Analyzing Video Game Sales

February 18, 2019

1 Analyzing Video Game Sales

Jacob Miller The following dataset comes from Kaggle (https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings) and captures video game sales and Metacritic ratings for video games ranging from 1980 to 2016. The data is organized by descending order of Global_Sales. I am going to look at sales by platform, genre, geographical region, developer and game rating. The goal will be to get an understanding of optimal markets to target as a video game developer.

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
```

```
In [2]: df = pd.read_csv('Video_Games_Sales_as_of_22_Dec_2016.csv')
```

Let's start by getting a basic understanding of what the data looks like. I'll look at the shape of the dataframe, the columns, the first 5 entries, and then focus in on the very first entry for clarity.

```
In [3]: df.shape
```

```
Out[3]: (16719, 16)
```

```
In [4]: df.columns
```

```
Out[4]: Index(['Name', 'Platform', 'Year_of_Release', 'Genre', 'Publisher', 'NA_Sales',
               'EU_Sales', 'JP_Sales', 'Other_Sales', 'Global_Sales', 'Critic_Score',
               'Critic_Count', 'User_Score', 'User_Count', 'Developer', 'Rating'],
              dtype='object')
```

```
In [5]: df.head()
```

```
Out[5]:                        Name Platform  Year_of_Release         Genre Publisher  \
        0                Wii Sports      Wii           2006.0        Sports  Nintendo
        1         Super Mario Bros.      NES           1985.0      Platform  Nintendo
        2            Mario Kart Wii      Wii           2008.0        Racing  Nintendo
        3         Wii Sports Resort      Wii           2009.0        Sports  Nintendo
        4  Pokemon Red/Pokemon Blue       GB           1996.0  Role-Playing  Nintendo
```

```
   NA_Sales  EU_Sales  JP_Sales  Other_Sales  Global_Sales  Critic_Score  \
0     41.36     28.96      3.77         8.45         82.53          76.0
1     29.08      3.58      6.81         0.77         40.24           NaN
2     15.68     12.76      3.79         3.29         35.52          82.0
3     15.61     10.93      3.28         2.95         32.77          80.0
4     11.27      8.89     10.22         1.00         31.37           NaN

   Critic_Count User_Score  User_Count Developer Rating
0          51.0          8       322.0  Nintendo      E
1           NaN        NaN         NaN       NaN    NaN
2          73.0        8.3       709.0  Nintendo      E
3          73.0          8       192.0  Nintendo      E
4           NaN        NaN         NaN       NaN    NaN
```

```
In [6]: df.iloc[0]
```

```
Out[6]: Name                  Wii Sports
        Platform                     Wii
        Year_of_Release             2006
        Genre                     Sports
        Publisher               Nintendo
        NA_Sales                   41.36
        EU_Sales                   28.96
        JP_Sales                    3.77
        Other_Sales                 8.45
        Global_Sales               82.53
        Critic_Score                  76
        Critic_Count                  51
        User_Score                     8
        User_Count                   322
        Developer               Nintendo
        Rating                         E
        Name: 0, dtype: object
```

Now that we have a better understanding of the dataset, let's check what NaN values we have.

```
In [7]: df[df.isnull().any(axis = 1)].shape[0]
```

```
Out[7]: 9894
```

Ok, well that's a lot of rows to drop. Let's look at just the rows with NaN in the Name column. Even if one of the other columns is NaN (e.g. Critic_Score) we can still get useful information from most of the other columns.

```
In [8]: df[df['Name'].isnull()]
```

```
Out[8]:       Name Platform  Year_of_Release Genre             Publisher  NA_Sales  \
        659    NaN      GEN           1993.0   NaN  Acclaim Entertainment      1.78
        14246  NaN      GEN           1993.0   NaN  Acclaim Entertainment      0.00
```

```
       EU_Sales  JP_Sales  Other_Sales  Global_Sales  Critic_Score  \
659        0.53      0.00         0.08          2.39           NaN
14246      0.00      0.03         0.00          0.03           NaN

       Critic_Count User_Score  User_Count Developer Rating
659             NaN        NaN         NaN       NaN    NaN
14246           NaN        NaN         NaN       NaN    NaN
```

That's a lot more reasonable. We only have 2 games that are lacking names all together, so they really don't provide much useful information. Let's just drop these 2 rows. You can see from the shape afterwards that 2 rows were dropped.

```
In [9]: df.dropna(axis = 0, subset = ['Name'], inplace = True)
        df.shape
```

```
Out[9]: (16717, 16)
```

We now have a really good understanding of how our data looks and how it is presented, so let's take a little detour and check out some of my favorite games. First, I'll look at how many Final Fantasy games are included in the entire dataset.

```
In [10]: df[df['Name'].str.contains('Final Fantasy')].shape[0]
```

```
Out[10]: 91
```

91 games is a little too many to display here, so I'll look at the top 25. Then, I'll focus in on my favorite one of the series, Final Fantasy IX.

```
In [11]: df[df['Name'].str.contains('Final Fantasy')][:25]
```

```
Out[11]:                                  Name Platform  Year_of_Release  \
         65              Final Fantasy VII       PS           1997.0
         84                Final Fantasy X      PS2           2001.0
         88             Final Fantasy VIII       PS           1999.0
         148             Final Fantasy XII      PS2           2006.0
         173            Final Fantasy XIII      PS3           2009.0
         175              Final Fantasy IX       PS           2000.0
         177             Final Fantasy X-2      PS2           2003.0
         388             Final Fantasy III     SNES           1994.0
         431   Crisis Core: Final Fantasy VII      PSP           2007.0
         578          Final Fantasy XIII-2      PS3           2011.0
         632               Final Fantasy V     SNES           1992.0
         633         Final Fantasy Tactics       PS           1997.0
         723       Dissidia: Final Fantasy      PSP           2008.0
         759            Final Fantasy XIII     X360           2010.0
         776  Final Fantasy Tactics Advance      GBA           2003.0
         802             Final Fantasy III       DS           2006.0
         985              Final Fantasy II     SNES           1991.0
```

1236	Final Fantasy XIV: A Realm Reborn	PC	2010.0
1267	Final Fantasy: Crystal Chronicles	GC	2003.0
1276	Dirge of Cerberus: Final Fantasy VII	PS2	2006.0
1383	Final Fantasy III	NES	1990.0
1411	Final Fantasy XII: Revenant Wings	DS	2007.0
1569	Final Fantasy X / X-2 HD Remaster	PS3	2013.0
1654	Final Fantasy IV	DS	2007.0
1703	Final Fantasy I & II: Dawn of Souls	GBA	2004.0

	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	\
65	Role-Playing	Sony Computer Entertainment	3.01	2.47	3.28	
84	Role-Playing	Sony Computer Entertainment	2.91	2.07	2.73	
88	Role-Playing	SquareSoft	2.28	1.72	3.63	
148	Role-Playing	Square Enix	1.88	0.00	2.33	
173	Role-Playing	Square Enix	1.74	1.21	1.87	
175	Role-Playing	SquareSoft	1.62	0.77	2.78	
177	Role-Playing	Electronic Arts	1.92	1.08	2.11	
388	Role-Playing	SquareSoft	0.86	0.00	2.55	
431	Role-Playing	Square Enix	1.35	0.59	0.80	
578	Role-Playing	Square Enix	0.78	0.73	0.89	
632	Role-Playing	SquareSoft	0.00	0.00	2.43	
633	Role-Playing	SquareSoft	0.93	0.12	1.34	
723	Fighting	Square Enix	0.51	0.50	0.91	
759	Role-Playing	Square Enix	1.28	0.67	0.01	
776	Role-Playing	SquareSoft	0.82	0.37	0.89	
802	Role-Playing	Square Enix	0.89	0.04	1.07	
985	Role-Playing	Square	0.24	0.09	1.33	
1236	Role-Playing	Square Enix	0.88	0.48	0.00	
1267	Role-Playing	Nintendo	0.72	0.38	0.36	
1276	Shooter	Square Enix	0.47	0.37	0.52	
1383	Role-Playing	SquareSoft	0.00	0.00	1.39	
1411	Role-Playing	Square Enix	0.33	0.41	0.54	
1569	Role-Playing	Square Enix	0.43	0.36	0.32	
1654	Simulation	Square Enix	0.51	0.04	0.62	
1703	Role-Playing	Nintendo	0.64	0.24	0.29	

	Other_Sales	Global_Sales	Critic_Score	Critic_Count	User_Score	\
65	0.96	9.72	92.0	20.0	9.2	
84	0.33	8.05	92.0	53.0	8.7	
88	0.23	7.86	90.0	24.0	8.6	
148	1.74	5.95	92.0	64.0	7.6	
173	0.51	5.33	83.0	83.0	7.3	
175	0.14	5.30	94.0	22.0	8.9	
177	0.17	5.29	85.0	45.0	6.6	
388	0.02	3.42	NaN	NaN	NaN	
431	0.43	3.18	83.0	67.0	8	
578	0.23	2.63	79.0	53.0	6.6	
632	0.02	2.45	NaN	NaN	NaN	

```
633           0.06           2.45          83.0           12.0          8.2
723           0.31           2.23          79.0           61.0            8
759           0.21           2.16          82.0           54.0          6.3
776           0.05           2.13          87.0           44.0          7.8
802           0.09           2.08          77.0           45.0          7.1
985           0.12           1.77           NaN            NaN          NaN
1236          0.17           1.52           NaN            NaN          NaN
1267          0.04           1.49          80.0           55.0          8.1
1276          0.12           1.48          57.0           51.0          6.9
1383          0.01           1.40           NaN            NaN          NaN
1411          0.10           1.37          81.0           44.0          6.1
1569          0.16           1.27          85.0           50.0          8.5
1654          0.05           1.21          85.0           52.0          7.7
1703          0.02           1.19           NaN            NaN          NaN

      User_Count                     Developer Rating
65        1282.0                    SquareSoft      T
84        1056.0                    SquareSoft      T
88         644.0                    SquareSoft      T
148        972.0                   Square Enix      T
173       2483.0                   Square Enix      T
175        779.0                    SquareSoft      T
177        400.0                    SquareSoft      T
388          NaN                           NaN    NaN
431        463.0                   Square Enix      T
578        707.0                   Square Enix      T
632          NaN                           NaN    NaN
633        190.0                    SquareSoft      T
723        139.0                   Square Enix      T
759        539.0                   Square Enix      T
776        212.0                   Square Enix      E
802        136.0               Matrix Software   E10+
985          NaN                           NaN    NaN
1236         NaN                           NaN    NaN
1267        94.0                   Square Enix      T
1276       103.0                   Square Enix      T
1383         NaN                           NaN    NaN
1411        60.0  Square Enix, Think and Feel   E10+
1569       332.0                       Virtuos      T
1654       164.0               Matrix Software   E10+
1703         NaN                           NaN    NaN
```

In [12]:
```python
df[df['Name'] == 'Final Fantasy IX'].T
```

Out[12]:
```
                              175
Name             Final Fantasy IX
Platform                       PS
Year_of_Release              2000
```

```
Genre                Role-Playing
Publisher              SquareSoft
NA_Sales                     1.62
EU_Sales                     0.77
JP_Sales                     2.78
Other_Sales                  0.14
Global_Sales                  5.3
Critic_Score                   94
Critic_Count                   22
User_Score                    8.9
User_Count                    779
Developer              SquareSoft
Rating                          T
```

Next, let's start doing a little analysis on the dataset. In order to not skew the analysis based on a few outlier reviews, I'm going to drop any rows that both less than 5 Critic_Scores and less than 5 User_Scores. Below, you can see that User_Score is a string, so we have to cast it as a number to be able to check how many reviews there actually are.

```
In [13]: print('Critic_Score type: ' + str(type(df['Critic_Score'][0])))
         print('User_Score type: ' + str(type(df['User_Score'][0])))

Critic_Score type: <class 'numpy.float64'>
User_Score type: <class 'str'>
```

```
In [14]: try:
             df['User_Score'] = df['User_Score'].astype(np.float64)
         except ValueError as e:
             print('--- Oops! --- \nValueError: ' + str(e))

--- Oops! ---
ValueError: could not convert string to float: 'tbd'
```

Huh... ok. So numbers within the User_Score column are stored as strings, but there are *also* non-numeric string values. Let's see how many of those we have.

```
In [15]: len(df[df['User_Score'] == 'tbd'])

Out[15]: 2425
```

Wow. That's a lot of scores to be determined, so now let's drop them all. If we are doing an investigation of ratings, it won't help us much to carry around a bunch of useless ratings.

```
In [16]: df.drop(df[(df['User_Score'] == 'tbd')].index, inplace = True)
         len(df[df['User_Score'] == 'tbd'])

Out[16]: 0
```

Now that we've cleared that up, we'll cast the User_Scores into a numeric-type, and then we'll look at which games have less than 5 reviews.

```
In [17]: df['User_Score'] = df['User_Score'].astype(np.float64)
         df[(df['Critic_Count'] < 5) & (df['User_Count'] < 5)]
```

```
Out[17]:                                      Name Platform  Year_of_Release  \
         1567                Battle Arena Toshinden       PS           1994.0
         2420                           Ford Racing       PS           2001.0
         3924                              The BIGS      PS2           2007.0
         4219      Dragon Ball Z: Collectible Card Game      GBA           2002.0
         4689                 Marvel Super Hero Squad       DS           2009.0
         5029                     Get Fit with Mel B      PS3           2010.0
         6994                   Gundam Battle Assault       PS           1998.0
         7367     SBK X: Superbike World Championship      PS3           2010.0
         8075                     Six Flags Fun Park       DS           2008.0
         8701                Chaotic: Shadow Warriors     X360           2009.0
         9774                                   NCIS      PS3           2011.0
         10157      Cloudy With a Chance of Meatballs      Wii           2009.0
         10177              Astro Boy: The Video Game      PS2           2009.0
         10618                          Sudoku Mania       DS           2006.0
         10656               Rolling Stone: Drum King      Wii           2009.0
         11258                         Puss in Boots      PS3           2011.0
         11549                        Hidden Invasion      PS2           2001.0
         13413                   Cabela's Trophy Bucks     X360           2007.0
         13632                           LEGO Friends      3DS           2013.0
         14801                   MX vs. ATV Supercross      PS3           2014.0
         14891                          Madden NFL 07      GBA           2006.0
         14897                               AniMates!       DS           2007.0
         15044                     NBA Starting Five       XB           2002.0

                   Genre                     Publisher  NA_Sales  EU_Sales  \
         1567   Fighting  Sony Computer Entertainment      0.39      0.26
         2420     Racing           Empire Interactive      0.48      0.33
         3924     Sports         Take-Two Interactive      0.25      0.19
         4219       Misc                   Infogrames      0.33      0.12
         4689   Fighting                          THQ      0.36      0.02
         5029     Sports             Black Bean Games      0.15      0.16
         6994   Fighting          Namco Bandai Games      0.13      0.09
         7367     Racing             Black Bean Games      0.05      0.12
         8075       Misc          Brash Entertainment      0.17      0.00
         8701     Action                   Activision      0.14      0.00
         9774  Adventure                      Ubisoft      0.07      0.04
         10157  Platform                      Ubisoft      0.10      0.00
         10177    Action                  D3Publisher      0.05      0.04
         10618    Puzzle        Zoo Digital Publishing      0.09      0.00
         10656      Misc                    505 Games      0.09      0.00
         11258    Action                          THQ      0.07      0.01
```

11549	Shooter	Swing! Entertainment	0.04	0.03
13413	Sports	Activision Value	0.04	0.00
13632	Action	Warner Bros. Interactive Entertainment	0.00	0.04
14801	Racing	Nordic Games	0.02	0.01
14891	Sports	Electronic Arts	0.02	0.01
14897	Simulation	Lexicon Entertainment	0.02	0.00
15044	Sports	Konami Digital Entertainment	0.02	0.01

	JP_Sales	Other_Sales	Global_Sales	Critic_Score	Critic_Count \
1567	0.53	0.08	1.27	69.0	4.0
2420	0.00	0.06	0.86	53.0	4.0
3924	0.00	0.06	0.51	71.0	4.0
4219	0.00	0.01	0.46	62.0	4.0
4689	0.00	0.03	0.41	61.0	4.0
5029	0.00	0.07	0.38	73.0	4.0
6994	0.00	0.02	0.23	61.0	4.0
7367	0.00	0.04	0.21	73.0	4.0
8075	0.00	0.01	0.18	34.0	4.0
8701	0.00	0.01	0.16	60.0	4.0
9774	0.00	0.02	0.12	50.0	4.0
10157	0.00	0.01	0.11	68.0	4.0
10177	0.00	0.01	0.11	56.0	4.0
10618	0.00	0.01	0.10	25.0	4.0
10656	0.00	0.01	0.10	32.0	4.0
11258	0.00	0.01	0.09	70.0	4.0
11549	0.00	0.01	0.08	39.0	4.0
13413	0.00	0.00	0.05	39.0	4.0
13632	0.00	0.00	0.04	43.0	4.0
14801	0.00	0.00	0.03	58.0	4.0
14891	0.00	0.00	0.03	68.0	4.0
14897	0.00	0.00	0.03	33.0	4.0
15044	0.00	0.00	0.02	48.0	4.0

	User_Score	User_Count	Developer	Rating
1567	6.3	4.0	Tamsoft	T
2420	5.8	4.0	Toolbox Design	E
3924	4.8	4.0	Blue Castle Games	E
4219	5.5	4.0	ImaginEngine	E
4689	2.8	4.0	THQ	E10+
5029	5.8	4.0	Lightning Fish Games	E
6994	7.8	4.0	Natsume	T
7367	7.5	4.0	Milestone S.r.l	E10+
8075	7.0	4.0	Brash Entertainment	E
8701	5.0	4.0	FUN Labs	E10+
9774	4.0	4.0	Ubisoft	T
10157	5.3	4.0	Ubisoft Shanghai	E
10177	5.0	4.0	High Voltage Software	E10+
10618	3.5	4.0	FrontLine Studios	E

```
10656          1.8          4.0                  505 Games      T
11258          7.0          4.0                        THQ   E10+
11549          5.8          4.0                       Toka      T
13413          3.0          4.0                   FUN Labs      T
13632          8.3          4.0              Hellbent Games      E
14801          3.5          4.0    Nordic Games Publishing      E
14891          9.3          4.0       Exient Entertainment      E
14897          2.5          4.0               Dreamcatcher      E
15044          5.5          4.0                     Konami      E
```

Great. So as badly as someone may want to "Get Fit With Mel B", apparently not many other people did. Let's take a look at the shape of the dataframe before dropping these games, then we'll drop them and take a look at the shape afterwards to make sure everything went ok. We can tell that 23 games were dropped.

```
In [18]: print('--- Shape before dropping games with <5 reviews ---\n' + \
               str(df.shape))

         df.drop(df[(df['Critic_Count'] < 5) & (df['User_Count'] < 5)].index,
                    inplace = True)

         print('\n--- Shape after dropping games with <5 reviews ---\n' + \
               str(df.shape))
```

```
--- Shape before dropping games with <5 reviews ---
(14292, 16)

--- Shape after dropping games with <5 reviews ---
(14269, 16)
```

I'd like to see what the correlation is between certain categories in this dataframe, but in order to do that, we need to get the categories we want to look at into numeric values. We'll start by creating a new "integer_df"

```
In [19]: integer_df = df.copy()
```

This is where things get a little backwards. The goal is to convert all instances of one string value (e.g. "E10+") into a numeric representation (e.g. "1"). However, it turns out some of the Developers and Ratings are actually being interpreted as numeric values, even though they are obviously strings (e.g. "E10+"). For example, you can see below that the `np.unique` is trying (and failing) to compare floats and strings. I won't explain in detail how I was able to get to the bottom of this, but in order to resolve this, first we have to cast **all** values into strings, and then convert them to a numeric representation.

```
In [20]: try:
             np.unique(df['Rating'])
         except TypeError as e:
             print('--- Oops! --- \nTypeError: ' + str(e))
```

```
--- Oops! ---
TypeError: '<' not supported between instances of 'float' and 'str'
```

```
In [21]: integer_df['Developer'] = integer_df['Developer'].astype(str)
         integer_df['Rating'] = integer_df['Rating'].astype(str)
```

```
In [22]: def integer_dictionary(string_column):
             '''
             Returns a dictionary of {string : integer} for a column/series in dataframe
             '''
             return {string : i for i, string in \
                     enumerate(np.unique(integer_df[string_column]))}

         platform_ints = integer_dictionary('Platform')
         genre_ints = integer_dictionary('Genre')
         developer_ints = integer_dictionary('Developer')
         rating_ints = integer_dictionary('Rating')
```

```
In [23]: def apply_int_trans(string_column, integer_dict):
             '''
             Translates all instances of a string to an integer within one column
             '''
             return integer_df[string_column].apply(lambda x: integer_dict[x])

         integer_df['Platform'] = apply_int_trans('Platform', platform_ints)
         integer_df['Genre'] = apply_int_trans('Genre', genre_ints)
         integer_df['Developer'] = apply_int_trans('Developer', developer_ints)
         integer_df['Rating'] = apply_int_trans('Rating', rating_ints)
```

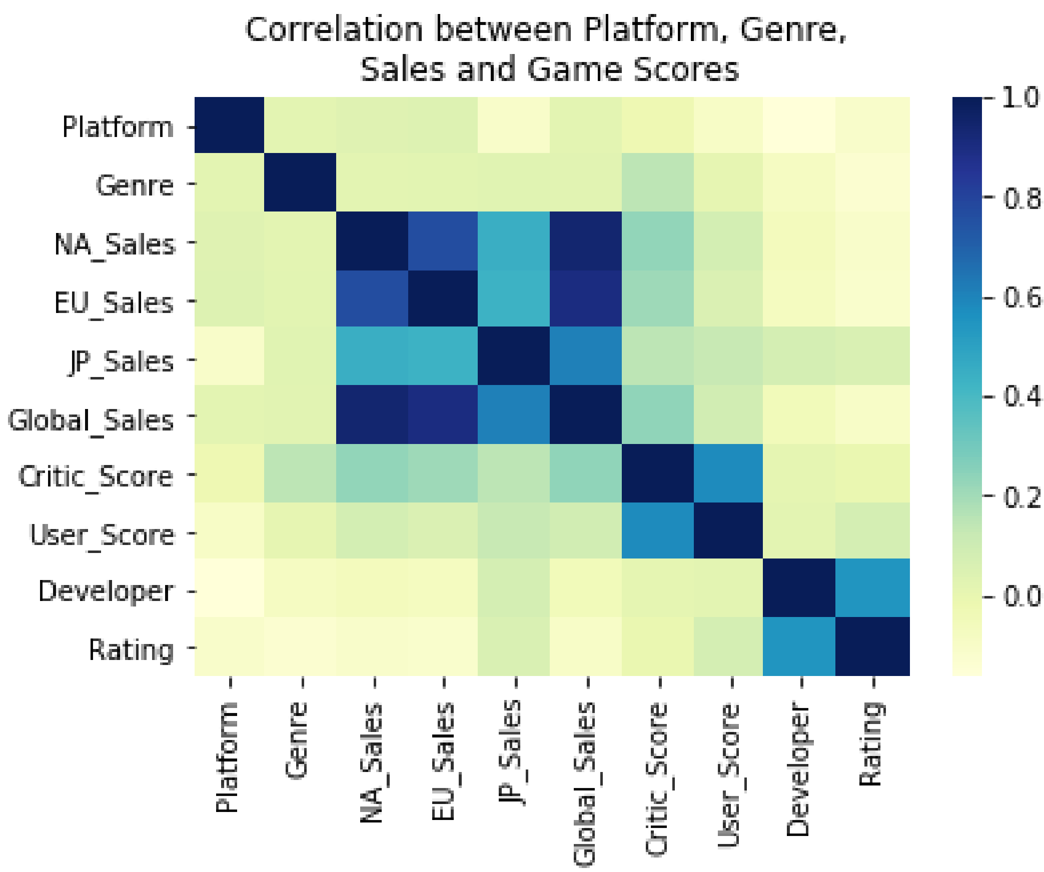
You can see below that Platform, Genre, Developer and Rating have now all been converted to numeric representations. We can now use these to plot a heatmap and see what type of correlations we have.

```
In [24]: integer_df.head()
```

```
Out[24]:                         Name  Platform  Year_of_Release  Genre Publisher  \
         0                 Wii Sports        26           2006.0     10  Nintendo
         1          Super Mario Bros.        11           1985.0      4  Nintendo
         2             Mario Kart Wii        26           2008.0      6  Nintendo
         3          Wii Sports Resort        26           2009.0     10  Nintendo
         4   Pokemon Red/Pokemon Blue         5           1996.0      7  Nintendo

            NA_Sales  EU_Sales  JP_Sales  Other_Sales  Global_Sales  Critic_Score  \
         0     41.36     28.96      3.77         8.45         82.53          76.0
         1     29.08      3.58      6.81         0.77         40.24           NaN
         2     15.68     12.76      3.79         3.29         35.52          82.0
         3     15.61     10.93      3.28         2.95         32.77          80.0
         4     11.27      8.89     10.22         1.00         31.37           NaN
```

```
   Critic_Count  User_Score  User_Count  Developer  Rating
0          51.0         8.0       322.0        835       1
1           NaN         NaN         NaN       1380       8
2          73.0         8.3       709.0        835       1
3          73.0         8.0       192.0        835       1
4           NaN         NaN         NaN       1380       8
```

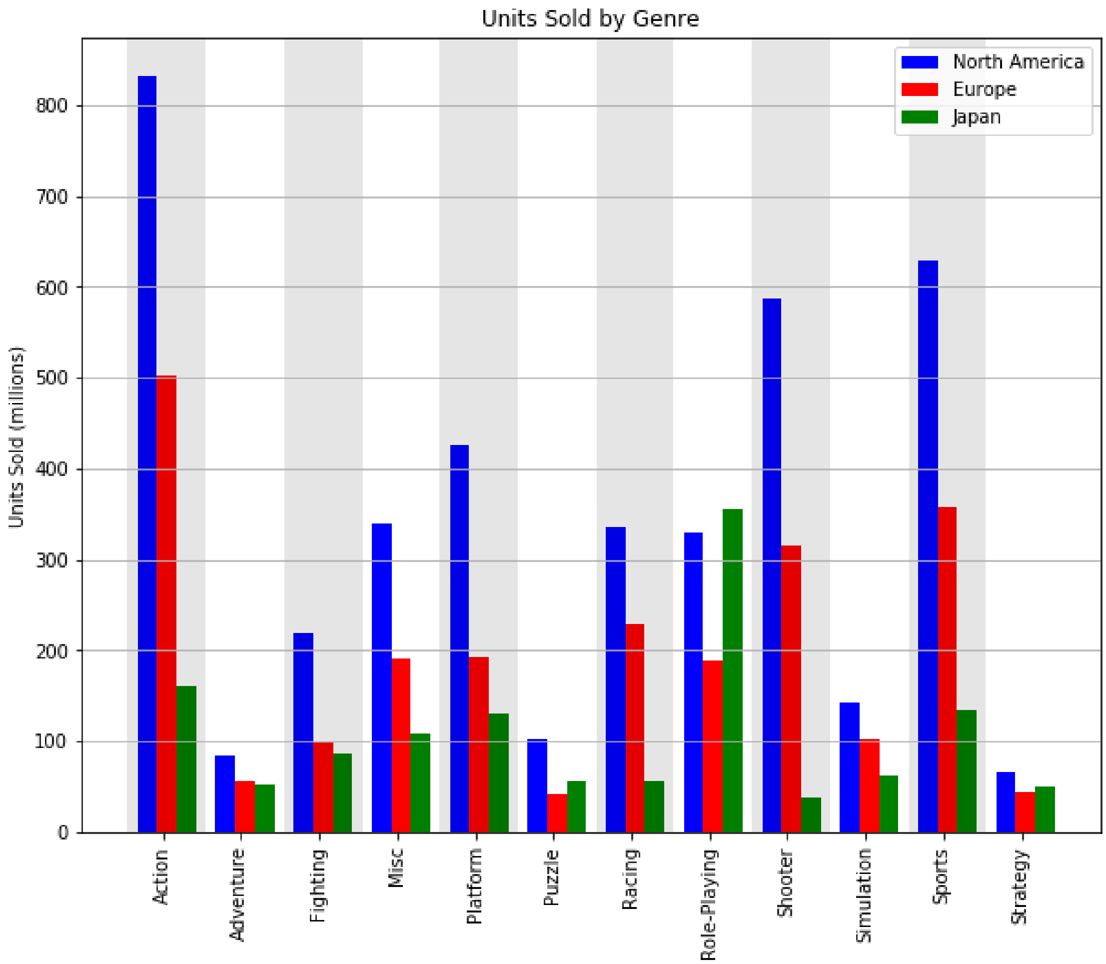
In [25]:
```python
correlation_features = ['Platform',
                        'Genre',
                        'NA_Sales',
                        'EU_Sales',
                        'JP_Sales',
                        'Global_Sales',
                        'Critic_Score',
                        'User_Score',
                        'Developer',
                        'Rating']

sns.heatmap(integer_df[correlation_features].corr(), cmap = 'YlGnBu')
plt.title('Correlation between Platform, Genre, \nSales and Game Scores')
plt.show()
```



Heatmaps are really fun and can help tease out some correlations in the data. For instance, it's probably pretty obvious that sales between various geographic regions are correlated, but I wouldn't have thought that Ratings and Developers were so closely correlated.

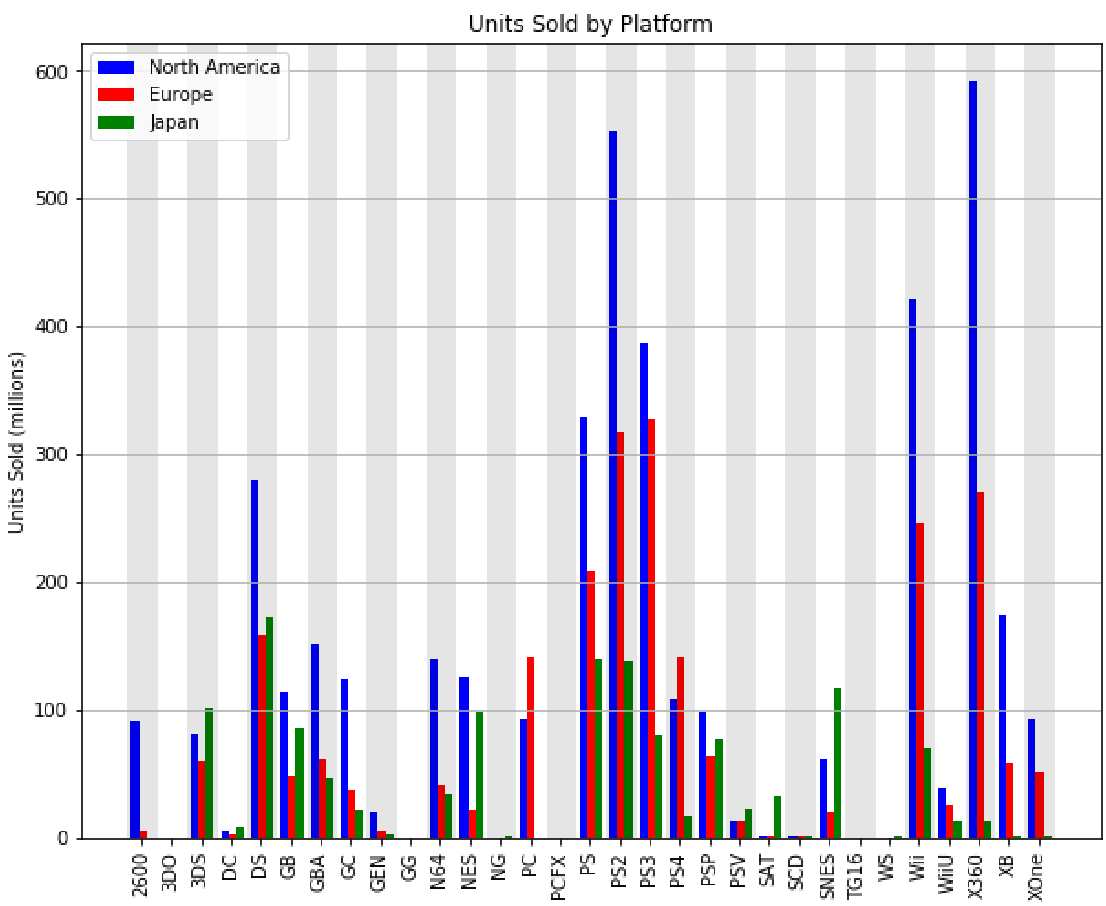
Let's look at how game sales break down by genre and by platform. If I was a game developer, this information would definitely help me decide what kind of game and platform to target. One thing that pops out right away is the different between the Japanese and North American sales in the Role-Playing and Shooter genres. **Role-Playing** games have much higher sales than any other genre in Japan, whereas Shooters are actually the least sold Genre. On the other hand, in North America, **Shooters** are the third most-sold genre and Role-Playing is the seventh.

```
In [26]: def sales_by_genre(dataframe, region):
             '''
             Returns a dictionary of {genre : sales in millions of units} for "region"
             '''
             return {genre : dataframe[dataframe['Genre'] == genre][region].sum() \
                     for genre in genre_ints.keys()}

         na_sales_by_genre = sales_by_genre(df, 'NA_Sales')
         eu_sales_by_genre = sales_by_genre(df, 'EU_Sales')
         jp_sales_by_genre = sales_by_genre(df, 'JP_Sales')

         genre_indices = range(len(na_sales_by_genre))
         genre_width = np.min(np.diff(genre_indices))/4

         fig, ax = plt.subplots(figsize = (10, 8))
         bar1 = ax.bar(genre_indices - genre_width, na_sales_by_genre.values(),
                       genre_width, color = 'blue')
         bar2 = ax.bar(genre_indices, eu_sales_by_genre.values(),
                       genre_width, color = 'red')
         bar3 = ax.bar(genre_indices + genre_width, jp_sales_by_genre.values(),
                       genre_width, color = 'green')

         plt.xticks(rotation = 90)
         ax.set_xticks(genre_indices - genre_width/5)
         ax.set_xticklabels((na_sales_by_genre.keys()))

         plt.title('Units Sold by Genre')
         plt.ylabel('Units Sold (millions)')
         plt.legend(labels = ['North America', 'Europe', 'Japan'], loc = 'best')
         plt.grid(which = 'major', axis = 'y')

         for i in range(0, len(na_sales_by_genre), 2):
             plt.axvspan(i - 0.5, i + 0.5, facecolor = 'black', alpha = 0.1)

         plt.show()
```




```
In [27]: def sales_by_platform(region):
             '''
             Returns a dictionary of {platform : sales in millions of units}
             for "region"
             '''
             return {platform : df[df['Platform'] == platform][region].sum() for \
                     platform in platform_ints.keys()}

         na_sales_by_platform = sales_by_platform('NA_Sales')
         eu_sales_by_platform = sales_by_platform('EU_Sales')
         jp_sales_by_platform = sales_by_platform('JP_Sales')

         platform_indices = range(len(na_sales_by_platform))
         platform_width = np.min(np.diff(platform_indices))/4

         fig, ax = plt.subplots(figsize = (10, 8))
```

```
bar1 = ax.bar(platform_indices - platform_width, na_sales_by_platform.values(),
              platform_width, color = 'blue')
bar2 = ax.bar(platform_indices, eu_sales_by_platform.values(),
              platform_width, color = 'red')
bar3 = ax.bar(platform_indices + platform_width, jp_sales_by_platform.values(),
              platform_width, color = 'green')

plt.xticks(rotation = 90)
ax.set_xticks(platform_indices - platform_width/5)
ax.set_xticklabels((na_sales_by_platform.keys()))

plt.title('Units Sold by Platform')
plt.ylabel('Units Sold (millions)')
plt.legend(labels = ['North America', 'Europe', 'Japan'], loc = 'best')
plt.grid(which = 'major', axis = 'y')

for i in range(0, len(na_sales_by_platform), 2):
    plt.axvspan(i - 0.5, i + 0.5, facecolor = 'black', alpha = 0.1)

plt.show()
```



In order to figure out what the top developers are (by number of games developed), we'll get all of the unique developers and the number of times they show up. Then, we'll create a dictionary that will allow us to first sort the developers by number of games developed, and then subsequently lookup developers numerically.

```
In [28]: devs, dev_counts = np.unique(integer_df['Developer'], return_counts = True)
         dev_counts_dict = {i : [dev_counts, i] for i, dev_counts in \
                            enumerate(dev_counts)}
         top_devs_integers = sorted(dev_counts_dict.values(), reverse = True)

         rev = dict({(v, k) for k, v in developer_ints.items()})
```

```
In [29]: rev[1000]
```

```
Out[29]: 'Rising Star Games'
```

Great! Now let's see who our top developer is...

```
In [30]: rev[top_devs_integers[0][1]]
```

```
Out[30]: 'nan'
```

Oops... Remember, we didn't want to drop 9000+ rows of data, but that means some NaNs will show up in our data. That's fine - we can work around them when we know where they are. So let's just look at the top 20 developers that are **not** NaN.

```
In [31]: top_20_integers = top_devs_integers[1:21]
         top_20_devs = [rev[dev[1]] for dev in top_20_integers]
         print('\n--- Top 20 Developers ---')
         for i, dev in enumerate(top_20_devs):
             print(str(i + 1) + '. ' + dev)
```

```

--- Top 20 Developers ---
1. EA Sports
2. EA Canada
3. Capcom
4. Ubisoft
5. Konami
6. EA Tiburon
7. Ubisoft Montreal
8. Visual Concepts
9. Omega Force
10. Electronic Arts
11. TT Games
12. Traveller's Tales
13. Nintendo
14. Vicarious Visions
```

```
15. Codemasters
16. Yuke's
17. Namco
18. Maxis
19. Neversoft Entertainment
20. Midway
```

Some familiar names there, but I'd never heard of "Vicarious Visions" before. Let's briefly investigate them as an aside. Turns out they developed many of the Guitar Hero and Crash Bandicoot games. Neat!

```
In [32]: df[df['Developer'] == 'Vicarious Visions']\
         [['Name', 'Developer', 'Critic_Score']].head(20)
```

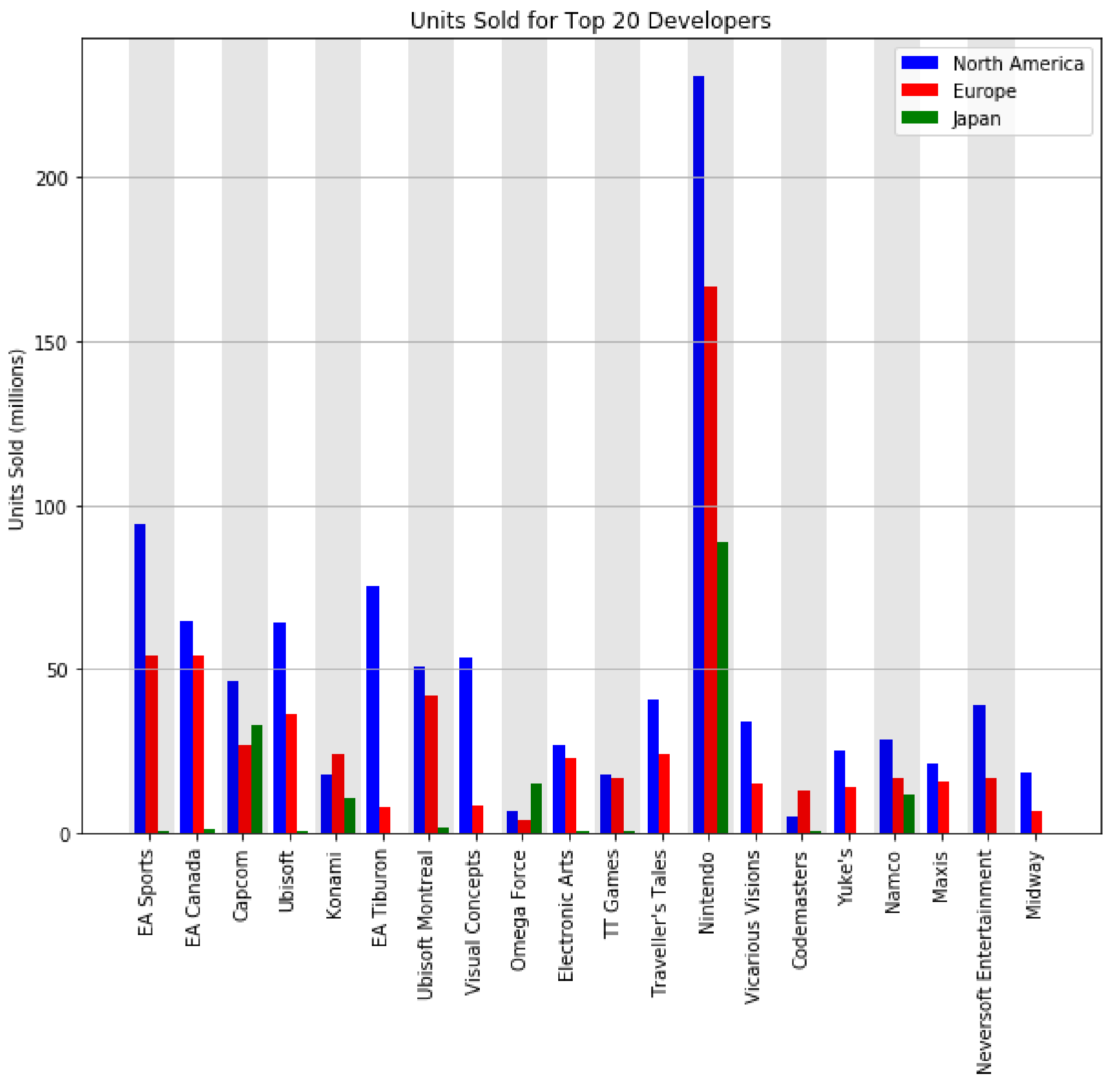
```
Out[32]:                                                   Name          Developer  \
         234                   Guitar Hero III: Legends of Rock  Vicarious Visions
         351                          Guitar Hero: World Tour  Vicarious Visions
         383                             Guitar Hero: On Tour  Vicarious Visions
         519                                     Finding Nemo  Vicarious Visions
         873                                 Crash Nitro Kart  Vicarious Visions
         1032               Crash Bandicoot: The Huge Adventure  Vicarious Visions
         1181                     Guitar Hero: On Tour Decades  Vicarious Visions
         1205                     Spider-Man 2: Enter: Electro  Vicarious Visions
         1206                                    Guitar Hero 5  Vicarious Visions
         1421                     Crash Bandicoot 2: N-Tranced  Vicarious Visions
         1454                                           Doom 3  Vicarious Visions
         1478                              Skylanders SWAP Force  Vicarious Visions
         1547                           Guitar Hero: Aerosmith  Vicarious Visions
         1625   SpongeBob SquarePants: Battle for Bikini Bottom  Vicarious Visions
         1692  SpongeBob SquarePants: Revenge of the Flying D...  Vicarious Visions
         1715                          Tony Hawk's Pro Skater 2  Vicarious Visions
         2034                              Skylanders SWAP Force  Vicarious Visions
         2128                                        Band Hero  Vicarious Visions
         2294                      Marvel: Ultimate Alliance 2  Vicarious Visions
         2310                                      Spider-Man 3  Vicarious Visions

               Critic_Score
         234           86.0
         351           86.0
         383           71.0
         519            NaN
         873           69.0
         1032          78.0
         1181          72.0
         1205          74.0
         1206          89.0
         1421          75.0
```

```
1454          88.0
1478          83.0
1547          73.0
1625           NaN
1692          71.0
1715          95.0
2034          83.0
2128          79.0
2294          73.0
2310          50.0
```

We are now ready to look at sales based on developers. Doesn't come as much of a shock that Nintendo leads this category, since they primarily develop many of the games for their own systems.

```
In [33]: def sales_by_developer(region):
             '''
             Returns a dictionary of {developer : sales in millions of units}
             for "region"
             '''
             return {developer : df[df['Developer'] == developer][region].sum() for \
                     developer in top_20_devs}

         na_sales_by_developer = sales_by_developer('NA_Sales')
         eu_sales_by_developer = sales_by_developer('EU_Sales')
         jp_sales_by_developer = sales_by_developer('JP_Sales')

         developer_indices = range(len(na_sales_by_developer))
         developer_width = np.min(np.diff(developer_indices))/4

         fig, ax = plt.subplots(figsize = (10, 8))

         bar1 = ax.bar(developer_indices - developer_width,
                       na_sales_by_developer.values(),
                       developer_width, color = 'blue')
         bar2 = ax.bar(developer_indices,
                       eu_sales_by_developer.values(),
                       developer_width, color = 'red')
         bar3 = ax.bar(developer_indices + developer_width,
                       jp_sales_by_developer.values(),
                       developer_width, color = 'green')

         plt.xticks(rotation = 90)
         ax.set_xticks(developer_indices - developer_width/5)
         ax.set_xticklabels((na_sales_by_developer.keys()))

         plt.title('Units Sold for Top 20 Developers')
         plt.ylabel('Units Sold (millions)')
```

```
plt.legend(labels = ['North America', 'Europe', 'Japan'], loc = 'best')
plt.grid(which = 'major', axis = 'y')

for i in range(0, len(na_sales_by_developer), 2):
    plt.axvspan(i - 0.5, i + 0.5, facecolor = 'black', alpha = 0.1)

plt.show()
```



Next, we are going to look at game sales by rating to get an idea of what type of rating would sell the most. Current ratings are (in ascending order of age group) E, E10+, T, M, AO. Below, you'll also see a couple of "K-A" which was what the "E" rating was known as prior to 1998, and "RP", or Rating Pending. Since these are such low counts, I am going to not carry them through (along with the single AO rated game).

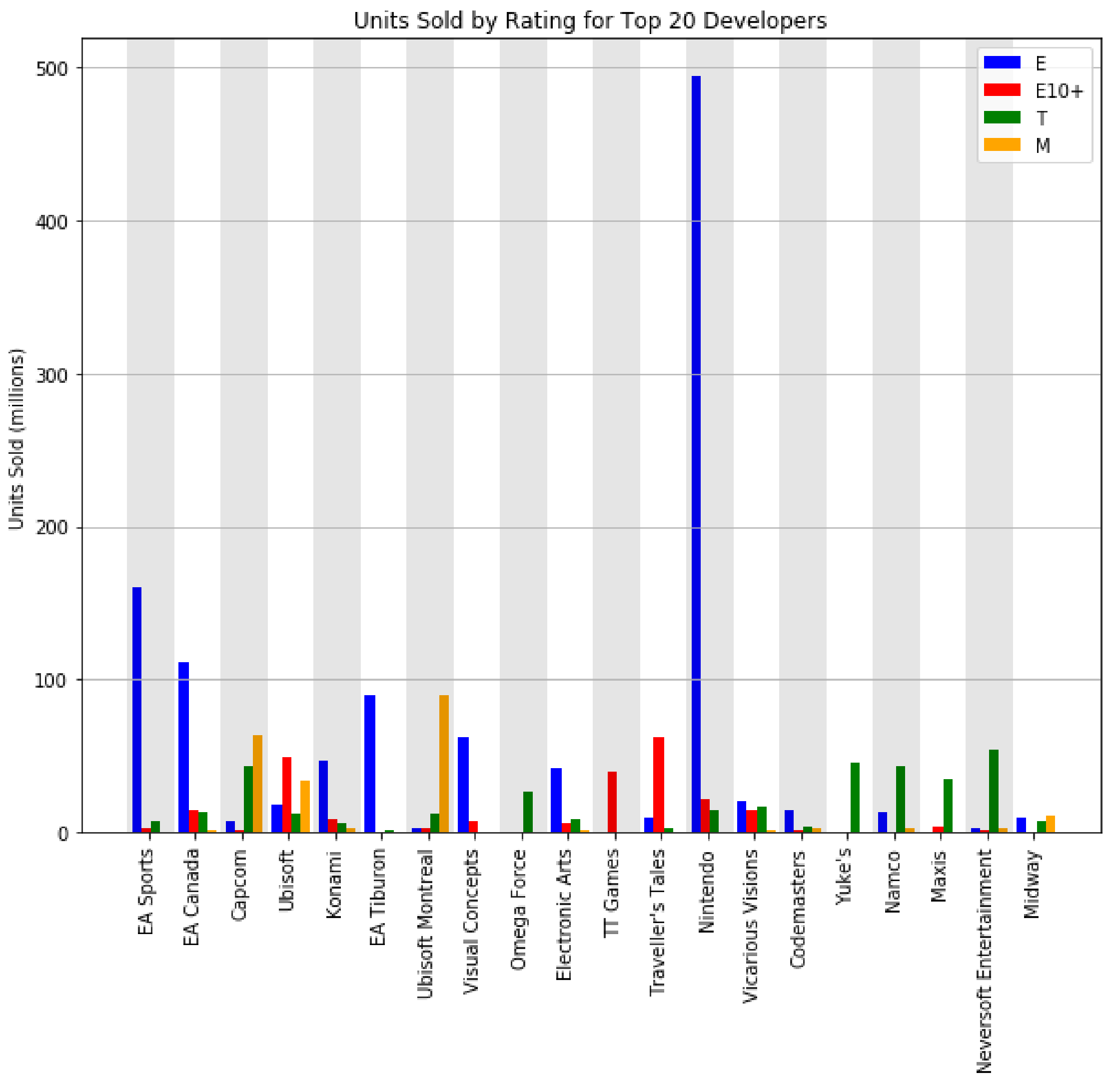
```
In [34]: df['Rating'] = df['Rating'].astype(str)
         ratings, ratings_counts = np.unique(df['Rating'], return_counts = True)
```

```
        for i in zip(ratings, ratings_counts):
            print(i)
```

```
('AO', 1)
('E', 2403)
('E10+', 1045)
('EC', 1)
('K-A', 3)
('M', 1521)
('RP', 3)
('T', 2575)
('nan', 6717)
```

```
In [35]: def sales_by_rating(rating, dev_list):
             '''
             Returns dictionary of {rating : sales in millions of units} for each
             "developer"
             '''
             return {developer : df[(df['Developer'] == developer) & \
                                    (df['Rating'] == rating)]['Global_Sales'].sum() \
                                     for developer in dev_list}

         e_sales_by_developer = sales_by_rating('E', top_20_devs)
         e10_sales_by_developer = sales_by_rating('E10+', top_20_devs)
         t_sales_by_developer = sales_by_rating('T', top_20_devs)
         m_sales_by_developer = sales_by_rating('M', top_20_devs)

         rating_indices = range(len(e_sales_by_developer))
         rating_width = np.min(np.diff(rating_indices))/5

         fig, ax = plt.subplots(figsize = (10, 8))

         bar1 = ax.bar(rating_indices - 1.5*rating_width, e_sales_by_developer.values(),
                       rating_width, color = 'blue')
         bar2 = ax.bar(rating_indices - 0.5*rating_width, e10_sales_by_developer.values(),
                       rating_width, color = 'red')
         bar3 = ax.bar(rating_indices + 0.5*rating_width, t_sales_by_developer.values(),
                       rating_width, color = 'green')
         bar4 = ax.bar(rating_indices + 1.5*rating_width, m_sales_by_developer.values(),
                       rating_width, color = 'orange')

         plt.xticks(rotation = 90)
         ax.set_xticks(developer_indices - developer_width/5)
         ax.set_xticklabels((na_sales_by_developer.keys()))

         plt.title('Units Sold by Rating for Top 20 Developers')
```

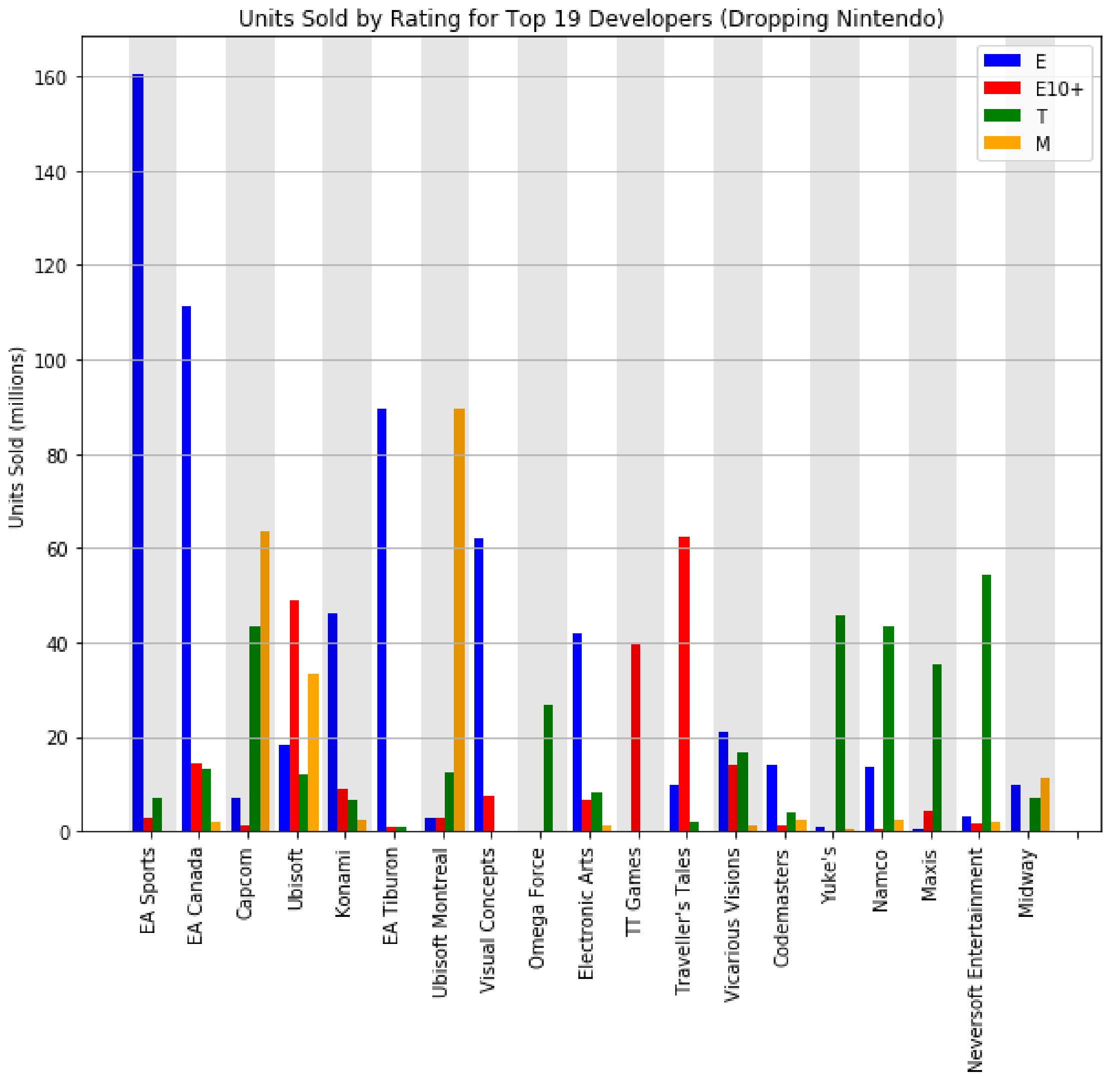
```
plt.ylabel('Units Sold (millions)')
plt.legend(labels = ['E', 'E10+', 'T', 'M'], loc = 'best')
plt.grid(which = 'major', axis = 'y')

for i in range(0, len(na_sales_by_developer), 2):
    plt.axvspan(i - 0.5, i + 0.5, facecolor = 'black', alpha = 0.1)

plt.show()
```



Well that's a pretty telling graph: Nintendo makes a **LOT** of rated-E games. However, it makes it pretty difficult to see any resolution on the other developers, so let's briefly disregard Nintendo and look at this again.

```
In [36]: top_19_devs = [dev for dev in top_20_devs if not dev == 'Nintendo']
```

```
e_sales_by_developer = sales_by_rating('E', top_19_devs)
e10_sales_by_developer = sales_by_rating('E10+', top_19_devs)
t_sales_by_developer = sales_by_rating('T', top_19_devs)
m_sales_by_developer = sales_by_rating('M', top_19_devs)

rating_indices = range(len(e_sales_by_developer))
rating_width = np.min(np.diff(rating_indices))/5

fig, ax = plt.subplots(figsize = (10, 8))

bar1 = ax.bar(rating_indices - 1.5*rating_width, e_sales_by_developer.values(),
              rating_width, color = 'blue')
bar2 = ax.bar(rating_indices - 0.5*rating_width, e10_sales_by_developer.values(),
              rating_width, color = 'red')
bar3 = ax.bar(rating_indices + 0.5*rating_width, t_sales_by_developer.values(),
              rating_width, color = 'green')
bar4 = ax.bar(rating_indices + 1.5*rating_width, m_sales_by_developer.values(),
              rating_width, color = 'orange')

plt.xticks(rotation = 90)
ax.set_xticks(developer_indices - developer_width/5)
ax.set_xticklabels(top_19_devs)

plt.title('Units Sold by Rating for Top 19 Developers (Dropping Nintendo)')
plt.ylabel('Units Sold (millions)')
plt.legend(labels = ['E', 'E10+', 'T', 'M'], loc = 'best')
plt.grid(which = 'major', axis = 'y')

for i in range(0, len(na_sales_by_developer), 2):
    plt.axvspan(i - 0.5, i + 0.5, facecolor = 'black', alpha = 0.1)

plt.show()
```



This has been a pretty exhaustive exploration of game sales over the last almost-40 years. However, what if we were interested in developing a game for the recent generation of consoles? Keep in mind, this dataset came out in 2016, so it doesn't include the Nintendo Switch, but will still give us a good picture.

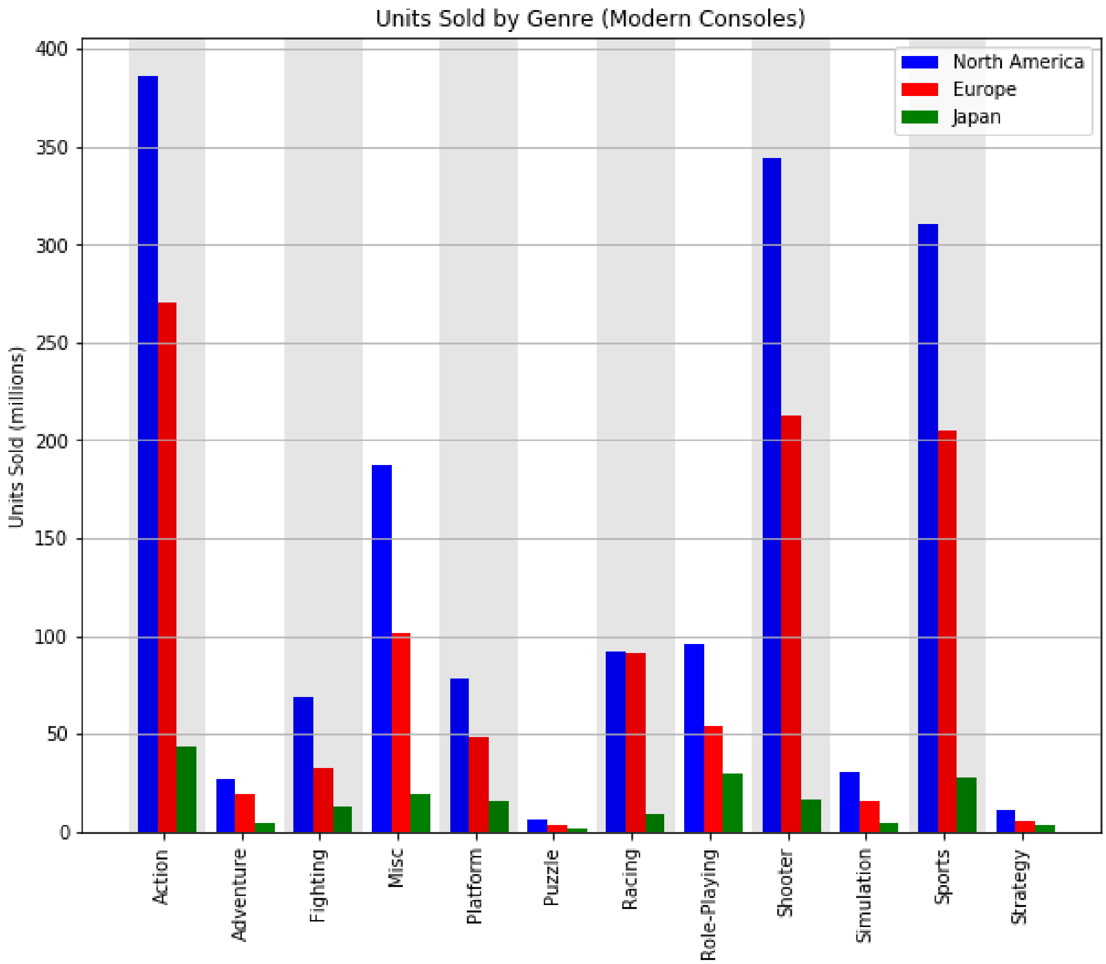
```
In [37]: modern = ['PS3', 'PS4', 'Wii', 'WiiU', 'X360', 'XOne']
         modern_df = pd.DataFrame()
         for platform in modern:
             modern_df = pd.concat([modern_df, df[df['Platform'] == platform]])
```

```
In [38]: modern_df.shape
```

```
Out[38]: (4058, 16)
```

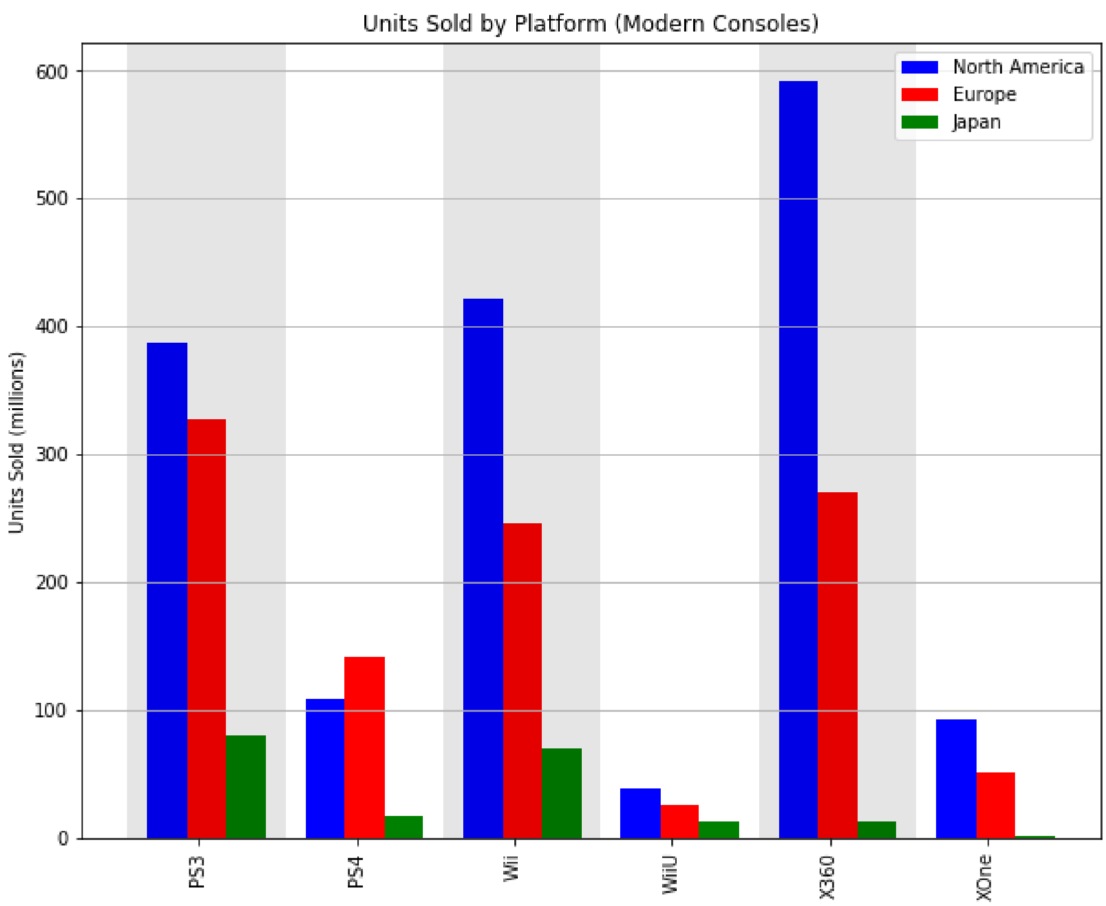
```
In [39]: na_sales_by_genre = sales_by_genre(modern_df, 'NA_Sales')
         eu_sales_by_genre = sales_by_genre(modern_df, 'EU_Sales')
```

```
jp_sales_by_genre = sales_by_genre(modern_df, 'JP_Sales')

genre_indices = range(len(na_sales_by_genre))
genre_width = np.min(np.diff(genre_indices))/4

fig, ax = plt.subplots(figsize = (10, 8))
bar1 = ax.bar(genre_indices - genre_width, na_sales_by_genre.values(),
              genre_width, color = 'blue')
bar2 = ax.bar(genre_indices, eu_sales_by_genre.values(),
              genre_width, color = 'red')
bar3 = ax.bar(genre_indices + genre_width, jp_sales_by_genre.values(),
              genre_width, color = 'green')

plt.xticks(rotation = 90)
ax.set_xticks(genre_indices - genre_width/5)
ax.set_xticklabels((na_sales_by_genre.keys()))

plt.title('Units Sold by Genre (Modern Consoles)')
plt.ylabel('Units Sold (millions)')
plt.legend(labels = ['North America', 'Europe', 'Japan'], loc = 'best')
plt.grid(which = 'major', axis = 'y')

for i in range(0, len(na_sales_by_genre), 2):
    plt.axvspan(i - 0.5, i + 0.5, facecolor = 'black', alpha = 0.1)

plt.show()
```




```
In [40]: def sales_by_platform(region):
             '''
             Returns a dictionary of {platform : sales in millions of units}
             for "region"
             '''
             return {platform : modern_df[modern_df['Platform'] == platform][region].\
                     sum() for platform in modern}

         na_sales_by_platform = sales_by_platform('NA_Sales')
         eu_sales_by_platform = sales_by_platform('EU_Sales')
         jp_sales_by_platform = sales_by_platform('JP_Sales')

         platform_indices = range(len(modern))
         platform_width = np.min(np.diff(platform_indices))/4

         fig, ax = plt.subplots(figsize = (10, 8))
```

```
bar1 = ax.bar(platform_indices - platform_width, na_sales_by_platform.values(),
              platform_width, color = 'blue')
bar2 = ax.bar(platform_indices, eu_sales_by_platform.values(),
              platform_width, color = 'red')
bar3 = ax.bar(platform_indices + platform_width, jp_sales_by_platform.values(),
              platform_width, color = 'green')

plt.xticks(rotation = 90)
ax.set_xticks(platform_indices - platform_width/5)
ax.set_xticklabels((na_sales_by_platform.keys()))

plt.title('Units Sold by Platform (Modern Consoles)')
plt.ylabel('Units Sold (millions)')
plt.legend(labels = ['North America', 'Europe', 'Japan'], loc = 'best')
plt.grid(which = 'major', axis = 'y')

for i in range(0, len(na_sales_by_platform), 2):
    plt.axvspan(i - 0.5, i + 0.5, facecolor = 'black', alpha = 0.1)

plt.show()
```



So there you have it - of the big-name developers, the ones having the most success (by far!) are selling **Action**, **Shooter**, or **Sports** games. Keeping in mind this dataset is at least 2 years old, there have been a lot of data generated for the current generation of consoles (PS4, Xbox One, Nintendo Switch) that is not captured here. Last generation, the **Xbox 360** was a clear winner. However, it's looking like the **PS4** is taking the crown so far in this generation!