How Influential are Music Critics?*

GRADON NICHOLLS, Non-Degree Student, Joint Program in Survey Methodology, University of Maryland

Put abstract here

Keywords: Music Streaming, Experience Goods, R, APIs, Web Scraping

Introduction

When the quality of a product is unknown to the consumer prior to consumption, the consumer must rely on the experiences of others in order to make a purchase decision. These types of products are known as "experience goods" (Reinstein and Snyder, 2005), and include things like restaurants, movies, TV shows, tourist destinations, and music.

Before purchasing experience goods, consumers may rely both on expert reviews as well as reviews from other consumers or "users". Expert and user reviews are not perfectly correlated and may provide differing information: while expert reviews provide a sense of professionalism and objectivity, user reviews (though biased towards highly negative or highly positive scores) offer a relatable lay-person point of view (Amblee and Bui, 2007). Use of review scores can be thought of as a method for reducing search costs when researching a product (Huang, Lurie and Mitra, 2009) in which case consumers must come up with methods for integrating reviews from multiple sources (West and Broniarczyk, 1998), providing a demand for review aggregators such as Rotten Tomatoes and Metacritic.

The goal of this paper is to study the role music critics play with respect to the commercial performance of an artist's album. Anecdotally, it seems to be the case that critics have an impact on the music industry (Brennan, 2006). For instance, the mere fact that music critics exist at all, and have for so long, provides evidence that their work serves a need in the market. Music labels would seem to agree, as they often provide advance copies of their albums to reviewers so that reviews can coincide with the album's release date.

The music industry, and with it the role of music critics, has evolved as the economy becomes increasingly digitalized. Music can be consumed rapidly and at very low cost. Waldfogel (2015) provides a summary of the music industry in the age of "new media," and notes the ability of artists to find success despite a lack of airplay in traditional media. An example is The Arctic Monkeys, whose early success is attributed to the distribution of their demos by fans through the internet (Morey, 2008)–i.e. they are one of the first bands to "go viral."

In the age of new media, a music critic need not be a faceless author. At the time of knitting this document, the music review channel theneedledrop[1] has more than 2.5 million subscribers and nearly 800 million views across all videos. Anthony Fantano, the curator of the channel, is not just a reviewer but an internet personality with a dedicated following, and has a side channel on YouTube for more "off-the-cuff" content. Thus, I argue that Fantano may exist as a cross between an "expert" and "user" reviewer–in other words, he builds a lay-person trust through his vlog-like content, while maintaining credibility as an objective and thorough reviewer through his main channel.

In the new media landscape, we can hypothesize a clear causal pathway between critic reviews and album performance that did not exist in a pre-digital world. For example, in the description of a YouTube video or in the text of an article, a critic can include a link which sends the viewer or reader directly to listen to the

*This paper is prepared for the course SURV727 taught by Ruben Bach and Christoph Kern jointly at U Michigan and U Maryland. The project files can be found on my github. I use the Article2 template developed by Steve Miller to create this paper.

[1] https://www.youtube.com/channel/UCt7fwAhXDy3oNFTAzF2o8Pw

album. Streaming platforms like Spotify or Apple Music even allow you to embed their players in a webpage for even more immediate listening. Artists have often complained of low pay-out per stream on such platforms.[2] However, as of 2016, streaming accounted for a majority of revenue in the industry, and growth in total revenue was driven mostly by paid subscriptions to streaming services.[3] This, combined with the fact that streaming could also lead to fans buying physical albums, merchandise, or concert tickets, make streaming a relevant variable of interest, and it is therefore the focus of this paper.

In order to correlate review scores with number of listens, I make use of several data sources. I start with a list of top 1000 artists on Spotify, and obtain information on their albums and release dates from the Spotify api. For each album, I pull its respective article from the Wikipedia api, and use web scraping techniques to obtain review scores from the page (if they exist). Unfortunately, the Spotify api does not provide the number of listens for an album. I therefore use the Last.fm api to collect data on the number of "scrobbles", which is a term for music listening data that Last.fm has collected from other apps that a user is running.

In estimation, we should consider that a critic can take the role of either an *influencer* or a *predictor* (Eliashberg and Shugan, 1997). As an *influencer*, a positive review could entice their audience to listen to an album. Additionally, the review may act simply as a source of publicity (Sorensen and Rasmussen, 2004), regardless of whether an album is scored positively or negatively. As a *predictor*, the reviewer may simply be forecasting the popularity of an album. As discussed in Reinstein and Snyder (2005), this latter effect is a spurious correlation, and does not represent the causal effect that we are interested in.

I attempt to control for the predictor effect in two ways. First, I assume that correlation between review scores and listens for older albums represent the predictor effect only–that is, any effect critics have had has dissipated over time. Under this reasoning, by including a time dummy interacted with review scores, I should capture the influencer effect at the margin. Second, I make use of artist fixed effects to account for unobserved artist-specific variables. This should control for predictor effects in certain cases–for example, if reviewers tend to give good reviews to already-popular artists, or if there are demographic effects.

Data

I make use of several data sources in order to collect data on artists, their albums, the number of listens for each album, and the review scores of each album. As a high-level summary, the procedure is as follows:

- Start with a list of artist names
- For each artist name, use Spotify api to to find their albums
- For each album, use Last.fm api to find the album's scrobbles
- For each album, use the Google Custom Search api to find the url of the album's wikipedia page
- Use web scraping techniques to obtain review scores from the wikipedia page

The following subsections provide details and code for each task. The code to produce the full dataset takes a long time to run, so for brevity I show an example of one artist and one album here. The full code can be found in CreateData.R in the GitHub repository for this project.

Note that we make use of the following packages:

```
if (!require("pacman")) install.packages("pacman")
library(pacman)
p_load(char=c("bookdown","curl","httr","jsonlite","rvest","dplyr"
```

[2] e.g. https://www.nytimes.com/2021/05/07/arts/music/streaming-music-payments.html
[3] http://www.riaa.com/wp-content/uploads/2017/03/RIAA-2016-Year-End-News-Notes.pdf

```
              ,"papeR","vtable","stargazer","sandwich","stevetemplates"
             ,"spotifyr","stringr","tm","utils","WikipediR","ggplot2","plm"))
```

List of artist names

Given the uncountable number of artists that have ever existed, it is not immediately clear how best to choose which artists should be included in the analysis. For now, I prioritize convenience, and use a list of the top 1000 Spotify artists as pulled by chartmasters.org. It seems reasonable to start with the most popular artists given that they make up the largest share of the market. However, it could bias results if, for example, reviews have a greater impact on lesser-known artists. To get a sense of the popularity of the artists, the number 1 and number 1000 artists currently have about 55 and 7 million monthly listeners, respectively, at the time of writing.

Below is code[4] for scraping the list from chartmaster's webpage. Note that the list is updated regularly, and so may differ slightly from my list (due to a glitch, I collected the data in two batches, once on November 1 2021 and one on December 15, 2021). Currently, the top 4 artists on Spotify are Drake, Ed Sheeran, Bad Bunny, and Ariana Grande.

```
chartmasterurl = "https://chartmasters.org/most-streamed-artists-ever-on-spotify/"
chartwebpage = read_html(chartmasterurl)
xpath='//td[(((count(preceding-sibling::*) + 1) = 2) and parent::*)]'
artistnames = html_nodes(chartwebpage, xpath=xpath)
artistnames = html_text(artistnames,trim=T)
head(artistnames,4)
```

```
## [1] "Drake"         "Ed Sheeran"    "Bad Bunny"     "Ariana Grande"
```

Spotify API

To obtain album information for each artist, I make use of the Spotify API. First, I search for the artist's name on Spotify and take the first result. As long as the list of artists we start with is relatively close to their official names, we should succeed in finding the correct listing in Spotify. It further helps that in this case, the list I start with is itself pulled from Spotify by chartmasters.org. Below, I obtain the id number and artist name (as defined by Spotify) for Drake. In this case, the artist's name is the same as what we started with.

```
search = spotifyr::search_spotify(q="Drake", type="artist")
artist_id = search$id[1]
artist = spotifyr::get_artist(id=artist_id, authorization=access_token)
artist_name = artist$name
artist_name
```

```
## [1] "Drake"
```

Using the artist id number, I obtain a list of Drake's albums in Spotify. Note that albums can be pulled only 50 at a time. This is accounted for in the main code by pulling albums 50 at a time until all albums are found.

[4] I make use of the selector gadget to help with web scraping.

```
albums = spotifyr::get_artist_albums(id=artist_id,include_groups="album"
                                     ,market="CA",limit=50
                                     ,authorization=access_token)
albums %>%
  select(c(name,release_date,total_tracks)) %>%
  slice(16:24)
```

```
##                                   name release_date total_tracks
## 1 If You're Reading This It's Too Late   2015-02-12           17
## 2 If You're Reading This It's Too Late   2015-02-12           17
## 3        Nothing Was The Same (Deluxe)   2013-01-01           16
## 4        Nothing Was The Same (Deluxe)   2013-01-01           15
## 5        Nothing Was The Same (Deluxe)   2013-01-01           16
## 6        Nothing Was The Same (Deluxe)   2013-01-01           16
## 7        Nothing Was The Same (Deluxe)   2013-01-01           15
## 8                 Nothing Was The Same   2013-01-01           13
## 9                 Nothing Was The Same   2013-01-01           13
```

You will note that multiple versions of the same album exist in Spotify. Sometimes they are identical, though other times they may have deluxe versions, differing numbers of tracks, or differing release dates. Since we do not pull listens from Spotify itself, we can remove duplicates from the list without having to worry about aggregating albums together. To remove duplicates, I first remove text contained within round or square brackets. Next, I remove any albums that contain duplicate names or release dates. In this case we are left with 12 distinct albums released by Drake, as seen below:

```
albums$name = gsub("\\s*\\([^\\)]+\\)","",albums$name)
albums = albums[!duplicated(albums$name),]
albums = albums[!duplicated(albums$release_date),]

albums %>%
  select(c(name,release_date,total_tracks))
```

```
##                                   name release_date total_tracks
## 1                   Certified Lover Boy   2021-09-03           21
## 3                  Dark Lane Demo Tapes   2020-05-01           14
## 5                          Care Package   2019-08-02           17
## 7                           So Far Gone   2019-02-14           18
## 8                              Scorpion   2018-06-29           25
## 10                           More Life   2017-03-18           22
## 12                               Views   2016-05-06           20
## 14              What A Time To Be Alive   2015-09-25           11
## 16 If You're Reading This It's Too Late   2015-02-12           17
## 18                 Nothing Was The Same   2013-01-01           16
## 25                           Take Care   2011-11-15           19
## 31                       Thank Me Later   2010-01-01           15
```

We can obtain information about an album's tracks by using the album's id number provided by the API. Due to time constraints, I have not yet experimented with using track analysis in this paper, and so all we need for now is the album's name. For the remainder of this section we will use Drake's most recent album, Certified Lover Boy, as the example.

```
album_info=spotifyr::get_album(id=albums$id[1]
                                    ,authorization=access_token)
album_name = album_info$name
album_name
```

```
## [1] "Certified Lover Boy"
```

Last.fm

As previously noted, Spotify does not provide access to listening statistics through their API. I therefore turn to Last.fm to collect a proxy measure. As of 2014, Last.fm does not offer its own streaming service, but instead provides integration with Spotify and YouTube.[5] Last.fm's main service is to allow users to track their music listening behaviour across multiple devices and apps. To do this, it collects information from other apps in the background, a process called "scrobbling."

The benefit of using Last.fm scrobbles in our analysis is that we obtain data on listening behaviour from many different sources (for example, both Spotify and Apple Music would be captured). The major downside is that Last.fm is an opt-in service. This means there is a potential for selection bias in our analysis. I hypothesize that Last.fm users represent more dedicated music listeners, and thus may be more aware of music news and reviews. If that is the case, correlations between listening and review scores may be biased upwards, though are still informative about a large subgroup of listeners.

That aside, obtaining scrobbles is fairly straightforward thanks to Last.fm's API. Since Last.FM's data is scrobbled from other sources, the official artist and album names from Spotify can be used to construct a valid url in most cases, as long as we take care to strip out characters that have special meanings.

```
url_artist = str_replace_all(artist_name,"\\+","%2B")
url_album  = str_replace_all(album_name,"\\+","%2B")
url_artist = str_replace_all(url_artist,"&","%26")
url_album  = str_replace_all(url_album,"&","%26")
url_artist = str_replace_all(url_artist,"#","%23")
url_album  = str_replace_all(url_album,"#","%23")

url = paste0("http://ws.audioscrobbler.com/2.0/",
             "?method=album.getinfo",
             "&api_key=","xxxxxxxx",
             "&artist=", gsub(" ", "+", url_artist),
             "&album=", gsub(" ", "+", url_album),
             "&format=json")
url
```

```
## [1] "http://ws.audioscrobbler.com/2.0/?method=album.getinfo&api_key=xxxxxxxx&artist=Drake&al
```

[5] https://blog.last.fm/2014/01/29/did-someone-say-on-demand

Below we parse the JSON file in order to obtain the number of scrobbles. A quick check on Last.FM shows that the number we pulled is the correct one.

```
data_json = httr::GET(url)
data_json = jsonlite::fromJSON(rawToChar(data_json$content))
album_scrobbles = as.integer(data_json$album$playcount)
album_scrobbles
```

```
## [1] 17603629
```

Wikipedia

Metacritic is an obvious source for aggregated album review scores. However, web scraping at a mass scale is not permitted on the platform,[6] and they do not offer an API service. I therefore turn to Wikipedia for review score information.

Unfortunately, Wikipedia is not as exhaustive as Metacritic. In our example of Drake's Certified Lover Boy, the Metacritic page shows 20 reviews while the corresponding wiki page shows only 10. As will be seen later, most Wikipedia pages I collected contained 10 or fewer reviews, suggesting that 10 may be the upper bound chosen by Wikipedia moderators. It is worth further investigation as to how reviews are chosen to be included. Interestingly, neither Wikipedia nor Metacritic host scores from YouTube channel theneedledrop, meaning that results may reflect old media more than new media.

Rather than try to construct the correct Wikipedia url by hand, I use the Google Custom Search API, programmed to query only wikipedia.org, to search the phrase "[artist name] [album name] album". The output is a list of the 10 most relevant web pages as selected by Google. I try the first result, and if that fails, assume review scores and/or the desired article do not exist. Anecdotally this has worked well, though more work should be done to test the accuracy of the algorithm.

```
keyword = paste0(artist_name," ",album_name," ","album")
keyword = gsub("[[:punct:]]", " ", keyword)
keyword = str_squish(keyword)
keyword = gsub(" ", "+", keyword)

url = paste0("https://www.googleapis.com/customsearch/v1?"
             , "key=", google.key
             , "&q=", keyword
             , "&gl=us"
             , "&hl=en"
             , "&cx=", google.cx
             , "&fields=items(link)"
)
googlesearch = GET(url)
content = rawToChar(googlesearch$content)
searchresults = fromJSON(content)
searchresults
```

[6] That being said, I place code for scraping individual Metacritic pages in the Appendix I.

```
## $items
##                                                                                 link
## 1                                  https://en.wikipedia.org/wiki/Certified_Lover_Boy
## 2                                  https://de.wikipedia.org/wiki/Certified_Lover_Boy
## 3                                     https://en.wikipedia.org/wiki/Drake_(musician)
## 4                                  https://fr.wikipedia.org/wiki/Certified_Lover_Boy
## 5                                  https://es.wikipedia.org/wiki/Certified_Lover_Boy
## 6                                    https://en.wikipedia.org/wiki/Drake_discography
## 7  https://en.wikipedia.org/wiki/List_of_Billboard_200_number-one_albums_of_2021
## 8                                    https://en.wikipedia.org/wiki/Girls_Want_Girls
## 9                 https://en.wikipedia.org/wiki/Solid_(Young_Thug_and_Gunna_song)
## 10                                       https://en.wikipedia.org/wiki/In_the_Bible
```

Rather than scrape the Wikipedia page, I make use of the Wikipedia API to retrieve the page's content.

```
wikiurl = searchresults$items$link[1]

# deal with special characters
pgname = curl_unescape(wikiurl)

# keep the end of the url -> the page name wikipedia api wants
pgname = gsub("https://.*\\.wikipedia\\.org.*/","",pgname)

# obtain page content
webpage = WikipediR::page_content(language="en",project="wikipedia",page_name=pgname)
```

From there, there are quite a number of processing steps required in order to properly format review scores. The complete code can be found in Appendix II. In short, we must convert scores from various units to a consistent scale (e.g. 3/5 stars, 8/10 stars, 60%, 50/100). In the case of letter grades (e.g. B+), I use Metaritic's rules for converting to a numeric grade[7]. Reviews based solely on words (e.g. "positive") are excluded from analysis. I also pull release date information as a way to cross-reference with Spotify.

The end result can be seen below:

```
release_date_wiki
```

```
## [1] "2021-09-03"
```

```
wikiscores_data
```

```
##                      V1 V2
## 3              AllMusic 60
## 4         The Arts Desk 60
## 5                 Clash 60
## 6  Entertainment Weekly 50
```

[7] https://www.metacritic.com/about-metascores

```
## 7           The Guardian 60
## 8       The Independent 40
## 9  The Line of Best Fit 50
## 10                  NME 40
## 11            Pitchfork 66
## 12        Rolling Stone 70
```

Exploratory Data Analysis and Cleaning

The result of the previous steps is a data frame with 12,694 albums.[8] Here I load the data. As previously mentioned, I pulled data in two batches, and so append the two batches together, creating a flag for each batch so I can account for the time discrepancy in estimation.

```
load(file="finaldata.RData")
load(file="finaldata_pt2.RData")

finaldata = finaldata %>%
  mutate(batch=1) %>%
  dplyr::select(-isbad)
finaldata_pt2 = finaldata_pt2 %>% mutate(batch=2)
finaldata = rbind(finaldata,finaldata_pt2)
```

We need a method of aggregating review scores in order to conduct analysis. Here I consider the number of available scores and the mean score, but one could consider other measures like order statistics or measures of spread.

```
finaldata$numscores =
  unlist(lapply(finaldata$wiki_scores, FUN = function(y) {length(y)}))
finaldata$meanscores =
  unlist(lapply(finaldata$wiki_scores, FUN = function(y) {mean(y,na.rm=T)}))
```

First, let's look at the distribution of number of scores by album:

[8] For the full code, see Appendix III.

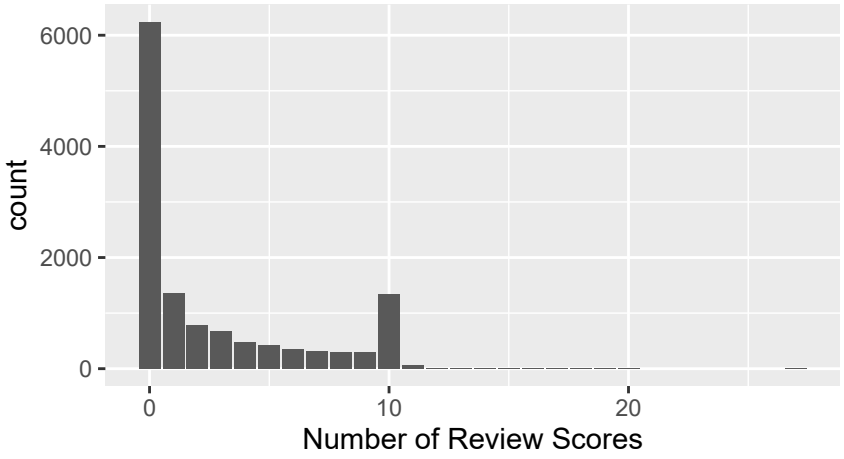

Figure 1: Distribution of Number of Review Scores

We note first that there is a suspicious cluster at 10 reviews. As noted above, Wikipedia pages are not ehaustive in their listing of review scores, and it seems that 10 reviews is the usual maximum provided on Wikipedia pages. Thus, there is right-tail censoring in this variable, and this should be better understood in future.

Additionally, nearly half of the albums pulled have 0 review scores. Given 12,694 albums and 1,000 artists, we have on average about 12 albums per artist. This is somewhat high, and suggests there may be the presence of live albums, EPs, etc that are lesser known and less reviewed. I also note the presence of classical musicians and music associated with children's cartoons, that are popular but not in scope of the study. For now, I drop all 0's, under the assumption that popular artists in scope for analysis would likely have all of their major albums reviewed.

```
cleandata = finaldata %>%
  filter(numscores>0)
```

Next we look at the number of albums by artist:

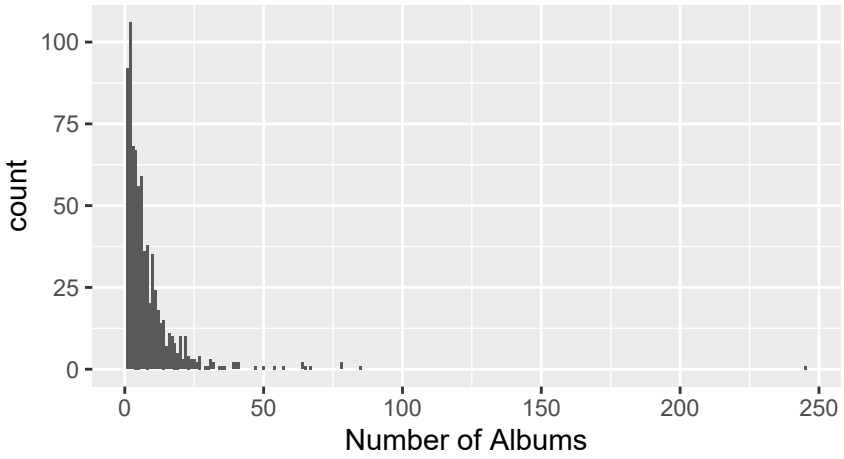

Figure 2: Distribution of Number of Albums

We note first that there are several extreme outliers who have up to nearly 250 albums. Let's take a look at a few of the most extreme outliers:

Table 1: Artists with More than 50 Albums

Artist Name	Num. Albums
Elvis Presley	54
Frank Sinatra	64
Johnny Cash	78
Bob Dylan	57
Frédéric Chopin	245
Neil Young	67
Dolly Parton	65
Willie Nelson	85
Ella Fitzgerald	78
Grateful Dead	64

Frederic Chopin, a classical composer, is a clear outlier and not in scope. I therefore drop from subsequent analysis. Other outliers do appear however to be in scope. The next biggest outlier is Willie Nelson with 85 albums. Let's take a look at the distribution of the number of scrobbles of Willie's albums:

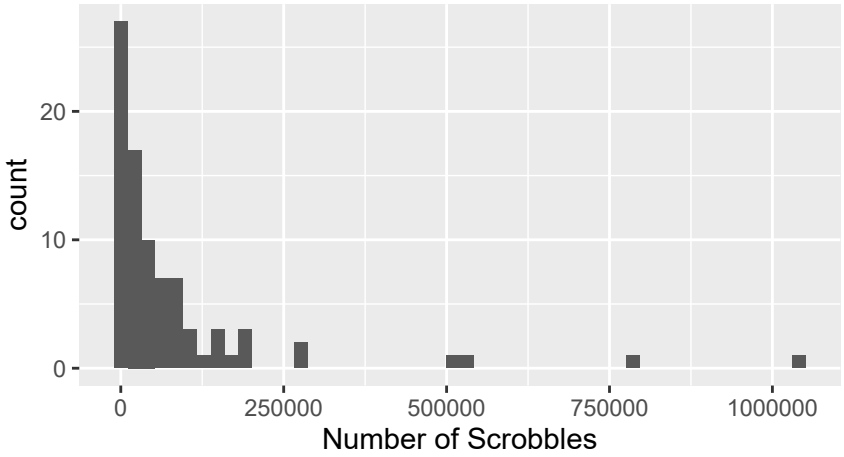

Figure 3: Distribution of Scrobbles Among Willie Nelson Albums

Willie Nelson appears to have quite a few albums with very few scrobbles. These likely represent more niche releases such as live albums or demos. While not necessarily out of scope, it would be worth doing a robustness check in the analysis stage where these are dropped. For now I use a simple method: classify the smallest 2.5% of an artist's albums as "outliers".

```
cleandata = cleandata %>%
  group_by(artist_id) %>%
  mutate(lowerbound=quantile(album_scrobbles,p=0.025,na.rm=T)) %>%
  ungroup() %>%
```

```
  mutate(outlier=album_scrobbles<lowerbound)
```

We can see that in the case of Willie Nelson, we would drop three albums with only a handful of scrobbles.

```
cleandata %>%
  filter(artist_name=="Willie Nelson") %>%
  dplyr::select(c(artist_name,album_name,album_scrobbles,outlier)) %>%
  filter(outlier==TRUE) %>%
  kable(caption="Outlier Example: Willie Nelson"
        ,col.names=c("Artist","Album","Scrobbles","Outlier?")) %>%
  kable_styling(latex_options = "HOLD_position")
```

Table 2: Outlier Example: Willie Nelson

Artist	Album	Scrobbles	Outlier?
Willie Nelson	If I Can Find a Clean Shirt	93	TRUE
Willie Nelson	The Winning Hand	270	TRUE
Willie Nelson	WWII	149	TRUE

Finally, I drop a few albums with missing or 0 scrobbles.

```
cleandata = cleandata %>%
  filter(!is.na(album_scrobbles)) %>%
  filter(album_scrobbles>0)
```

The resulting dataset has 755 artists and 6193 albums. Below are summary tables of artist-level and album-level statistics.

Table 3: Artist-level Summary Statistics

Variable	N	Mean	Std. Dev.	Min	Pctl. 25	Pctl. 75	Max
Artist Scrobbles	755	25642436.976	46461607.397	13	2743014.5	27194475.5	556318942
Num. Albums	755	8.203	9.879	1	2	10	85
Num. Followers	755	6158321.025	8536062.242	3790	2044827.5	6485957.5	88706690

Table 4: Album-level Summary Statistics

Variable	N	Mean	Std. Dev.	Min	Pctl. 25	Pctl. 75	Max
Album Scrobbles							
Num. Review Scores	6193	5.347	3.566	1	2	9	27
Avg. Review Scores	6118	69.705	13.067	13.333	60	79.5	100

Model

Model 1: Linear model with fixed effects

Let s_{ijt} denote the number of scrobbles that artist i has gotten on album j after t days have occured–i.e. scrobbles are cumulative over time. Then we model scrobbles as follows:

$$s_{ijt} = \alpha_i + \beta_t t + \beta_r r_{ij} + \epsilon_{ijt}$$

where $\alpha_0, \alpha_i, \beta_t, \beta_r$ are parameters, α_i is an artist-specific fixed effect, r_{ij} is an aggregated measure of review scores, and ϵ_{ijt} is a random innovation. The main advantage of this model is the inclusion of artist fixed effects, which account for unobserved artist-specific factors–for example, and artists inherent popularity.

Model 2: Log-transformed

It is reasonable to think that the effect of review scores on scrobbles could be based on percentage changes rather than linear changes. In such a case, log transforming the key variables of interest may produce more realistic results:

$$\log s_{ijt} = \alpha_i + \beta_t \log t + \beta_r \log r_{ij} + \epsilon_{ijt}$$

Model 3: Legacy artists

As discussed in the introduction, one method I can use to account for endogeneity is to assume that correlations between album performance and review scores for legacy albums–that is, albums released prior to the existence of streaming platforms–is spurious. In other words, reviews of the album predict audience reception rather than influence audience behaviour. In this way, differences between legacy albums and newer albums should reflect the causal portion of β_r, the parameter of interest.

This is a diff-in-diff approach, and is modelled as follows. Let $L_i = 1$ if artist i is a legacy artist and 0 otherwise. Then,

$$\log s_{ijt} = \alpha_i + \beta_t \log t + \beta_0 L_i + \beta_r (1 - L_i) \log r_{ij} + \epsilon_{ijt}$$

As for how to define legacy albums, for now I use the cutoff of albums released prior to 2010, as Spotify grew its user base in the early 2010's.

Note on Standard Errors

Note that it is likely that our data is clustered–that is, scrobbles from different albums of the same artist are correlated. While OLS estimates are still unbiased under this assumption, standard errors will be too small. Thus, I use cluster-robust standard errors in all analysis.

Results

Estimation

Before estimation, I define a few variables:

```
# num days since album release
estdata = cleandata %>%
  mutate(releasedate = as.Date(album_releasedate)) %>%
  mutate(currdate = ifelse(batch==1,"2021-12-05","2021-12-15")) %>%
  mutate(currdate = as.Date(currdate)) %>%
  mutate(numdays = as.numeric(currdate - releasedate))

# classify legacy album
estdata = estdata %>%
  mutate(legacy = as.numeric(releasedate<as.Date("2010-01-01")))
```

Here I estimate each of the three models:

```
model1 = plm(album_scrobbles ~ numdays + meanscores
          ,data=estdata
          ,index="artist_id"
          ,model="within")
cov1       = vcovHC(model1,method="arellano",type="HC1",cluster="group")
robust.se1 = sqrt(diag(cov1))

model2 = plm(log(album_scrobbles) ~ log(numdays) + log(meanscores)
          ,data=estdata
          ,index="artist_id"
          ,model="within")
cov2       = vcovHC(model2,method="arellano",type="HC1",cluster="group")
robust.se2 = sqrt(diag(cov2))

estdata = estdata %>% mutate(logscores_diffindiff = (1-legacy)*log(meanscores))
model3 = plm(log(album_scrobbles) ~ log(numdays) + legacy + logscores_diffindiff
          ,data=estdata
          ,index="artist_id"
          ,model="within")
cov3       = vcovHC(model3,method="arellano",type="HC1",cluster="group")
robust.se3 = sqrt(diag(cov3))
```

```
stargazer(model1,model2,model3
          ,se=list(robust.se1,robust.se2,robust.se3)
          ,title="Regression Results")
```

% Table created by stargazer v.5.2.2 by Marek Hlavac, Harvard University. E-mail: hlavac at fas.harvard.edu
% Date and time: Fri, Dec 17, 2021 - 12:11:42 AM

Interpretation of Results

Table 5 presents results from each of the three models. I note first that in all models, average review scores are statistically significant predictors of number of scrobbles at 1%. To interpet the model, let's use an example of

Table 5: Regression Results

	Dependent variable:		
	album_scrobbles	log(album_scrobbles)	
	(1)	(2)	(3)
numdays	142.101***		
	(21.233)		
meanscores	59,910.570***		
	(9,031.494)		
log(numdays)		0.493***	0.447***
		(0.031)	(0.035)
log(meanscores)		1.939***	
		(0.150)	
legacy			7.417***
			(1.346)
logscores_diffindiff			1.711***
			(0.319)
Observations	5,688	5,688	5,688
R^2	0.021	0.105	0.078
Adjusted R^2	−0.127	−0.031	−0.062
F Statistic	54.181*** (df = 2; 4938)	289.532*** (df = 2; 4938)	139.580*** (df = 3; 4937)
Note:			*p<0.1; **p<0.05; ***p<0.01

an album that has 100,000 scrobbles and an average review score of 50. If they increased their review score to an 80, model 1 predicts they would gain about 1.8 million scrobbles. Model 2 predicts they would gain about 250,000, and Model 3 predicts they would gain about 220,000.

Thus, we see first that the log-transformed version provides more realistic predictions as expected. This is also reflected in better model fit statistcs. Furthermore, we observe that the diff-in-diff version predicts a smaller gain, though not much different from the model without this component.

Conclusions / Future Work

At the onset of this project, I started with three primary research questions:

- Do music critics affect album performance?
- Differences between user and expert scores?
- Difference in new and old media?

This paper does some investigation into the first question, although time and data constraints preclude me from looking at the second and third. After accounting for artist fixed effects, and using a diff-in-diff approach to account for spurious correlations, we find that review scores do seem to have an effect on album performance, at least based on our current data and assumptions.

Future work should focus on performing robustness checks on my current analysis. Furthermore, additional controls could be included in the model. Easy choices using similar data approaches would be track information (e.g. perhaps "happier" albums get more listens AND better reviews) from the Spotify API and demographic information about the artists from their Wikipedia pages.

Additional work could be directed at fine tuning my data algorithm. While I know by example that it seems to work in many cases, it would be great to do a systematic analysis to determine how effective my method is. For example, what is the nature of the nearly half of albums that had no reviews?

Further, I would reconsider how to overcome data limitations with respect to listening numbers (Last.FM has potential selection issues as discussed) and user scores (to analyze user vs expert score effects). The former could be purchased from chartmasters.org, while the latter theoretically exists on metacritic–perhaps one day they will allow access to the data via API.

Finally, in the long term it would be interesting to merge this data with review scores from theneedledrop's YouTube channel to study old vs new media effects, particularly by genre. theneedledrop puts his review scores in the description of his videos, allowing one in theory to access them via YouTube's API.

Appendix I: Web Scraping Critic and User Scores

Here is code to scrape critic and user scores from Metacritic, using Drake's Certified Lover Boy as an example. I include it here to document my work, but do NOT suggest using it on a mass scale as web scraping is prohibited by their terms of service.

```
# get album name in right format
album_name_meta = tolower(tm::removePunctuation(album_name))
album_name_meta = album_name_meta %>%
  str_replace_all("&", "") %>%
  tolower() %>%
  str_squish() %>%
  str_replace_all(" ", "-")

# get artists name in right format
arist_name_meta = tolower(removePunctuation(artist_name))
artist_name_meta = arist_name_meta %>%
  str_replace_all("&", "") %>%
  tolower() %>%
  str_squish() %>%
  str_replace_all(" ", "-")

# critic scores
# read in html
url_critic = paste0('https://www.metacritic.com/music/'
              ,album_name_meta,'/'
              ,artist_name_meta
              ,'/critic-reviews')
url_user   = paste0('https://www.metacritic.com/music/'
              ,album_name_meta,'/'
              ,artist_name_meta
              ,'/user-reviews')

if (!http_error(url_critic)) {

  webpage = read_html(url_critic)

  # critic scores
  xpath = '//*[(@id = "main")]//*[contains(concat( " ", @class, " " ), concat( " ", "indiv", "
  criticscores = html_nodes(webpage, xpath=xpath)
  criticscores = html_text(criticscores,trim=T)
  criticscores = as.numeric(criticscores)

  # user scores
  # read the aggregated score
  webpage = read_html(url_user)
```

```
  xpath = '//*[contains(concat( " ", @class, " " ), concat( " ", "large", " " ))]'
  userscore_agg = html_nodes(webpage, xpath=xpath)
  userscore_agg = html_text(userscore_agg,trim=T)
  userscore_agg = as.numeric(userscore_agg[1])

  # read number positives, neutral, negative
  xpath='//*[(@id = "main")]//*[contains(concat( " ", @class, " " ), concat( " ", "count", " "
  userscore_meta = html_nodes(webpage, xpath=xpath)
  userscore_meta = html_text(userscore_meta,trim=T)
  userscore_meta = as.numeric(userscore_meta[2:4])
}
```

```
criticscores
```

```
##  [1] 80 72 70 68 67 67 66 60 60 60 60 60 50 50 45 40 40 40 40 40
```

```
userscore_agg
```

```
## [1] 3.5
```

```
userscore_meta
```

```
## [1] 237 186 655
```

Appendix II: Scraping Wikipedia Articles for Critic Scores

```
# Function for converting letter grades to number grades
lettergrade_num = function(x) {
  if (x=="A+" | x=="A") {
    return("100")
  } else if (x=="A-") {
    return("91")
  } else if (x=="B+") {
    return("83")
  } else if (x=="B") {
    return("75")
  } else if (x=="B-") {
    return("67")
  } else if (x=="C+") {
    return("58")
  } else if (x=="C") {
    return("50")
  } else if (x=="C-") {
    return("42")
  } else if (x=="D+") {
    return("33")
  } else if (x=="D") {
    return("25")
  } else if (x=="D-") {
    return("16")
  } else if (x=="F+") {
    return("8")
  } else if (x=="F" | x=="F-") {
    return("0")
  } else {
    return(x)
  }
}

# get web page content
webpage = read_html(webpage$parse$text$`*`)
xpath='//*[contains(concat( " ", @class, " " ), concat( " ", "floatright", " " ))]//td'
wikiscores = html_nodes(webpage, xpath=xpath)
wikiscores_stars = html_attr(html_element(wikiscores,css='span'),'title')
wikiscores = html_text(wikiscores,trim=T)

  # Convert scores to numeric out of 100
  wikiscores_data = as.data.frame(matrix(wikiscores,ncol=2,byrow=T))
  wikiscores_stars_data = as.data.frame(matrix(wikiscores_stars,ncol=2,byrow=T))
```

```r
# deal with specific review websites
selector = (!is.na(wikiscores_stars_data$V2) & !(tolower(wikiscores_data$V1)=="tom hull - on
wikiscores_data$V2[selector] = wikiscores_stars_data$V2[selector]
wikiscores_data = wikiscores_data[wikiscores_data$V1!="Metacritic",]
wikiscores_data = wikiscores_data[wikiscores_data$V1!="AnyDecentMusic?",]
# remove [] and content inside
wikiscores_data$V2 = gsub("\\[.*?\\]","",wikiscores_data$V2)
# remove () but NOT CONTENT (sometimes the score is contained within () )
wikiscores_data$V2 = str_replace(wikiscores_data$V2,"\\(","")
wikiscores_data$V2 = str_replace(wikiscores_data$V2,"\\)","")
# fix ill-behaved special characters
wikiscores_data$V2 = sapply(wikiscores_data$V2 , function(x) URLencode(x) )
wikiscores_data$V2 = str_replace(wikiscores_data$V2,"%E2%88%92","-")
wikiscores_data$V2 = str_replace(wikiscores_data$V2,"%E2%80%93","-")
wikiscores_data$V2 = str_replace(wikiscores_data$V2,"%E2%80%94","-")
wikiscores_data$V2 = sapply(wikiscores_data$V2 , function(x) URLdecode(x) )
wikiscores_data$V2 = str_replace(wikiscores_data$V2,"\\?\\^\\'","-")
# covert letter grade to numeric
wikiscores_data$V2 = sapply(str_trim(wikiscores_data$V2) , lettergrade_num)
# remove unit of measure (stars or discs)
wikiscores_data$V2 = stringr::str_replace(wikiscores_data$V2,"stars","")
wikiscores_data$V2 = stringr::str_replace(wikiscores_data$V2,"discs","")
# remove %'s
wikiscores_data$V2 = stringr::str_replace(wikiscores_data$V2,"%","")
# remove remaining scores that contain no numbers
wikiscores_data = wikiscores_data[grepl("\\d",wikiscores_data$V2),]

# convert to numeric
wikiscores_data$V2 = sapply(wikiscores_data$V2,
                            function(x) tryCatch(eval(parse(text=x)),
                                                 error=function(e){},
                                                 finally=NULL) )

# final set of scores to put in dataset
wikiscores_data = wikiscores_data[!sapply(wikiscores_data$V2,is.null),]
wikiscores_data$V2 = as.numeric(wikiscores_data$V2)
wikiscores_data$V2[wikiscores_data$V2<=1] = 100*wikiscores_data$V2[wikiscores_data$V2<=1]

xpath = '//*[contains(concat( " ", @class, " " ), concat( " ", "published", " " ))]'
release_date_wiki = html_nodes(webpage, xpath=xpath)
release_date_wiki = html_text(release_date_wiki,trim=T)
release_date_wiki = release_date_wiki[2]
```

Appendix III: Full Code for Generating Data

```
# load packages and define helper functions
source("Header.R")

# get list of artist names
load("artistnamelist.RData") # loads object "artistnames"

# For each name in the list, create data of albums and reviews
artistdata = NULL
for (aname in artistnames[694:1000]) {

## SPOTIFY DATA #####################################

  id = '3f766570eef74a6986a646bf7dd73102'
  secret = '5f884eb17551445ba76ab797ec7505b6'
  Sys.setenv(SPOTIFY_CLIENT_ID = id)
  Sys.setenv(SPOTIFY_CLIENT_SECRET = secret)
  access_token = spotifyr::get_spotify_access_token()

  # artist info
  search = spotifyr::search_spotify(q=aname, type="artist")
  artist_id = search$id[1]
  artist = spotifyr::get_artist(id=artist_id, authorization=access_token)
  artist_name = artist$name

  # album list
  albums = spotifyr::get_artist_albums(id=artist_id,include_groups="album",market="CA",limit=5
  nalb = length(albums)
  offset = 1
  while(nalb>0) {
      offset = offset + 50
      newalbs = spotifyr::get_artist_albums(id=artist_id,include_groups="album",market="CA",li
      nalb = length(newalbs)
      albums = rbind(albums,newalbs)
  }


  # Dedupe album list
  albums$name = gsub("\\s*\\([^\\)]+\\)","",albums$name) # remove "deluxe" versions, etc.
  if(length(albums$name)>0) { # only proceed if they have albums!
    albums = albums[!duplicated(albums$name),]
    albums = albums[!duplicated(albums$release_date),]
  } else {
    print(paste0("Skipping ", artist_name, "----no albums!"))
```

```
}

# loop through albums
album_ids = albums$id
for (id in album_ids) {
  example_album=spotifyr::get_album(id=id,authorization=access_token)

  ## LAST FM ###################################################################
  api_key = "749b4800592668088e8a563845cf906f"
  # '&' and '+' and '#' cause issues
  url_artist = stringr::str_replace_all(artist_name,"\\+","%2B")
  url_album = stringr::str_replace_all(example_album$name,"\\+","%2B")
  url_artist = stringr::str_replace_all(url_artist,"&","%26")
  url_album = stringr::str_replace_all(url_album,"&","%26")
  url_artist = stringr::str_replace_all(url_artist,"#","%23")
  url_album = stringr::str_replace_all(url_album,"#","%23")

  url = paste0("http://ws.audioscrobbler.com/2.0/",
               "?method=album.getinfo",
               "&api_key=",api_key,
               "&artist=", gsub(" ", "+", url_artist),
               "&album=", gsub(" ", "+", url_album),
               "&format=json")
  data_json = httr::GET(url)
  data_json = jsonlite::fromJSON(rawToChar(data_json$content))
  album_scrobbles = as.integer(data_json$album$playcount)
  if (length(album_scrobbles)==0) { # handle missing scrobbles
    album_scrobbles = NA
  }
  ############################################################################

  ## CRITIC SCORES FROM WIKIPEDIA ##############################################

  # Use google api to search for the (most likely) correct wikipedia page
  # (note that this is a custom search api that searches only wikipedia)
   google.key = 'AIzaSyBKJ9b9c1zKEEUs-g2PTrosIHO8epctAls'
   google.cx = '2fb4b305ff06c300d'

   # keywords to be searched in google, separated by '+'
   keyword = paste0(artist_name," ",example_album$name," ","album")
   keyword = gsub("[[:punct:]]", " ", keyword)
   keyword = stringr::str_squish(keyword)
   keyword = gsub(" ", "+", keyword)

   # api call
```

```
    url = paste0("https://www.googleapis.com/customsearch/v1?"
                 , "key=", google.key
                 , "&q=", keyword
                 , "&gl=us"
                 , "&hl=en"
                 , "&cx=", google.cx
                 , "&fields=items(link)"
    )
    googlesearch = httr::GET(url)
    content = rawToChar(googlesearch$content); Encoding(content) = "UTF-8"
    searchresults = jsonlite::fromJSON(content)

    if (length(searchresults)!=0) { # only proceed if google search finds at least one url

      wikiurl = searchresults$items$link[1]

     # Wikipedia API -> using the page title, parse the album page and extract album scores
       # do some cleaning to the url to get the proper page name
         # deal with special characters
         pgname = curl::curl_unescape(wikiurl)
         # keep the end of the url -> the page name wikipedia api wants
         pgname = gsub("https://.*\\.wikipedia\\.org.*/","",pgname)

       # get the language of the wiki article
         wikilang = gsub("https://","",wikiurl)
         wikilang = gsub("\\.wikipedia.org.*","",wikilang)

       # obtain page content
         # try english language first, otherwise use language of url
         error = 0
         tryCatch(
             expr    = {webpage = WikipediR::page_content(language="en",project="wikipedia",pa
           , error   = function(e){}
           , finally = {tryCatch(
                   expr = {webpage = WikipediR::page_content(language=wikilang,project="wikipe
                 , error = function(e){print(paste0("Skipping url: ",wikiurl)); assign("error"
                 , finally = {})})

       if (error!=1) { # only proceed if wikipedia url worked
       # parse wikipedia content
       webpage = rvest::read_html(webpage$parse$text$`*`)
       wikiscores = rvest::html_nodes(webpage, xpath='//*[contains(concat( " ", @class, " " )
       wikiscores_stars = rvest::html_attr(rvest::html_element(wikiscores,css='span'),'title')
       wikiscores = rvest::html_text(wikiscores,trim=T)
```

```
    # some formatting for the scores -> convert them all into standard numeric out of 100
    wikiscores_data = as.data.frame(matrix(wikiscores,ncol=2,byrow=T))
    wikiscores_stars_data = as.data.frame(matrix(wikiscores_stars,ncol=2,byrow=T))
      # deal with specific review websites
       selector = (!is.na(wikiscores_stars_data$V2) & !(tolower(wikiscores_data$V1)=="tom l
       wikiscores_data$V2[selector] = wikiscores_stars_data$V2[selector]
       wikiscores_data = wikiscores_data[wikiscores_data$V1!="Metacritic",] # this is a re
       wikiscores_data = wikiscores_data[wikiscores_data$V1!="AnyDecentMusic?",] # this is
      # remove [] and content inside
       wikiscores_data$V2 = gsub("\\[.*?\\]","",wikiscores_data$V2)
      # remove () but NOT CONTENT (sometimes the score is contained within ()
       wikiscores_data$V2 = stringr::str_replace(wikiscores_data$V2,"\\(","")
       wikiscores_data$V2 = stringr::str_replace(wikiscores_data$V2,"\\)","")
      # fix ill-behaved special characters
       wikiscores_data$V2 = sapply(wikiscores_data$V2 , function(x) utils::URLencode(x) )
       wikiscores_data$V2 = stringr::str_replace(wikiscores_data$V2,"%E2%88%92","-")
       wikiscores_data$V2 = stringr::str_replace(wikiscores_data$V2,"%E2%80%93","-")
       wikiscores_data$V2 = stringr::str_replace(wikiscores_data$V2,"%E2%80%94","-")
       wikiscores_data$V2 = sapply(wikiscores_data$V2 , function(x) utils::URLdecode(x) )
       wikiscores_data$V2 = stringr::str_replace(wikiscores_data$V2,"\\?\\^\\'","-")
    # covert letter grade to numeric
      wikiscores_data$V2 = sapply(stringr::str_trim(wikiscores_data$V2) , lettergrade_num)
      # remove unit of measure (stars or discs)
       wikiscores_data$V2 = stringr::str_replace(wikiscores_data$V2,"stars","")
       wikiscores_data$V2 = stringr::str_replace(wikiscores_data$V2,"discs","")
      # remove %'s
       wikiscores_data$V2 = stringr::str_replace(wikiscores_data$V2,"%","")
      # remove remaining scores that contain no numbers
       wikiscores_data = wikiscores_data[grepl("\\d",wikiscores_data$V2),]

      # convert to numeric
      wikiscores_data$V2 = sapply(wikiscores_data$V2,
                                  function(x) tryCatch(eval(parse(text=x)),
                                                       error=function(e){},
                                                       finally=NULL) )
    # final set of scores to put in dataset
      wikiscores_data = wikiscores_data[!sapply(wikiscores_data$V2,is.null),]
      wikiscores_data$V2 = as.numeric(wikiscores_data$V2)
      wikiscores_data$V2[wikiscores_data$V2<=1] = 100*wikiscores_data$V2[wikiscores_data$V
  ##########################################################################

  ## CRITIC SCORES FROM WIKIPEDIA ##########################################
    release_date_wiki = rvest::html_nodes(webpage, xpath='//*[contains(concat( " ", @class
    release_date_wiki = rvest::html_text(release_date_wiki,trim=T)
    release_date_wiki = release_date_wiki[2]
```

```
      ##############################################################################

      wiki_scores = list(wikiscores_data$V2)

    } else { # if no wiki article found, set variables to empty
      release_date_wiki = NA
      wiki_scores = list(numeric(0))
    }


    # grow the dataset with the additional tracks and album review info
      artistdata = rbind(artistdata,
                  cbind(artist_name,
                        artist_id,
                        artist$followers$total,
                        example_album$name,
                        id,
                        example_album$popularity,
                        example_album$release_date,
                        album_scrobbles,
                        release_date_wiki,
                        wiki_scores))

    print(cbind(artist_name,
                artist_id,
                artist$followers$total,
                example_album$name,
                id,
                example_album$popularity,
                example_album$release_date,
                album_scrobbles,
                release_date_wiki,
                wiki_scores))
    } # end of if(error!=1)
    } # end of album loop
} # end of artist loop
colnames(artistdata) = c("artist_name","artist_id","artist_followers","album_name","album_id",
                         "album_popularity", "album_releasedate","album_scrobbles","release_da
artistdata = as.data.frame(artistdata)
##########################################################
```

References

Amblee, Naveen and Tung Bui. 2007. "Freeware downloads: An empirical investigation into the impact of expert and user reviews on demand for digital goods." *AMCIS 2007 Proceedings* p. 21.

Brennan, Matt. 2006. "The rough guide to critics: musicians discuss the role of the music press." *Popular Music* 25(2):221–234.

Eliashberg, Jehoshua and Steven M Shugan. 1997. "Film critics: Influencers or predictors?" *Journal of marketing* 61(2):68–78.

Huang, Peng, Nicholas H Lurie and Sabyasachi Mitra. 2009. "Searching for experience on the web: An empirical examination of consumer behavior for search and experience goods." *Journal of marketing* 73(2):55–69.

Morey, Justin. 2008. "Arctic Monkeys—The Demos vs. The Album." *Journal on the Art of Record Production* 4.

Reinstein, David A and Christopher M Snyder. 2005. "The influence of expert reviews on consumer demand for experience goods: A case study of movie critics." *The journal of industrial economics* 53(1):27–51.

Sorensen, Alan T and Scott J Rasmussen. 2004. "Is any publicity good publicity? A note on the impact of book reviews." *NBER Working Paper, Stanford University* .

Waldfogel, Joel. 2015. 14. Digitization and the Quality of New Media Products: The Case of Music. In *Economic analysis of the digital economy*. University of Chicago Press pp. 407–442.

West, Patricia M and Susan M Broniarczyk. 1998. "Integrating multiple opinions: The role of aspiration level on consumer response to critic consensus." *Journal of Consumer Research* 25(1):38–51.