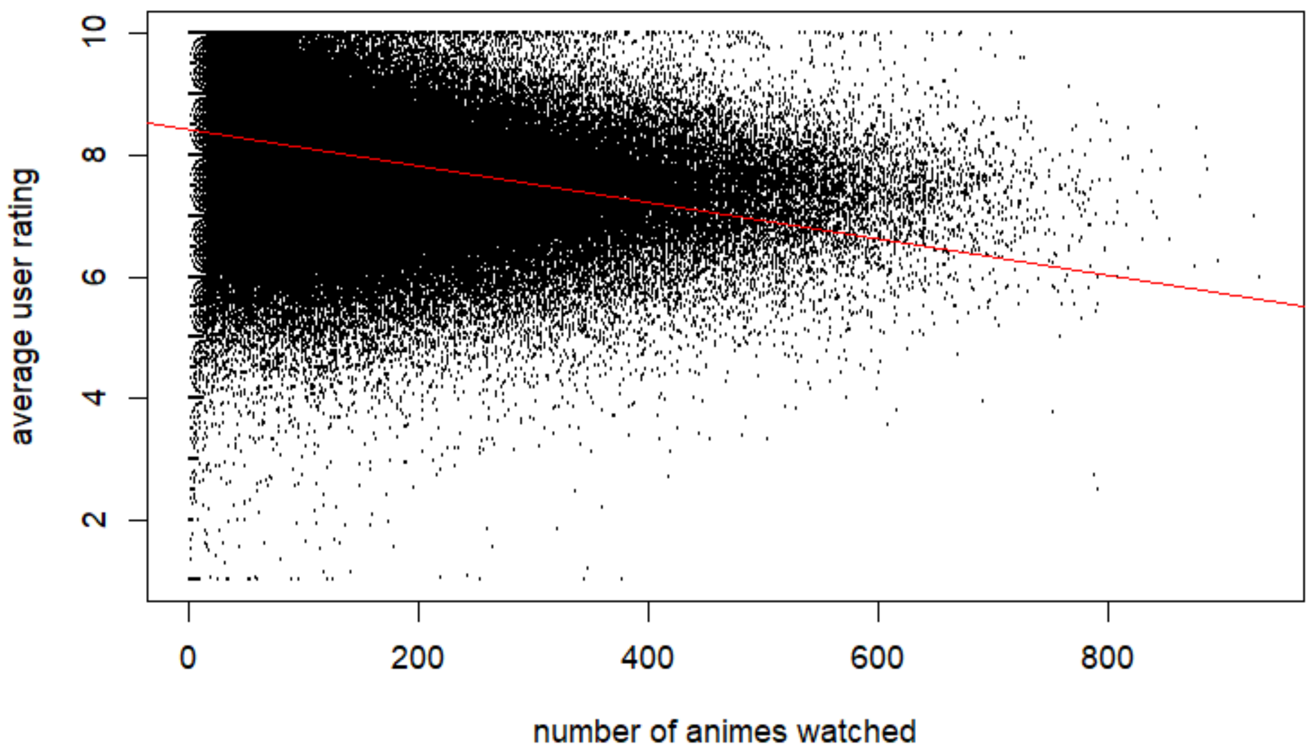
```
predict(model,data.frame(c=1200,y=0),interval = "confidence") #other guy
```

```
       fit      lwr      upr
1 4.834901 4.807464 4.862338
```

Look at the stark difference between our predicted average scores! I am also curious how the line of best fit compares to the scatter plot of data. I am pretty certain the trend is not linear. It probably is a curve where slope starts negative and increases as c (number of animes watched) increases. After all, most people wouldn’t watch thousands of shows if they completely despised them, right? There has to be a point of diminishing losses in enjoyment.

Hide

```
plot(experienceDF$c,experienceDF$av,cex=.1,xlab = "number of animes watched",ylab="average user rating")
abline(a=coef(model)[1],b=coef(model)[2],col="red")
```



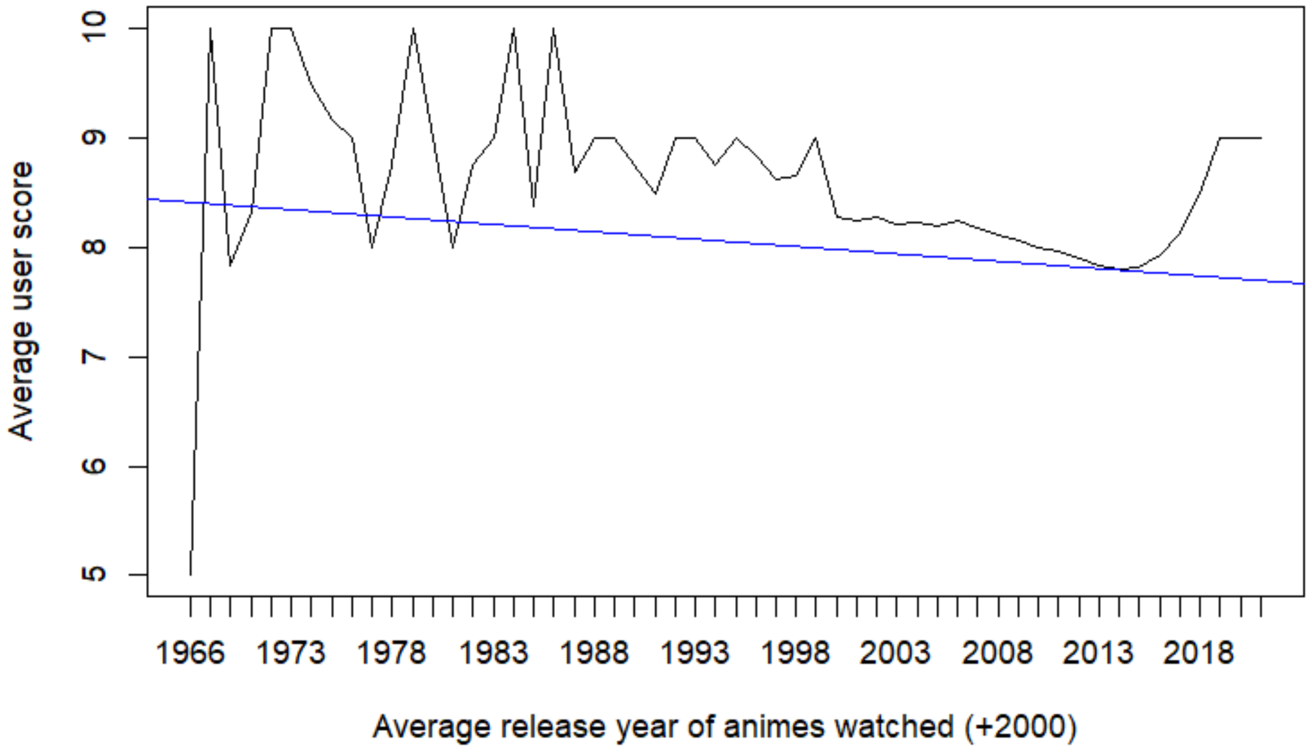
I think the biggest takeaway from this plot is the R squared value and how it is visualized. Relative to the slope, the variation around the line of best is extremely large, indicating how little about average scores the # of animes watched explains.

Hide

```
#plot(experienceDF$y,experienceDF$av,cex=.1,xlab="Average release year of animes watched (+2000)")
#head(experienceDF)
back = tapply(experienceDF$av,as.integer(experienceDF$y),median)
plot(back,type="l",xlab="Average release year of animes watched (+2000)",xaxt="n",ylab="Average user score")
axis(1,labels=as.integer(names(back))+2000,at=1:length(back))
```

Hide

```
abline(a=coef(model)[1],b=coef(model)[3],col="blue")
```



The same goes for the “era” predictor. I would not trust this model to predict someone’s average rating based on sole these factors. I also need to verify which predictor is actually doing more of the explaining, so I will manually “step” for that and make a model only based on # of animes watched.

Hide

```
modelSmaller = lm(av~c,data = experienceDF)
summary(modelSmaller)
```

```


Call:
lm(formula = av ~ c, data = experienceDF)

Residuals:
    Min      1Q  Median      3Q     Max 
-7.2902 -0.5306  0.0020  0.5729  3.9590 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  8.293e+00  1.461e-03  5675.1   <2e-16 ***
c           -3.141e-03  1.105e-05  -284.3   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.9066 on 679214 degrees of freedom
Multiple R-squared:  0.1063,	Adjusted R-squared:  0.1063 
F-statistic: 8.083e+04 on 1 and 679214 DF,  p-value: < 2.2e-16
```

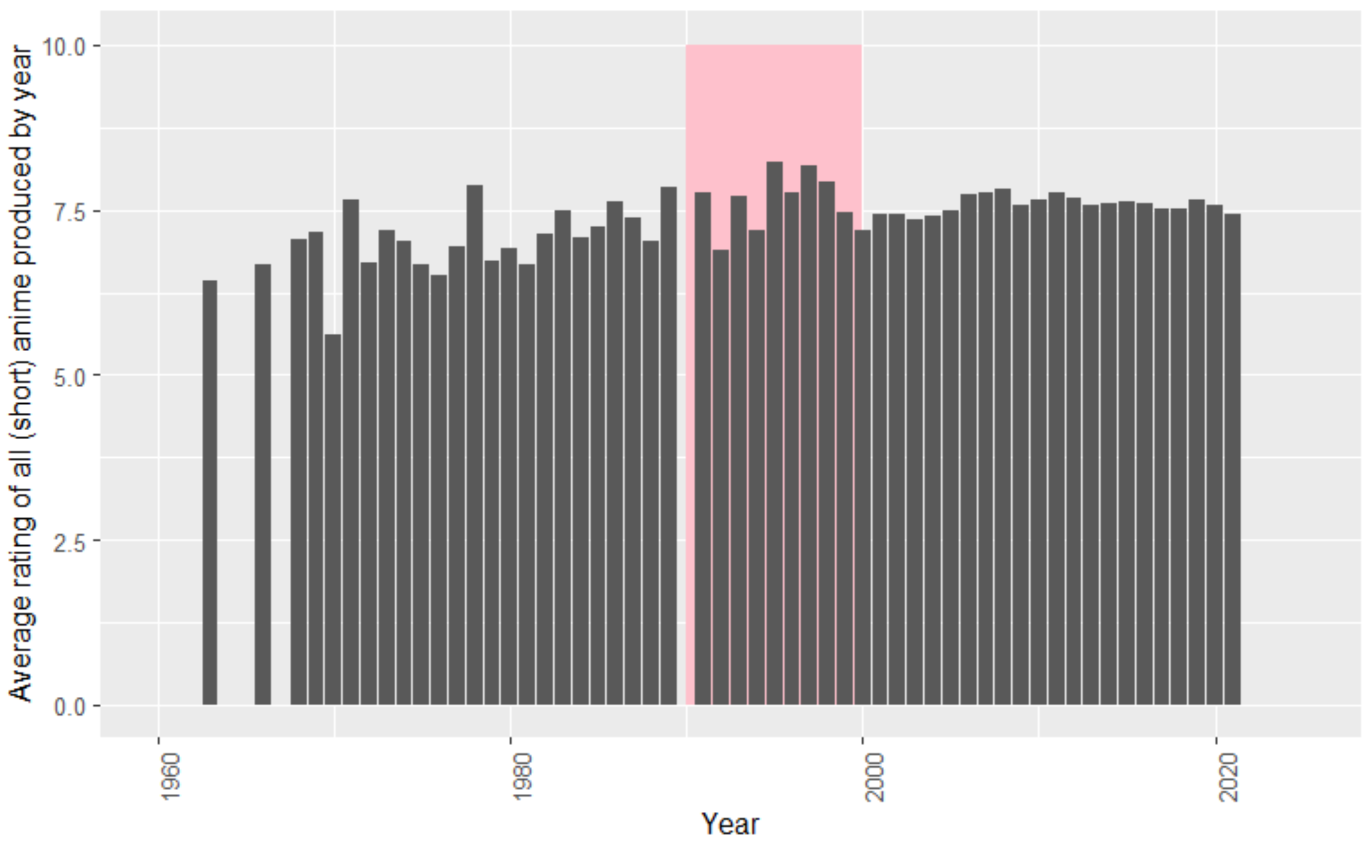
Wow, the R squared value only decreased by around .03. I guess the “era” of anime a user watches barely explained their average score compared to how many animes they have watched. Another thing to point out is the residual standard error. Because all anime ratings are integers between 0 and 10, this error of .9 is obviously high (it's almost a tenth of the entire spectrum). This further demonstrates how little this model explains on its own. I think calculating the standard deviation of all of the anime scores would be beneficial to compare residual errors against.

Another interesting consideration is that a linear model may not be appropriate for the relationship between era and average score. Many agree that anime had its golden age during the 1990's, with many of the classics from that decade still being enjoyed and given top ratings today. Thus, it is reasonable to guess that an upside-down parabola would be a better fit for predicting average scores based on anime "era." However, I do not know how to fit or interpret curved regression models. The graph below will illustrate the golden era in the 1990's. This graph also slightly contradicts the pattern in my other rating by year graph, but that graph was inconsistent for years before 1990 because very few anime existed back then that fit my criteria, which was relatively short TV shows.

Hide

```
query6 = "Select avg(u.score) average,substr(a.start_date,1,4) dates from (
user_records u inner join
anime_info a on u.anime_id = a.id
and a.media_type='tv' and Cast (
    JulianDay(a.end_date) - JulianDay(a.start_date)
 As Integer) < 183) -- this means shows that only last 26 or less episodes
group by substr(a.start_date,1,4) --order by dates"
goldenage = dbGetQuery(con,query6)
goldenage$dates = as.numeric(goldenage$dates)
p <- ggplot(data=goldenage,aes(x=dates,y=average)) +
  geom_rect(data=NULL,aes(xmin=1990,xmax=2000,ymin=0,ymax=10),
            alpha=0.3,fill="pink") +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=1))+
  xlab("Year") + ylab("Average rating of all (short) anime produced by year") +
  xlim(1960, 2025)


p
```



Hide

```
sdquery = "SELECT AVG((u.score - sub.a) * (u.score - sub.a)) as var from user_records u,
(SELECT AVG(score) AS a FROM user_records) AS sub;"
variance = dbGetQuery(con,sdquery)
sqrt(variance[1,1])
```

```
[1] 1.682516
```

Now we can compare Residual SE for scores to 1.68 as a general guide. The .9 Residual SE from the previous model is definitely lower, but not by enough in my opinion.

###Correlations between scores of different anime We all know that our current preference for tv shows can be at least partly explained by our past watching decisions and reviews. That's how Netflix can boast that 80% of its watched content is based on its algorithmic recommendations. With anime, I think there are bound to be certain types of shows whose ratings are strongly correlated. From now on, I will refer to the pearson correlation between all of the scores given to two shows by all users who have watched both shows as the "correlation between animes." In other words, if anime1 and anime2 are given scores 5,7,6 and 5,8,6 respectively by the same 3 people, these animes are more strongly correlated than anime3, which was given scores 7,4,3 by the same 3 people.

I predict that shows that contain the same genres will have stronger correlations between their scores.

I also expect to see different seasons from the same show as very strongly correlated. This intuitively makes sense since most of the content and target audience of shows stays the same across multiple seasons. Myanimelist.com lists different seasons of a show as separate shows, so I will manually browse the data to see if correlated shows are indeed different seasons of the same show.

Lastly, **I predict that shows from the same year will be strongly correlated.** My logic behind this is that as a person's life changes, so do their preferences. However, within one year or a few months, a typical person's life does not change much, thus making their ratings for the shows during that time more similar. Also, although this is anecdotal evidence, many myanimelist.com users do not update their ratings regularly. Instead, they log onto the site about a few times a year just to log the shows they remember watching. Therefore, these types of people are more likely to only remember shows by the approximate time they watched them and not give them distinctive ratings based on content. Some even argue that this is the correct way to use the website and that more active users are pretentious or gatekeeping, but that's a discussion for another day.

Hide

```
library(reticulate)
use_python("C:/Users/daniel/anaconda3/envs/data/python.exe",required=T)
```

This code connects to the SQL database in python

Hide

```
import sqlite3
con = sqlite3.connect("animeDB2.sqlite3")
#con.execute("select * from anime_info where score > 9.0").fetchall()
```

This code defines a custom aggregation function that takes two lists and finds the correlation between them. It then registers the function to the SQLite interpreter.

Hide

```
import numpy
class myCorr:
    def __init__(self):
        self.count = 0
        self.list1 = []
        self.list2 = []
    def step(self, value1,value2):
        self.list1.append(value1)
        self.list2.append(value2)
    def finalize(self):
        #print(self.list1[:10])
        #print(self.list2[:10])
        if len(self.list1) and len(self.list2):
            self.correlation = numpy.corrcoef(self.list1,self.list2)[0][1]
        else:
            self.correlation = 0.0
        return self.correlation

con.create_aggregate("corr", 2, myCorr)
```

Now, this long SQL query will find every user who has watched a specific show. It will then create pairs of integers that represent the user's rating for said show (id1) , and every other show they have watched (id2). It will repeat this for every user, creating a long chain of rating pairs. It will then group those pairs by the id2 column and apply the correlation function to find the pearson correlation between the ratings for id1 and id2. Then, if we sort in descending order, we get the animes most correlated with each other on top of the result set. One consideration is that this query takes an extremely long time for popular shows since they can have millions of rating pairs, so for shows that exceed an arbitrary threshold of ratings, I will take a random subset of ratings. I have wrapped up all of this in a python function that can takes an anime id as an input and gets the animes highest correlated to it.

Hide

```
import pandas as pd
from datetime import datetime
cur = con.cursor()
def calcCorrelations(anime_id):
    cur.execute(f"select num_scoring_users from anime_info where id = {anime_id}")
    occurances = cur.fetchone()[0]
    print(f"num_scoring_users: {occurances}")
    small = f"""select * from
    (select corr(score1,score2) scores,anime_id from
    ((select anime_id as anime_id2,username as username2,score as score2  from user_record
s where anime_id in({anime_id}))
    inner join  (select anime_id,username,score as score1 from user_records)
    on  username = username2) group by anime_id) inner join (select anime_title,id from an
ime_info where num_scoring_users > 4000)  on anime_id=id order by scores desc limit 100
0;"""
    large =    f"""select * from
    (select corr(score1,score2) scores,anime_id from
    ((select anime_id as anime_id2,username as username2,score as score2  from user_record
s where anime_id in({anime_id}) order by random() limit 15000)
    inner join  (select anime_id,username,score as score1 from user_records)
    on  username = username2) group by anime_id) inner join (select anime_title,id from an
ime_info where num_scoring_users > 10000)  on anime_id=id order by scores desc limit 100
0;"""
    start = datetime.now()
    if occurances > 17000:
        print("large")
        data = cur.execute(large).fetchall()
    else:
        print("small")
        data = cur.execute(small).fetchall()
    print("done")
    #print(asizeof.asizeof(result))
    print((datetime.now()-start))
    return pd.DataFrame(data,columns=["correlation","id","title","id2"])
corr20 = calcCorrelations(61)
```

```
C:\Users\daniel\ANACON~1\envs\data\lib\site-packages\numpy\lib\function_base.py:2642: Runt
imeWarning: invalid value encountered in true_divide
  c /= stddev[:, None]
C:\Users\daniel\ANACON~1\envs\data\lib\site-packages\numpy\lib\function_base.py:2643: Runt
imeWarning: invalid value encountered in true_divide
  c /= stddev[None, :]
C:\Users\daniel\ANACON~1\envs\data\lib\site-packages\numpy\lib\function_base.py:2634: Runt
imeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
C:\Users\daniel\ANACON~1\envs\data\lib\site-packages\numpy\lib\function_base.py:2493: Runt
imeWarning: divide by zero encountered in true_divide
  c *= np.true_divide(1, fact)
C:\Users\daniel\ANACON~1\envs\data\lib\site-packages\numpy\lib\function_base.py:2493: Runt
imeWarning: invalid value encountered in multiply
  c *= np.true_divide(1, fact)
```

```
num_scoring_users: 89484
large
done
0:00:06.595363
```

The reticulate library lets us read python pandas dataframes as R dataframes

Hide

```
quit
head(py$corr20)
```

	correlation <dbl>	id <dbl>	title <chr>	id2 <dbl>
1	1.0000000	61	D.N.Angel	61
2	0.7561539	40842	Idoly Pride	40842
3	0.6795894	38198	Nanatsu no Taizai: Eiyuu-tachi wa Hashagu	38198
4	0.6637495	36688	Shinmai Maou no Testament Departures	36688
5	0.6635295	33242	IS: Infinite Stratos 2 - Infinite Wedding	33242
6	0.6512184	40485	Strike the Blood IV	40485

6 rows

Reading the head of this dataframe gives us the correlation coefficients for the animes best correlated to the anime with id 61. After some manual review, these correlation coefficients seemed rather high, so I manually checked how many pairs of ratings they were based on. Most were only watched together with show 61 less than 1000 times in my dataset. Also, if you notice, I excluded certain animes from my results that weren't popular enough with the last inner join, but that alone doesn't make everything significant. That's because there is a good chance, out of thousands of shows, that at least a few are highly correlated with each-other by pure chance. I don't know if 1000 people is enough to make the correlation coefficient significant, so let's tackle this with R instead of python. This will be much slower due to the larger amount of data I have to fetch because RSqlite does not support custom aggregate functions, and because R is calculating significance for each correlation. For a reference alpha level, I will take .05 and divide by (# of valid shows) to the total combinations of 2 shows. For example, if I select only shows with at least 5000 ratings, only 4315 of such shows exist, so my alpha level would be .05/(4315) = 1.16e-5. I am not sure if this is a fair way to do this, but alpha levels are circumstantial anyway and for this situation, this makes sense. It also is very similar to bonferroni's correction for t tests.

In this query, I will try to do the same process as before, but instead I will do the aggregation in R with R's corr.test function to get significance. Also, notice the "order by random() limit n)" that is in this query and the previous. What this does is get n random users who have watched a certain anime and get all of their other ratings. I believe this small random sample of users will be representative of over 600,000 users, and will help the query run much faster. However, one side effect of this method is that by limiting our users, we limit the range of shows we can compare, since typically fewer users will have seen fewer distinct animes collectively. Therefore, more popular shows will likely be favored in this method, but that has technically always been the case, since my data is a sample of all users on myanimelist.com.

Hide

```
anime_id = 20899
possibleusers = dbGetQuery(con,"select distinct username from user_records")
randomUsers = sample(possibleusers$username,size=100000)
query7 = paste("
select u.score score1,f.score score2,f.anime_id from user_records f
inner join (select * from user_records where anime_id = ",anime_id," and username in(",pas
te(randomUsers,collapse=","),")) as u on u.username = f.username
inner join anime_info a on a.id=f.anime_id and a.num_scoring_users > 10000",sep=" ")
score.pairs <- dbGetQuery(con,query7)
agg = function(x1){
  c1 = as.numeric(sapply(strsplit(x1,";"),function(b){return(b[1])}))
  c2 = as.numeric(sapply(strsplit(x1,";"),function(b){return(b[2])}))

  if(length(na.omit(c1)) < 3){
    return (c(0.0,1.0))
  }
  t = cor.test(c1,c2)
  return (c(t$estimate["cor"],t$p.value))
}
```

At this point, I arbitrarily selected a sample size and minimum popularity level that made my SQL query run fast and give some consistent results.

Hide

```
score.pairs.agg = aggregate(paste(as.character(score1),as.character(score2), sep=";", coll
apse=NULL) ~ anime_id, data = score.pairs, FUN = agg,simplify = F)
score.pairs.agg = setNames(score.pairs.agg,c("id","correlation"))

#cor.test(iris$Sepal.Length,iris$Sepal.Width)$estimate["cor"]
score.pairs.agg$p.value =sapply(score.pairs.agg$correlation,
                                function(x) x[[2]])
score.pairs.agg$estimate =sapply(score.pairs.agg$correlation,
                                function(x) x[[1]])
head(score.pairs.agg)[1,2]
```

```
[[1]]
         cor
2.141883e-01 2.024966e-68
```

Hide

```
pair.names <- dbGetQuery(con,"select id,anime_title from anime_info")
score.pairs.agg2 = merge(score.pairs.agg,pair.names, by.x="id",by.y="id")
#Using p value calculated above 1.16e-5
score.pairs.agg.significant = subset(score.pairs.agg2,p.value<.0000116)
score.pairs.agg.significant[order(-score.pairs.agg.significant$estimate),]
```

id	correlation	p.value	estimate
<int>	<list>	<dbl>	<dbl>

	id	correlation	p.value	estimate
	<int>	<list>	<dbl>	<dbl>
1856	20899	<dbl [2]>	0.000000e+00	1.0000000
2080	26055	<dbl [2]>	0.000000e+00	0.8182723
191	300	<dbl [2]>	6.984095e-07	0.6358572
1579	14719	<dbl [2]>	0.000000e+00	0.5995235
2418	32555	<dbl [2]>	8.244560e-12	0.5746763
965	5521	<dbl [2]>	2.559563e-12	0.5525357
1005	6098	<dbl [2]>	5.987529e-09	0.5510098
352	665	<dbl [2]>	1.171223e-61	0.5468848
2959	37991	<dbl [2]>	0.000000e+00	0.5439944
236	407	<dbl [2]>	3.153137e-06	0.5312082

1-10 of 2,322 rows | 1-5 of 5 columns Previous 1 2 3 4 5 6 ... 100 Next

After doing all of the aggregation, reducing the data, and finding correlations and significance, we can see that for the show "JoJo no Kimyou na Bouken Part 3: Stardust Crusaders", there are 5 seasons currently released, and every single one of them show up at the top pages of the correlations to the part 3, with significances well below the alpha level. This shows how different seasons of the same show tend to be correlated, but I need to demonstrate it on multiple shows to prove that this test isn't a fluke. To do this, I will select an arbitrary list of anime that have at least 3 seasons, and get the top 10 anime correlated with each. I will also choose the "baseline" anime to be the second season of each simply because it prevents wasting computational power on users who have only rated season one of a show.

Hide

```
multiseason = c(33486,23281)#,23847,28891,11597,15451,10030,24277)
top10s = list()

for (i in c(1:length(multiseason))){
possibleusers = dbGetQuery(con,paste("select distinct username from user_records where ani
me_id = ",multiseason[i],sep=""))
#this takes either a random 80,000 or all of the users possible, whichever is smaller
randomUsers = sample(possibleusers$username,size=min(c(50000,length(possibleusers$usernam
e))))
query7 = paste("
select u.score score1,f.score score2,f.anime_id from user_records f
inner join (select * from user_records where anime_id = ",multiseason[i]," and username in
(",paste(randomUsers,collapse=","),")) as u on u.username = f.username
inner join anime_info a on a.id=f.anime_id and a.num_scoring_users > 10000",sep=" ")
score.pairs <- dbGetQuery(con,query7)

agg = function(x1){
  c1 = as.numeric(sapply(strsplit(x1,";"),function(b){return(b[1])}))
  c2 = as.numeric(sapply(strsplit(x1,";"),function(b){return(b[2])}))

  if(length(na.omit(c1)) < 3){
    return (c(0.0,1.0))
  }
  t = cor.test(c1,c2)
  return (c(t$estimate["cor"],t$p.value))
}
score.pairs.agg = aggregate(paste(as.character(score1),as.character(score2), sep=";", coll
apse=NULL) ~ anime_id, data = score.pairs, FUN = agg,simplify = F)
score.pairs.agg = setNames(score.pairs.agg,c("id","correlation"))

#cor.test(iris$Sepal.Length,iris$Sepal.Width)$estimate["cor"]
score.pairs.agg$p.value =sapply(score.pairs.agg$correlation,
                                function(x) x[[2]])
score.pairs.agg$estimate =sapply(score.pairs.agg$correlation,
                                function(x) x[[1]])
pair.names <- dbGetQuery(con,"select id,anime_title from anime_info")
score.pairs.agg2 = merge(score.pairs.agg,pair.names, by.x="id",by.y="id")
#Using p value calculated by .01/3354 potential comparisons
score.pairs.agg.significant = subset(score.pairs.agg2,p.value<.01/3354)
score.pairs.agg.significant= score.pairs.agg.significant[order(-score.pairs.agg.significan
t$estimate),]
  top10s[[i]] = score.pairs.agg.significant$anime_title[1:10]
}
top10s
```

```
[[1]]
 [1] "Boku no Hero Academia 2nd Season"
 [2] "Boku no Hero Academia"
 [3] "Boku no Hero Academia 3rd Season"
 [4] "Boku no Hero Academia 4th Season"
 [5] "Boku no Hero Academia: Sukue! Kyuujo Kunren!"
 [6] "Boku no Hero Academia the Movie 1: Futari no Hero"
 [7] "Boku no Hero Academia 2nd Season: Hero Note"
 [8] "Boku no Hero Academia: Training of the Dead"
 [9] "Boku no Hero Academia the Movie: Futari no Hero Specials"
[10] "Boku no Hero Academia: Ikinokore! Kesshi no Survival Kunren"

[[2]]
 [1] "Psycho-Pass 2"
 [2] "Psycho-Pass Movie"
 [3] "Psycho-Pass: Sinners of the System Case.1 - Tsumi to Batsu"
 [4] "Aldnoah.Zero 2nd Season"
 [5] "Galilei Donna"
 [6] "Koukaku Kidoutai Arise: Ghost in the Shell - Border:3 Ghost Tears"
 [7] "Aldnoah.Zero"
 [8] "Psycho-Pass: Sinners of the System Case.2 - First Guardian"
 [9] "Psycho-Pass 3: First Inspector"
[10] "Papa no Iukoto wo Kikinasai!: Pokkapoka"
```

Even for those who don’t watch anime, it is obvious that parts of the same series were very highly correlated together. Just look at the titles. To reiterate how these results were generated, the SQL query got every user (from a random sample) who watched a particular anime I chose and made pairs of that rating and every other rating the user had. Then, the correlation algorithm found the correlation between every pair of animes and the significance. It filtered out insignificant values and ordered the dataframe with the highest correlation coefficients on top. The 10 animes with the highest correlation coefficients to the chosen anime were added to the list. The top 1 highest correlation coefficient is always for the original anime itself, since that correlation is always 1. I could have done this for more anime, but it would take a very long time to run.

Correlations between genres

The next step is to compare correlations between ratings within genres. **My goal is to find out whether the scores for animes within the same genre have a stronger correlation than the scores of some random sample of anime.** To do this, I will employ a similiar strategy as the last algorithm, except instead of targeting specific anime_ids, i will be taking a random sample of pairs of scores from the same user, making sure that the scores are both attributed to a specific genre of anime. One immediate issue with this test is it relies on genres being independent of each other, which is not the case at all. Like mentioned earlier, action tends to go with adventure, romance tends to go with drama, and so on going by the myanimelist.com system. To minimize the intersection of our samples, I will handpick a set of genres I personally think have very little in common and run tests on them. These genres will also have to be popular enough so that the samples are truly representative of their population. For now, my definition of a popular genre is one with at least 500 distinct “shows” in the database. The genres I will compare are Romance, Fantasy, and Action. Also, these genres encompass about half of all the entries in the database, so they are not “niche” and should make for interesting comparison. Genres that are more niche like Sports or Horror would definitely have significant differences.

Hide

```
#THIS IS SLOW
multiGenres = c("Romance")#,"Action","Fantasy",";")
corrTests = list()

for (i in c(1:length(multiGenres))){
possibleanimes = dbGetQuery(con,paste("select id from anime_info where genres like '%",mul
tiGenres[i],"%' and num_scoring_users > 10000",sep=""))
#this takes either a random 80,000 or all of the users possible, whichever is smaller
randomAnimes = sample(possibleanimes$id,size=min(c(2,length(possibleanimes$id))))
#query7 = paste("select u.score score1,f.score score2,f.anime_id from user_records f inner
join (select * from user_records where anime_id = ",multiseason[i]," and username in(",pas
te(randomUsers,collapse=","),")) as u on u.username = f.username inner join anime_info a o
n a.id=f.anime_id and a.num_scoring_users > 10000",sep=" ")
query8 = paste("select f.score score1,u.score score2,f.anime_id from ((select * from user_
records u where anime_id in (select id from anime_info a where genres like'%",multiGenres
[i],"%' and num_scoring_users > 10000)) as u inner join (select * from user_records f wher
e anime_id in (select id from anime_info a where genres like '%",multiGenres[i],"%' and id
in(",paste(randomAnimes,collapse=","),"))) as f on u.username = f.username and not f.anime
_id = u.anime_id) ",sep="")
score.pairs2 = dbGetQuery(con,query8)
score.pairs2 = score.pairs2[sample(nrow(score.pairs2), 100000),]
print("here")
corrTests[i] = list(cor.test(score.pairs2$score1,score.pairs2$score2,adjust="bonferroni",a
lpha=.05))
names(corrTests) = multiGenres
}
corrTests
```

After trying the correlations approach, I realized it took way too long to perform, since it was effectively correlating over 4 million pairs of scores even with a small sample. Also, as the sample size increases, the number of scores to consider is squared. This is because unlike the last situation where we were finding the correlations for one anime at a time, analyzing correlations for a whole genre requires taking a decently sized sample of anime from that genre, and correlating those scores with all other scores in that genre. With an average of about 110 scores per user, and thousands of shows in some genres, this method becomes unruly.

However, if we do the aggregation on the SQL side and manually calculate the confidence intervals of our correlation coefficients based on the standard deviation and sample sizes, we can test for the differences in the correlations. So let’s do this in python again with our custom aggregation function.

Hide

```
import random
import numpy as np
import math
genres = ["Romance","Fantasy","Action",";"]
results = dict()
sampleSize = 10
for genre in genres:
  tempQ = f"select id from anime_info where genres like '%{genre}%' and num_scoring_users
> 10000"
  possibleAnimes = [i[0] for i in con.execute(tempQ).fetchall()]
  testAnimes = random.sample(possibleAnimes,sampleSize)
  #print(possibleAnimes[:10])
  #print(str(testAnimes)[1:-1])
  query10 = f"""select corr(score1, score2) scores,anime_id2 from
    ((select anime_id as anime_id2,username as username2,score as score2  from user_record
s where anime_id in({str(testAnimes)[1:-1]}) limit 500000)
    inner join  (select anime_id,username,score as score1 from user_records where anime_id
in(select id from anime_info where num_scoring_users > 10000 and genres like '%{genre}%'))
    on  username = username2)  group by anime_id2"""
  result = con.execute(query10).fetchall()
  resultFormatted = [i[0] for i in result]
  std = np.std(resultFormatted)/math.sqrt(sampleSize)
  average = sum(resultFormatted)/sampleSize
  results[genre] = (average+2*std,average-2*std)
print(results)
```

```
{'Romance': (0.36942004310992915, 0.27821545486181376), 'Fantasy': (0.35418830020616515,
0.2951844831584805), 'Action': (0.34376148138587087, 0.28762007587502175), ';': (0.3169966
528692377, 0.2742584472768046)}
```

In the above code, I took 10 anime from each of the genres of interest, took a random set of up to 500000 ratings for those anime, and paired them with every other rating in that genre. Then I took the correlations between each of the 10 anime scores and scores for other shows from the same users. Then, based on the sample size of 10 shows, I made 95% confidence intervals for the correlations between all the shows in a genre for the entire dataset. I know this approach is partially flawed since I not only sampled shows, but I also sampled the 500,000 ratings for those shows when there could have been more. However, without this limit, the code would take hours to run because, again, it would process millions of rows. Also, I don;t think STAT355 taught us how to handle sampling within samples. For now, we assume the samples of 500,000 ratings are representative of all ratings for the 10 shows and we can move on. Anyways, as shown by the confidence intervals, each of our confidence intervals intersect that of the “;” genre, which really matches to any and every genre in the dataset. Whether or not the difference in the means is large enough to claim that people tend to think similarly about shows from the same genres is subjective. After all, running this same code multiple times can produce very different results. Still, **we fail to reject the null hypothesis. There is not enough evidence to show that people feel differently about anime based on their genre**