Burton

March 24, 2021

1 Assignment 3 - Novel Visualization or Library

By: Tia Burton

1.1 A Scatter Chart for Ratings: An Introduction

I love movies, and I think it's fair to assume that there is a large population that enjoys movies as much as I do. Even before the pandemic, new streaming services were cropping up everwhere. However, the big 4, including Disney+ in this scenario, stayed on top. While thinking about movies to watch next, I decide to search the internet for the best movies list. To my surprise, someone had already compiled the movies, with ratings from two sources, into one spreadsheet and left it on Kaggle for public to use. If you're interested in taking a look directly at the data itself from the source, here is a link: https://www.kaggle.com/ruchi798/movies-on-netflix-prime-video-hulu-and-disney.

After selecting this dataset, I initially thought the best technique would be to plot to the movies themselves based on runtime and ratings in a scatter chart. I then took a quick look in Excel at the data and saw there were over 16,000 rows! It would not be most effective to plot all the individual movie titles at this time; I would like to know the high level differences for only the genres I like. So, that means, no fantasy, no sci-fy, no crime shows, and absolutely no Westerns. If I am able to first get the high level results, I'd be more excited to return to those individual movie titles in the service and age range of interest to do more comparisons.

The scatter chart would encompass the following: 1) Genres 2) Average movie ratings per genre per suggested age group 3) Count of movies per genre per suggested age group 4) The associated streaming service.

For each streaming service, there will be one plot or animated instance. Within that instance, the genres will be compare the number of movies available against the average ratings by age group. This would enable movie lovers, like myself, to quickly compare the ratings and number of movies for their next movie night.

With this level of complexity, the idea of multiple plots and animated instances might sounded challenging. For this reason, I selected plotly express. I will now briefly explain the advantages of using plotly to complete this visualization task.

1.2 Visualization Library: Plotly Express

Matplotlib and seaborn have been around for quite some time. Due to their early headstart and extremely enthusiastic open source partners, these libraries have grown to include more capabilities that have proven to be extremely useful, but there are still a few remaining limiations that led me to select plotly express.

Plotly is known as a nuanced plotting library that's also growing through it's open source collaboration, but it was originally founded in Canada by Chris and Jack Parmer, Matthew Sundquist, and Alex Johnson in 2012. It has now grown to around around 40 charts to fit a wide variety of use cases (statistical, financial, etc.). While this library was initially created in JavaScipt, the Python API allows users to declaratively create aesthetically pleasing, web-based visualizations. This API is free to use, but there is a more advance version, Dash, that is for sale with web application integration capabilities.

Given the list of variables in the previous section, Plotly Express does make creating this visualization easy. Unlike Matplotlib, Seaborn, and regular Plotly, Plotly Express is a quick way to build high level visuals with minimal commands. This library focuses on the key elements needed to properly communicate the right information. Plotly express allows vis builders to assemble charts, display labels, and create a legend in once line of code. Does this all sound too good to be true?

Plotly Express Scatter: https://plotly.com/python-api-reference/generated/plotly.express.scatter

I invite you to take a peak at the visualization in the last cell or the scatter link above if you are curious and would like to interact with the visualization at this time.

1.3 Installation of Plotly Express

In order to use this tool, general plotly 4, that includes plotly express, must be installed with the following instructions. No other rendering is needed for plotly as the packages contain everything needed to run successfully in a jupyter notebook.

```
To install Plotly:

1. Using the command line / terminal : $ pip install plotly==4.14.3

2. Using conda: $ conda install -c plotly plotly=4.14.3

3. Jupyter Notebook: $ pip install "notebook>=5.3" "ipwidgets>=7.5"
```

For additional support or for JupyterLab: 1. https://plotly.com/python/getting-started/ 2. https://plotly.com/python/getting-started/#jupyter-notebook-support

[1]:
```
# Let's get started with importing the necessary libraries

import numpy as np # utilizing aggregation methods for ratings
import pandas as pd # high level analysis tool
import plotly.express as px # primary graphing library for the scatter plot
import plotly.graph_objects as go #plotly go for future exploration or␣
 ↪additional traces to be added
import plotly.io as pio # additional plotly api that minimizes the commands␣
 ↪needed to run plotly in the notebook
```

1.4 Demonstration

1.4.1 Data Cleaning

To help us visualize the information, we must first bring in the dataset and perform some data cleaning in order to utilize this dataset to help us find the ratings by service, genre, and age group.

This dataset was retrieved from Kaggle. Even though I had access to other public information on these companies, such as stock price and movie descriptions, we will be focused on a singular source of data for this exercise.

[2]:
```
# Let's read in our CSV file with pandas.read_csv and take a look
streaming_movies = pd.read_csv('MoviesOnStreamingPlatforms_updated.
 ↪csv',index_col=0)
streaming_movies.head()
```

[2]:

	ID	Title	Year	Age	IMDb	Rotten Tomatoes	\
0	1	Inception	2010	13+	8.8	87%	
1	2	The Matrix	1999	18+	8.7	87%	
2	3	Avengers: Infinity War	2018	13+	8.5	84%	
3	4	Back to the Future	1985	7+	8.5	96%	
4	5	The Good, the Bad and the Ugly	1966	18+	8.8	97%	

	Netflix	Hulu	Prime Video	Disney+	Type	Directors	\
0	1	0	0	0	0	Christopher Nolan	
1	1	0	0	0	0	Lana Wachowski,Lilly Wachowski	
2	1	0	0	0	0	Anthony Russo,Joe Russo	
3	1	0	0	0	0	Robert Zemeckis	
4	1	0	1	0	0	Sergio Leone	

	Genres	Country	\
0	Action,Adventure,Sci-Fi,Thriller	United States,United Kingdom	
1	Action,Sci-Fi	United States	
2	Action,Adventure,Sci-Fi	United States	
3	Adventure,Comedy,Sci-Fi	United States	
4	Western	Italy,Spain,West Germany	

	Language	Runtime
0	English,Japanese,French	148.0
1	English	136.0
2	English	149.0
3	English	116.0
4	Italian	161.0

Ok, so now that we have our dataset. We're going to focus on the three parts of this data set that matter the most to us. We need the Genres, Type, Netflix, Hulu, Prime Video, Disney+, Age, and both ratings. We can go ahead and drop the other columns.

Nonetheless, let's continue with this investigation. As you can see in the example above, Rotten Tomatoes has a percentage where IMDb has a floating point. We must get the data on the same scale.

[3]:
```
# First, we want to manipulate the Rotten Tomatoes column. We will change the .
# We can just add it to the column header to help us remember that the value␣
 ↪represents a percentage.
# We can now utilize the pandas astype function with numpy float to help us␣
 ↪turn these columns types into
# Numpy floating point numbers.
```

```
streaming_movies['Rotten Tomatoes'] = streaming_movies['Rotten Tomatoes'].str.
 ↪replace('%','')
streaming_movies = streaming_movies.rename(columns={'Rotten Tomatoes':'Rotten␣
 ↪Tomatoes (%)'})
streaming_movies.IMDb = streaming_movies.IMDb.astype(np.float)
streaming_movies['Rotten Tomatoes (%)'] = streaming_movies['Rotten Tomatoes␣
 ↪(%)'].astype(np.float)



# Since we are looking to get the average percentage, and IMDb rates on a scale␣
 ↪from 0 - 10,
# we multiply the entire column by 10 to change the denominator of the data.␣
 ↪Remember, we removed
# the percentage sign and added it to the header

clean_ratings = streaming_movies[ streaming_movies['Rotten Tomatoes (%)'].
 ↪notna() & streaming_movies['IMDb'].notna() ]
clean_ratings.IMDb = clean_ratings.IMDb * 10
clean_ratings = clean_ratings.rename(columns={'IMDb':'IMDb (%)'})
clean_ratings['Composite (%)'] = (clean_ratings['IMDb (%)'] +␣
 ↪clean_ratings['Rotten Tomatoes (%)']) / 2

clean_ratings = clean_ratings.drop(columns=
    ['ID', 'Year', 'Type', 'Directors','Country','Language','Runtime']
)
```

[4]:
```
print('streaming_movies is greater than clean_ratings:', len(streaming_movies)␣
 ↪> len(clean_ratings))
print('imax max:', clean_ratings['IMDb (%)'].max(),', imax min:',␣
 ↪clean_ratings['IMDb (%)'].min())
print('rt max:', clean_ratings['Rotten Tomatoes (%)'].max(), ', rt min:',␣
 ↪clean_ratings['Rotten Tomatoes (%)'].min())
clean_ratings.head(3)
```

```
streaming_movies is greater than clean_ratings: True
imax max: 90.0 , imax min: 16.0
rt max: 100.0 , rt min: 2.0
```

[4]:

	Title	Age	IMDb (%)	Rotten Tomatoes (%)	Netflix	Hulu	Prime Video	Disney+	Genres	Composite (%)
0	Inception	13+	88.0	87.0	1	0	0	0	Action,Adventure,Sci-Fi,Thriller	87.5
1	The Matrix	18+	87.0	87.0	1	0				
2	Avengers: Infinity War	13+	85.0	84.0	1	0				

```
1            0        0                      Action,Sci-Fi           87.0
2            0        0            Action,Adventure,Sci-Fi           84.5
```

[5]:
```
# The next step is important because it will help us separate the genres␣
 ↪without losing data.
# We want to split the string in the Genre column
clean_ratings['Genres'] = clean_ratings['Genres'].str.split(',')
```

[6]:
```
# Next we must create a list of all the unique genres. I created helper␣
 ↪functions to first
# allow us retreive those unique values that were nested within lists in the␣
 ↪'Genre' column.
# The second function will help us keep track of the columns we've already␣
 ↪created. The last
# function allows us to use the apply function on our existing pandas dataframe␣
 ↪to create
# new columns and set the value to 1 for any movie that has that has a defined␣
 ↪genre.

genre_list = []


def to_list(item):
    if item != type(int) and item not in genre_list:
        genre_list.append(item)
    return item

def helper_func(df):

    if df.name == 'Genres':
        for index,genres in enumerate(df):
            if type(genres) == float:
                pass
            elif type(genres) == list and len(genres) > 1:
                for genre in genres:
                    to_list(genre)
            elif type(genres) == list and len(genres) == 1:
                to_list(genres[0])
    else:
        return df
    return df

def genre_assignment(assignee):
    count = 0
    if assignee.name == 'Genres':
        for genre in assignee:
            try:
```

```
                for g in genre:
                    clean_ratings[g].iloc[count] = 1
            except TypeError:
                pass
            count += 1
    return assignee

clean_ratings = clean_ratings.apply(helper_func)

# Create a column for each genre
for genre in genre_list:
    clean_ratings[genre] = np.NaN

# Uncomment the next line to see a sample of the data frame with the NaN genre␣
 ↪columns.
#clean_ratings.sample(3)
```

[7]:
```
# We can now assign 1.0 within the column for that specific genre using the␣
 ↪genre_assignment function that takes in a row of data,
# looks for the 'Genres' column, and either inputs the values or passes if␣
 ↪there is a TypeError (np.NaN is considered a float
# and python doesn't iterate over floating point numbers).

favorite_cr = clean_ratings.apply(genre_assignment).reset_index()

# The last step is to filter down the columns and rows once more to only focus␣
 ↪on the genres that are most important to me.
# As well as the age groups This section is dynamic, so if another genre was␣
 ↪more suitable or perferred,
# I encourage change and exploration
# while following along!

favorite_cr = favorite_cr.drop(
    columns= ['IMDb (%)', 'Rotten Tomatoes (%)','Western', 'Animation',
              'Family', 'Biography','Music', 'War', 'Crime', 'Fantasy',
              'Sci-Fi','History','Sport','Musical', 'News', 'Short',
               'Reality-TV', 'Talk-Show', 'Game-Show', 'Film-Noir' ]
)

favorite_cr = favorite_cr[favorite_cr.Age.notna()]
```

[8]:
```
favorite_cr.sample(3)
```

[8]:

	index	Title	Age	Netflix	Hulu	Prime Video	\
4478	7873	Mi America	18+	0	0	1	
1442	3575	Hunt for the Wilderpeople	13+	0	1	0	
3278	5697	Life of a King	13+	0	0	1	

```
      Disney+                     Genres  Composite (%)  Action  Adventure  \
4478        0              [Crime, Drama]           57.0     NaN        NaN
1442        0  [Adventure, Comedy, Drama]           87.5     NaN        1.0
3278        0                     [Drama]           56.0     NaN        NaN

      Thriller  Comedy  Drama  Romance  Mystery  Horror  Documentary
4478       NaN     NaN    1.0      NaN      NaN     NaN          NaN
1442       NaN     1.0    1.0      NaN      NaN     NaN          NaN
3278       NaN     NaN    1.0      NaN      NaN     NaN          NaN
```

Now, we can utilize this data to create two dataframes. One will hold the Ratings that are a percentage. The other will hold the total count for movies.

[9]:
```
def rating_by_gen(first_filter, variable_name):
    df = favorite_cr[ favorite_cr[first_filter] == 1]
    comp_for_age_gen = df[ favorite_cr[variable_name]== 1 ]
    return comp_for_age_gen.groupby(['Age'])['Composite (%)'].agg(np.average)

def total_count(first_filter, variable_name):
    df = favorite_cr[ favorite_cr[first_filter] == 1]
    comp_for_age_gen = df[ favorite_cr[variable_name]== 1 ]
    return comp_for_age_gen.groupby(['Age'])['Title'].agg('count')

big_4 = ['Netflix','Hulu','Prime Video','Disney+']
big_4_avg = ['Netflix_Avg_','Hulu_Avg_','Prime Video_Avg_','Disney+_Avg_']
streaming_genres = ['Action', 'Adventure', 'Thriller', 'Comedy','Drama',␣
 ↪'Romance', 'Mystery', 'Horror', 'Documentary']

Ratings_by_age = pd.DataFrame()
Totals = pd.DataFrame()

for ind,service in enumerate(big_4):
    for genre in streaming_genres:
        name = big_4_avg[ind] + genre
        tot_name = big_4[ind] + ' ' + genre
        Ratings_by_age[name] = rating_by_gen(genre, service)
        Totals[tot_name] = total_count(genre, service)

#Let's take a look at our Ratings by age, platform,
Ratings_by_age = Ratings_by_age.transpose().reset_index()
Ratings_by_age['Service'] = Ratings_by_age['index'].str.split('_')
Ratings_by_age['Genre'] = [i[2] for i in Ratings_by_age['Service']]
Ratings_by_age['Service'] = [i[0] for i in Ratings_by_age['Service']]
Ratings_by_age = Ratings_by_age.groupby(['Service','Genre']).agg(lambda x: x)
Ratings_by_age = Ratings_by_age.drop(columns='index')


#For the intermediate results, uncomment the .sample() lines of code.
```

```
Totals_by_age = pd.DataFrame(Totals.stack(),columns=['Total Movies'])
Totals_by_age = Totals_by_age.reset_index()
Totals_by_age.sample(3)

Ratings_by_age = pd.DataFrame(Ratings_by_age.reset_index(
).set_index(['Service','Genre']).stack(
), columns=['Ratings by Percent']).reset_index()

Ratings_by_age['Join_Col'] = Ratings_by_age['Service'] + ' ' +␣
 ↪Ratings_by_age['Genre']
#Ratings_by_age.sample(3)
#Totals_by_age.sample(3)
```

[10]:
```
# The last step to to get the data in the form we'd like it to be in, we must␣
 ↪merge the two dataframes that we created
# in the last cell.

final_df = Ratings_by_age.merge( Totals_by_age, how='left',
                                left_on=['Join_Col','Age'],
                                right_on=['level_1','Age'])


# Plotly express takes an entire dataframe as the first argument, so we want to␣
 ↪specify the index and change the name of
# any columns that we will need when plotting. Plotly conveniently takes the␣
 ↪string passed as the title for
# the axis or element.

final_df['level_1'] = final_df['Service']
movies_final = final_df.rename(columns={
    'Join_Col': 'Service_Genre',
    'Service': 'index',
    'level_1': 'Service',
}).set_index('index')

movies_final.head()
```

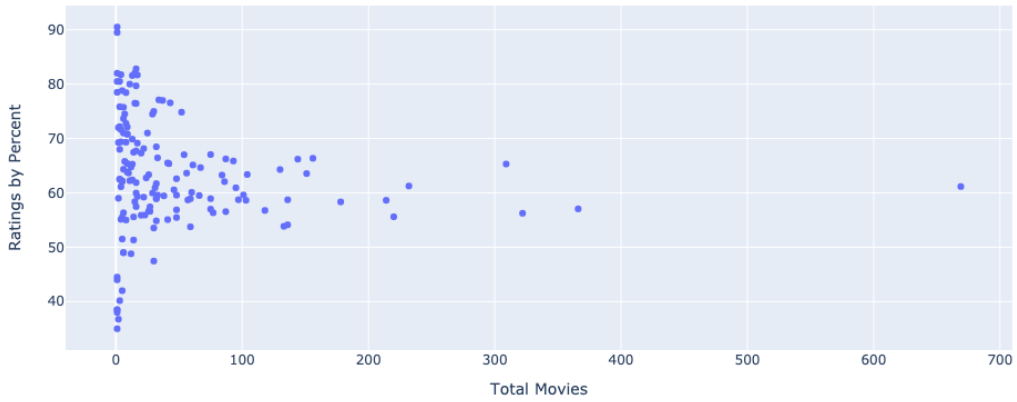
[10]:
```
             Genre  Age  Ratings by Percent      Service_Genre  Service  \
index
Disney+     Action  13+           74.482759     Disney+ Action  Disney+
Disney+     Action   7+           63.346154     Disney+ Action  Disney+
Disney+     Action  all           63.650000     Disney+ Action  Disney+
Disney+  Adventure  13+           74.983333  Disney+ Adventure  Disney+
Disney+  Adventure   7+           67.040000  Disney+ Adventure  Disney+

         Total Movies
index
Disney+          29.0
```

```
Disney+          26.0
Disney+          10.0
Disney+          30.0
Disney+          75.0
```

1.5 Data Visualization

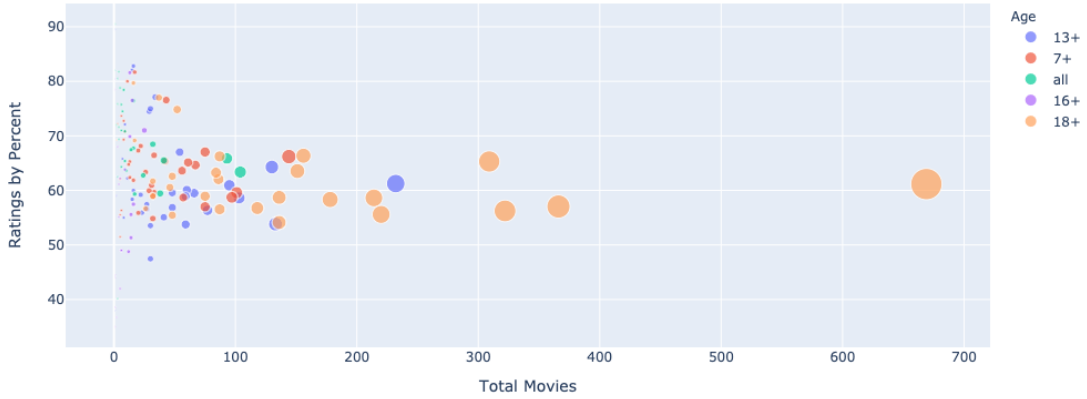
We now have our clean data set and are now ready to engage the Plotly Express library. I will add in a few components at each time to showcase the individual components that build to more advanced features, like animation frame and animation group.

[12]:
```
# Using the final_movies dataframe, we can use the string names from the␣
 ↪dataframe
# to declare what we would like to see and where. There's no option to create␣
 ↪subplots with plotly express as
# that is a more intensive building process. With the simple command, we can␣
 ↪still built a very comprehensive
# data visualization.

#First we pass it our dataframe and declare our x and y values.
fig1 = px.scatter(movies_final,
                  x='Total Movies',
                  y='Ratings by Percent')

#To view this plot just reiterate the name
fig1
```



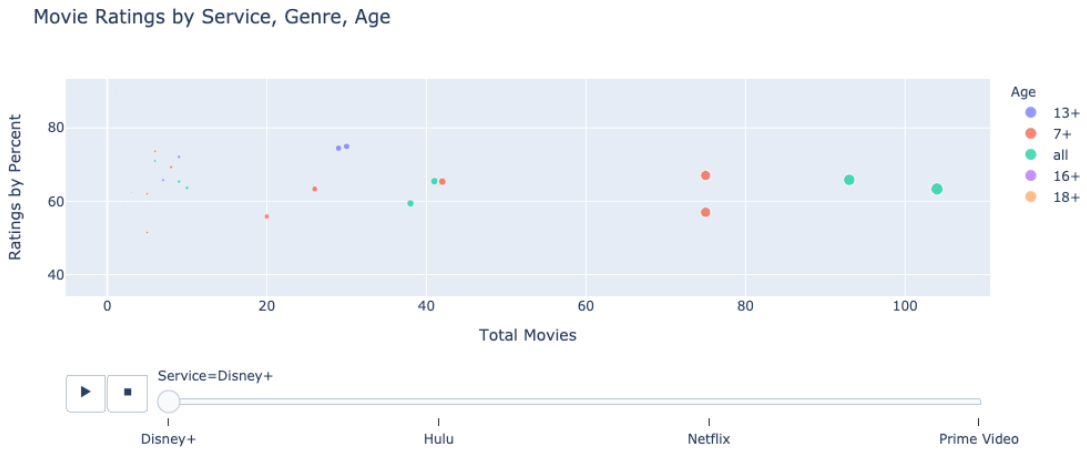
[13]:
```
# Ok, we see a few things. As previously mentioned, we have our x and y label,␣
 ↪and if you hover over the
# any of the dots you will see that plotly express automatically includes␣
 ↪interactive markers.
```

```
# that's about all we can gather from this visualization. However, we can␣
 ↪continue to customize this chart
# by using the size and color parameters to start differentiating the ages and␣
 ↪genres from one another in
# in the same space. Notice how Plotly automatically gives a legend for 'Age'␣
 ↪passed to color.


fig = px.scatter(movies_final,
                 x='Total Movies',
                 y='Ratings by Percent',
                 size='Total Movies',
                 color='Age',
                hover_name='Service_Genre')
fig
```



[14]:
```
# Wow, that really bought the chart to life. We can now see the different␣
 ↪colors for the age groups, the sizes
# also help to contextualize how large the collection is to one before looking␣
 ↪at the x-axis. Lastly,
# if you hover over the bubbles, you'll find that the name of the service is␣
 ↪now included. It is helpful,
# but it doesn't immediately communication what is being represented here. We␣
 ↪can add a header directly to our
# next figure when creating it, so we can clarify it's purpose.

# We'll now want to build the final chart. This is where the "animation" comes␣
 ↪in. Early on, I mentioned how
# Plotly Express doesn't allow subplots like Plotly does. Depending on the use␣
 ↪case, it might be more beneficial
```

```
# To start a project with plotly.go instead of px. In this scenario, we␣
 ↪actually give the illusion of sub
# plots with a twist by using the animation features. Since this is categorical␣
 ↪data, the play axis, usually
# used with time axis, acts as our selection tool. It also plays, but the␣
 ↪motion doesn't make too much sense
# because it's just switching between the different categorical elements.


# To enable our animation, We'll need both the animation frame and animation␣
 ↪group.
# Without the animation_frame parameter, animation_frame alone will not provide␣
 ↪interactivity.

final_fig = px.scatter(movies_final,
                       x='Total Movies',
                       y='Ratings by Percent',
                       size='Total Movies',
                       color='Age',
                       animation_group='Genre',
                       animation_frame='Service',
                       hover_name='Service_Genre',
                      title='Movie Ratings by Service, Genre, Age')
final_fig
```



Now, with service selection, movies can be compared by the amount of content available to the rating separately. If you click from Disney+ to Prime video, you can see that Prime Video and Netflix are two larger platforms, where Disney+ and Hulu have less.

Another interesting point in this visualization is that there are lower ratings for some movies that have a larger selection. A plausible explanation could be: Given the popularity of a genre,

a company will invest more money; thus more content gets release. As the content gets more exposure, there are more points where the content can be criticized. If the genre and content library stays small, it is possible that a niche audience is giving a lot of praise.

A future step for me will be to investigate the movies I'm most interested in via the ratings from this chart, but I think I'll start on one of the the platforms first then come back to see if I agree with the ratings.

Check out more Plotly Express examples here: https://plotly.com/python/plotly-express/