final_project

June 5, 2022

1 Exploring the impact of theme songs in Disney movies

by Christian Garrovillo

1.1 Foreword

This notebook will be showing some data analysis regarding `Disney` datasets located here

2 Introduction

2.1 Question(s) of interests

In this document, I will be exploring the effects that having a theme song has in the performance of a movie. More specifically, I will be attempting to answer the question:

Do movies with a theme song perform better than those without a theme song?

I chose this as based on personal experience, the most memorable Disney classics that comes to my mind can be associated with a theme song. Also, given that Disney has a large hand in the entertainment industry, I would not be surprised if they too have the data associated with theme songs; and that they use this data as fuel in their behind-the-scenes processes that outputs family favourites.

Thus, I predict that by the end of this analysis, we will find that titles with theme songs *do* in fact perform better than those without a theme song.

2.2 Dataset description

The Disney dataset is composed of 5 tables: `disney_movies_total_gross.csv`, `disney_revenue_1991-2016..csv`, `disney-characters.csv`, `disney-director.csv`, and `disney-voice-actors.csv`.

These datasets include various information regarding Disney's releases such as movie names, protagonist/antagonist characters, as well as information regarding the operations of each release such as box-office statistics, directors, and MPAA rating.

For my analysis, I will be narrowing my focus towards the following datasets:

- `disney_movies_total_gross.csv` - For gathering data about a movie's performance
- `disney-characters.csv` - For gathering data about a movie's main character, villain, and theme song.

3 Methods and Results

We'll begin by performing some basic setup, and taking a peek at the data structures we'll be working with.

We create copies of the original dataframe to preserve the original state, and so that we can reference it again later.

[1]:
```
# Required libraries needed for this analysis
import altair as alt
import pandas as pd
import numpy as np

# Importing Data
gross_orig = pd.read_csv('./data/disney_movies_total_gross.csv')
gross = gross_orig.copy()

characters_orig = pd.read_csv('./data/disney-characters.csv')
characters = characters_orig.copy()
```

[2]:
```
gross.sample(5)
```

[2]:
```
                      movie_title  release_date      genre MPAA_rating  \
408                      Mr. 3000  Sep 17, 2004     Comedy       PG-13
132              The Mighty Ducks   Oct 2, 1992     Comedy          PG
25   The Last Flight of Noah's Ark  Jun 25, 1980        NaN         NaN
251                The War at Home  Nov 20, 1996        NaN           R
83              The Little Mermaid  Nov 15, 1989  Adventure           G

     total_gross inflation_adjusted_gross
408   $21,800,302              $29,593,641
132   $50,752,337             $103,120,810
25    $11,000,000              $34,472,116
251       $34,368                  $65,543
83   $111,543,479             $223,726,012
```

[3]:
```
characters.sample(5)
```

[3]:
```
                               movie_title       release_date  \
10  \nThe Adventures of Ichabod and Mr. Toad    October 5, 1949
5                          \nSaludos Amigos   February 6, 1943
36                                 Tarzan      June 18, 1999
43                            Brother Bear   November 1, 2003
49                               \nTangled  November 24, 2010

                          hero                                villian  \
10  Mr. Toad and Ichabod Crane  Mr. Winkie and The Headless Horseman
5                  Donald Duck                                   NaN
```

```
36                     Tarzan                                 Clayton
43                      Kenai                                  Denahi
49                    Rapunzel                           Mother Gothel

                      song
10       The Merrily Song
5        Saludos Amigos\n
36  You'll Be in My Heart
43   Look Through My Eyes
49         I See the Light
```

From these tables, we have a basic understanding of what the datasets look like.

However, we should get as much relevant information as we can so we can approach the datasets with our best foot forward.

[4]:
```
gross.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 579 entries, 0 to 578
Data columns (total 6 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   movie_title              579 non-null    object
 1   release_date             579 non-null    object
 2   genre                    562 non-null    object
 3   MPAA_rating              523 non-null    object
 4   total_gross              579 non-null    object
 5   inflation_adjusted_gross  579 non-null    object
dtypes: object(6)
memory usage: 27.3+ KB
```

We can see that the `gross` dataset has 579 rows and 6 columns. For every `movie_title`, there exists: - A `release_date` - A `genre` - A `MPAA_rating` which determines the title's target demographic - A `total_gross` for the dollar amount that the title earned - and an `inflation_adjusted_gross` for the dollar amount that the title earned adjusted for inflation for the year 2016 (which is when the datasets were originally compiled)

[5]:
```
characters.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56 entries, 0 to 55
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   movie_title   56 non-null     object
 1   release_date  56 non-null     object
 2   hero          52 non-null     object
 3   villian       46 non-null     object
```

```
 4   song          47 non-null     object
dtypes: object(5)
memory usage: 2.3+ KB
```

We can see that the `characters` dataset has 56 rows and 5 columns. For every `movie_title`, there exists: - A `release_date` - A `hero` - A `villian` - and a `song` which is the theme song of the title.

Before we move forward any further, we will take steps in tidying our data so that our primary focus as we progress is on answering the analysis question being asked.

As we gathered more info about the `characters` dataset, we can easily see that the `villian` column is misspelled.

[6]:
```
# fix the spelling of the villian column.
characters = characters.rename(columns={'villian': 'villain'})
```

We can also change the data type of some columns to a type more appropriate for the data the column represents.

We'll change the `release_date` columns on both dataframes:

[7]:
```
# change release_date column type to datetime
gross['release_date'] = pd.to_datetime(gross['release_date'])
characters['release_date'] = pd.to_datetime(characters['release_date'])
```

Both `total_gross` & `inflation_adjusted_gross` columns on the `gross` dataframe are presented as of type *string*s, which makes it hard to analyze. We need to convert these into a type more appropriate for representing money such as *float*s, but the logic for this can easily get complicated and repetitive for the two columns.

To approach this, we can import a custom function that I created to clean up columns representing dollar amounts into a float.

[8]:
```
import parse_dollar_col as pdc

# convert the columns containing dollar values into a float using a utility␣
 ↪function.
gross = pdc.parse_dollar_col(gross, 'total_gross')
gross = pdc.parse_dollar_col(gross, 'inflation_adjusted_gross')
```

Our table structures are now as follows:

[9]:
```
gross.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 579 entries, 0 to 578
Data columns (total 6 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   movie_title             579 non-null    object
 1   release_date            579 non-null    datetime64[ns]
 2   genre                   562 non-null    object
```

```
 3   MPAA_rating               523 non-null    object
 4   total_gross               579 non-null    float64
 5   inflation_adjusted_gross  579 non-null    float64
dtypes: datetime64[ns](1), float64(2), object(3)
memory usage: 27.3+ KB
```

[10]:
```
characters.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56 entries, 0 to 55
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   movie_title   56 non-null     object
 1   release_date  56 non-null     datetime64[ns]
 2   hero          52 non-null     object
 3   villain       46 non-null     object
 4   song          47 non-null     object
dtypes: datetime64[ns](1), object(4)
memory usage: 2.3+ KB
```

The column names are fixed, and the data types that each column represents is now more accurate. But we are not done – some of the values of the characters dataset has extraneous values such as newline formatting characters, and there are also empty *strings* that we want to replace with *NaN* values for easier consumption later on.

[11]:
```
# remove newline formatting characters and replace empty strings with NaNs on␣
 ↪our characters dataframe.
characters = characters.replace('\n', '', regex=True).replace(r'^\s*$', np.nan,␣
 ↪regex=True)
characters.head(5)
```

[11]:

	movie_title	release_date	hero	villain	song
0	Snow White and the Seven Dwarfs	1937-12-21	Snow White	Evil Queen	Some Day My Prince Will Come
1	Pinocchio	1940-02-07	Pinocchio	Stromboli	When You Wish upon a Star
2	Fantasia	1940-11-13	NaN	Chernabog	NaN
3	Dumbo	1941-10-23	Dumbo	Ringmaster	Baby Mine
4	Bambi	1942-08-13	Bambi	Hunter	Love Is a Song

Upon further analysis of the `gross` dataset, we can also notice how there are duplicate values of the same `movie_title`, representing remakes. For accuracy, we'd like to only retain the movie with

the most recent `release_date`. This will make sense later when we filter our columns to those most relevant to answering our question.

[12]:
```
# remove duplicates (remakes) of movies, preferring the most recent release
gross = gross.sort_values('release_date', ascending=False).
 ↪drop_duplicates('movie_title').sort_index()
characters = characters.sort_values('release_date', ascending=False).
 ↪drop_duplicates('movie_title').sort_index()
```

Since we have a common column of `movie_title` between both dataframes, we can combine them into a single dataframe.

[13]:
```
# merge the characters & gross dataframes using movie_title as keys,  otherwise␣
 ↪excluding the data if the key does not exist on both dataframes.
characters_gross = characters.merge(gross, left_on='movie_title',␣
 ↪right_on='movie_title', how='inner')
characters_gross.head(5)
```

[13]:

	movie_title	release_date_x	hero	villain	song	release_date_y	genre	MPAA_rating	total_gross	inflation_adjusted_gross
0	Snow White and the Seven Dwarfs	1937-12-21	Snow White	Evil Queen	Some Day My Prince Will Come	1937-12-21	Musical	G	184925485.0	5.228953e+09
1	Pinocchio	1940-02-07	Pinocchio	Stromboli	When You Wish upon a Star	1940-02-09	Adventure	G	84300000.0	2.188229e+09
2	Fantasia	1940-11-13	NaN	Chernabog	NaN	1940-11-13	Musical	G	83320000.0	2.187091e+09
3	Cinderella	1950-02-15	Cinderella	Lady Tremaine	Bibbidi-Bobbidi-Boo	2015-03-13	Drama	PG	201151353.0	2.011514e+08
4	Alice in Wonderland	1951-07-28	Alice	Queen of Hearts	The Unbirthday Song	2010-03-05	Adventure	PG	334191110.0	3.570635e+08

Notice how the merge introduced duplicate columns `release_date_x` & `release_date_y`.

To reduce cognitive load, we can remove unrelated data and filter our columns to only those most relevant to us in answering our question:

- `movie_title`
- `song`
- `genre`
- `MPAA_rating`
- `inflation_adjusted_gross`

[14]:
```
# drop columns that are irrelevant to us in answering our questions
characters_gross = characters_gross.filter(items=['movie_title', 'song',␣
↪'genre', 'MPAA_rating', 'inflation_adjusted_gross'], axis=1)
characters_gross.sample(5)
```

[14]:

	movie_title	song	genre	MPAA_rating	inflation_adjusted_gross
0	Snow White and the Seven Dwarfs	Some Day My Prince Will Come	Musical	G	5.228953e+09
13	The Black Cauldron	NaN	Adventure	NaN	5.055314e+07
33	Chicken Little	NaN	Adventure	G	1.779547e+08
2	Fantasia	NaN	Musical	G	2.187091e+09
29	Lilo & Stitch	He Mele No Lilo	Adventure	PG	2.115067e+08

Note: Here we also removed the `total_gross` column. This is important in making sure our analysis is accurate since our dataset hosts a wide range of movies released from 1937 upto 2016, therefore our conclusions must be drawn using data as if a title were released in 2016 – hence the use of `inflation_adjusted_gross`.

Now that we have taken precautions by tidying our data, we can begin connecting the dots.

To start, we can first separate titles that have songs and those with no songs, and grouping them together. However, the column `song` as it stands right now is not ideal since the question we are asking requires a binary answer; it either has a song or does not. For this reason, we can create a new column `has_song` that determines whether a movie has a song or does not. This makes it easier for us to group them together later.

[15]:
```
# add a has_song column value for each row, dependent on their song value
characters_gross.loc[~pd.isna(characters_gross['song']), 'has_song'] = 'Yes'
characters_gross.loc[pd.isna(characters_gross['song']), 'has_song'] = 'No'
characters_gross.sample(5)
```

[15]:

	movie_title	song	genre	MPAA_rating	inflation_adjusted_gross	has_song
10	The Many Adventures of Winnie the Pooh	Winnie the Pooh	NaN	NaN	0.000000e+00	Yes
33	Chicken Little	NaN	Adventure	G	1.779547e+08	No
6	Sleeping Beauty	Once Upon a Dream	Drama	NaN	2.150583e+07	Yes
21	Pocahontas	Colors of the Wind				
0	Snow White and the Seven Dwarfs	Some Day My Prince Will Come				

```
21  Adventure           G              2.743710e+08      Yes
0     Musical           G              5.228953e+09      Yes
```

With that new *Nominal* column in the dataframe, let us put them into separate dataframes so we can visualize each bins individually.

[16]:
```
# put the movies into separate dataframes depending on whether they have a␣
↪theme song or not
characters_gross_w_song = characters_gross[characters_gross['has_song'] ==␣
↪'Yes']
characters_gross_wo_song = characters_gross[characters_gross['has_song'] ==␣
↪'No']
```

[17]:
```
(alt.Chart(characters_gross_w_song, width=700, height=200)
    .mark_bar()
    .encode(
        x=alt.X('movie_title:N', title='Movie', sort='y'),
        y=alt.Y('inflation_adjusted_gross:Q', title='Performance ($)')
    )
    .properties(title="Performance of movies that have a theme song"))
```

[17]:
```
alt.Chart(…)
```

[18]:
```
(alt.Chart(characters_gross_wo_song, width=700, height=200)
    .mark_bar(color='#FFE66D')
    .encode(
        x=alt.X('movie_title:N', title='Movie', sort='y'),
        y=alt.Y('inflation_adjusted_gross:Q', title='Performance ($)')
    )
    .properties(title="Performance of movies that do not have a theme song"))
```

[18]:
```
alt.Chart(…)
```

There are two things we can infer from these graphs:

- The variance of sample size between the two groups are way too far.
- Within a group, there is always at least 1 outlier of exceeding Performance.

[19]:
```
characters_gross_w_song.shape[0] / characters_gross_wo_song.shape[0]
```

[19]:
```
6.333333333333333
```

[20]:
```
characters_gross_w_song[:1]
```

[20]:
```
                      movie_title                          song    genre  \
0  Snow White and the Seven Dwarfs  Some Day My Prince Will Come  Musical

  MPAA_rating  inflation_adjusted_gross has_song
```

```
0           G              5.228953e+09      Yes
```

[21]:
```
characters_gross_wo_song[:1]
```

[21]:
```
   movie_title song    genre MPAA_rating  inflation_adjusted_gross has_song
2    Fantasia  NaN  Musical           G              2.187091e+09       No
```

To attain an accurate conclusion that effectively answers our question, we must match the sample sizes to make sure that ensure a fair comparison. We could use the function `.sample(6)` against the group of the larger size (`characters_gross_w_song`), which creates a random assortment of 6 movies to represent the group.

Having said that though, in the random chance that a sample includes a movie that is an outlier, the data is now an inaccurate representation of the group as the outlier will weight heavier than others within the sample.

Therefore, before we take a sample we must remove an equal amount of outliers between the two groups, so that we can create a sample size from the larger group that remains to be an accurate representation of the data.

Since one group has a very small sample size (`characters_gross_wo_song`), we'll remove only 1 outlier from both groups:

[22]:
```
# remove 1 outlier from both groups
characters_gross_w_song = characters_gross_w_song.
 ↪sort_values('inflation_adjusted_gross', ascending=False)[1:]
characters_gross_wo_song = characters_gross_wo_song.
 ↪sort_values('inflation_adjusted_gross', ascending=False)[1:]
```

[23]:
```
(alt.Chart(characters_gross_w_song, width=700, height=200)
    .mark_bar()
    .encode(
        x=alt.X('movie_title:N', title='Movie', sort='y'),
        y=alt.Y('inflation_adjusted_gross:Q', title='Performance')
    )
    .properties(title="Performance of movies that have a theme song"))
```

[23]:
```
alt.Chart(…)
```

[24]:
```
(alt.Chart(characters_gross_wo_song, width=700, height=200)
    .mark_bar(color='#FFE66D')
    .encode(
        x=alt.X('movie_title:N', title='Movie', sort='y'),
        y=alt.Y('inflation_adjusted_gross:Q', title='Performance')
    )
    .properties(title="Performance of movies that do not have a theme song"))
```

[24]:
```
alt.Chart(…)
```

We can now take a sample size from the larger group more accurately.

[25]:
```
characters_gross_w_song_sampled = characters_gross_w_song.sample(5)
characters_gross_w_song_sampled
```

[25]:
```
        movie_title                          song      genre MPAA_rating  \
1         Pinocchio   When You Wish upon a Star  Adventure           G
11     The Rescuers                The Journey  Adventure         NaN
29    Lilo & Stitch            He Mele No Lilo  Adventure          PG
30  Treasure Planet             I'm Still Here  Adventure          PG
24            Mulan  I'll Make a Man Out of You  Adventure           G

    inflation_adjusted_gross has_song
1               2.188229e+09      Yes
11              1.597439e+08      Yes
29              2.115067e+08      Yes
30              5.518914e+07      Yes
24              2.168078e+08      Yes
```

Now that we have these bins, we can group the data together based on whether they have a song or not, take the average `inflation_adjusted_gross` out of both groups and finally answer our first question:

Do movies with a theme song perform better than those without a theme song?

[26]:
```
# combine the bins back together, now that the outliers in both are removed
characters_gross = pd.concat([characters_gross_w_song_sampled,␣
↪characters_gross_wo_song])

# group by whether or not a movie has a song and get the mean␣
↪inflation_adjusted_gross
characters_gross_song_mean = characters_gross.groupby('has_song').
↪mean('inflation_adjusted_gross').reset_index()
characters_gross_song_mean
```

[26]:
```
  has_song  inflation_adjusted_gross
0       No               113530736.4
1      Yes               566295329.0
```

It seems that our hypothesis is correct. Movies **with** a theme song outperform those **without** a theme song.

Let's visualize this:

[27]:
```
(alt.Chart(characters_gross_song_mean, width=700, height=200)
    .mark_bar()
    .encode(
        x=alt.X('inflation_adjusted_gross:Q', title='Performance'),
        y=alt.Y('has_song:O', title='Movie has a theme song')
```

```
    )
    .properties(title="Performance of movies based on the existence of a theme␣
↪song"))
```

[27]: `alt.Chart(…)`

4 Discussions

In this report, I analyzed two datasets regarding Disney movies released from 1937 to 2016 in the interest of answering the question:

Do movies with a theme song perform better than those without a theme song?

To achieve an accurate answer, I separated the movies into 2 groups dependent on whether a movie has a theme song or not. Coincidentally, I also found that Disney produced 6x more movies with theme songs than those without. To maintain a fair comparison between the two groups, I took a random sample from the group of movies with a theme song; And to make sure the sample is accurate, I removed 1 outlier from both groups.

To no surprise, movies with a theme song performed better than those without. I expected this, as without involving mathematics or science for a second and instead thinking about the cultural impact that music has on children, emotions and society as a whole; it becomes clear that movies with theme songs would have the greater reach. This conclusion is powerful as it quantifies a rather vague variable that can be used in the decision making process when creating a Disney movie. By also looking at the large number of movies with theme songs, it could be assumed that Disney has known this fact and takes it into account behind-the-scenes.

In addition to our question, there are other interesting insights that could be researched and cannot be found through the datasets used in this report alone. It would be interesting to see whether the target audience makes a difference at all – *do the impact of theme songs saturate as an audience gets older*? It would also be interesting to see if the production costs of a theme song has an impact if any – *does spending more money on creating a theme song lead to family favorites*?

5 References

- Data Source

Christian Garrovillo