Capstone Project

TASK: Import any libraries you think you will use:

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

Part Two: Exploring Fandango Displayed Scores versus True User Ratings

Let's first explore the Fandango ratings to see if our analysis agrees with the article's conclusion.

TASK: Run the cell below to read in the fandango_scrape.csv file

In [2]:
```python
fandango = pd.read_csv("fandango_scrape.csv")
```

TASK: Explore the DataFrame Properties and Head.

In [3]:
```python
fandango.head()
```

Out [3]:

	FILM	STARS	RATING	VOTES
0	Fifty Shades of Grey (2015)	4.0	3.9	34846
1	Jurassic World (2015)	4.5	4.5	34390
2	American Sniper (2015)	5.0	4.8	34085
3	Furious 7 (2015)	5.0	4.8	33538
4	Inside Out (2015)	4.5	4.5	15749

In [4]:
```python
fandango.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 504 entries, 0 to 503
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   FILM    504 non-null    object 
 1   STARS   504 non-null    float64
 2   RATING  504 non-null    float64
 3   VOTES   504 non-null    int64  
dtypes: float64(2), int64(1), object(1)
memory usage: 15.9+ KB
```

In [5]:
```python
fandango.describe()
```

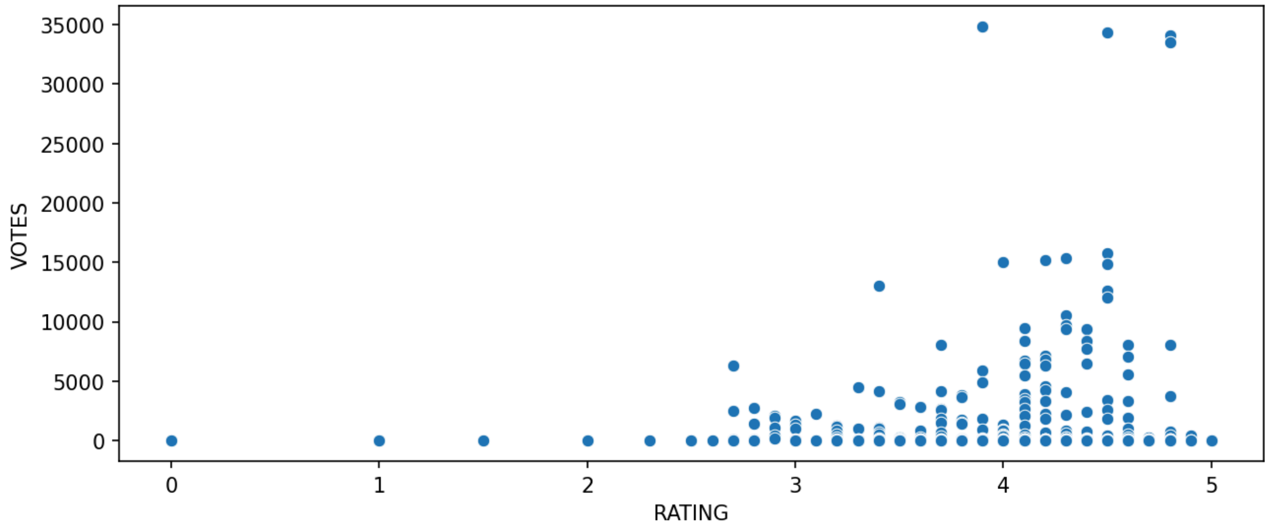
Out [5]:

	STARS	RATING	VOTES
count	504.000000	504.000000	504.000000
mean	3.558532	3.375794	1147.863095
std	1.563133	1.491223	3830.583136
min	0.000000	0.000000	0.000000
25%	3.500000	3.100000	3.000000
50%	4.000000	3.800000	18.500000
75%	4.500000	4.300000	189.750000
max	5.000000	5.000000	34846.000000

TASK: Let's explore the relationship between popularity of a film and its rating. Create a scatterplot showing the relationship between rating and votes. Feel free to edit visual styling to your preference.

In [6]:
```python
plt.figure(figsize=(10,4), dpi=150)
sns.scatterplot(data=fandango, x= 'RATING', y='VOTES')
```

Out [6]: <Axes: xlabel='RATING', ylabel='VOTES'>



TASK: Assuming that every row in the FILM title column has the same format:

Film Title Name (Year)

Create a new column that is able to strip the year from the title strings and set this new column as YEAR

In [7]:
```
fandango['YEAR'] = fandango['FILM'].apply(lambda title:title.split('(')[-1])
```

TASK: How many movies are in the Fandango DataFrame per year?

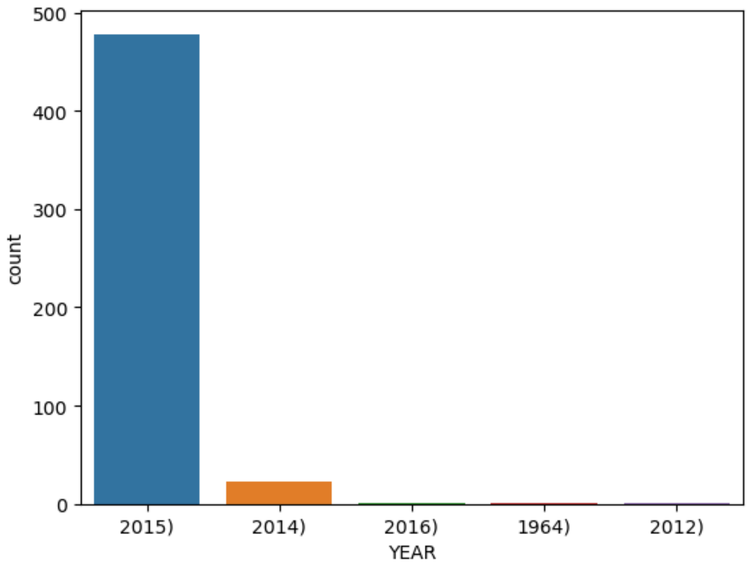
In [8]:
```
fandango['YEAR'].value_counts()
```

Out [8]:
```
YEAR
2015)    478
2014)     23
2016)      1
1964)      1
2012)      1
Name: count, dtype: int64
```

TASK: Visualize the count of movies per year with a plot:

In [9]:
```
sns.countplot(data=fandango, x='YEAR')
```

Out [9]: <Axes: xlabel='YEAR', ylabel='count'>



TASK: What are the 10 movies with the highest number of votes?

In [10]:
```
fandango.nlargest(10, 'VOTES')
```

Out [10]:

	FILM	STARS	RATING	VOTES	YEAR
0	Fifty Shades of Grey (2015)	4.0	3.9	34846	2015)
1	Jurassic World (2015)	4.5	4.5	34390	2015)
2	American Sniper (2015)	5.0	4.8	34085	2015)
3	Furious 7 (2015)	5.0	4.8	33538	2015)
4	Inside Out (2015)	4.5	4.5	15749	2015)

	FILM	STARS	RATING	VOTES	YEAR
5	The Hobbit: The Battle of the Five Armies (2014)	4.5	4.3	15337	2014)
6	Kingsman: The Secret Service (2015)	4.5	4.2	15205	2015)
7	Minions (2015)	4.0	4.0	14998	2015)
8	Avengers: Age of Ultron (2015)	5.0	4.5	14846	2015)
9	Into the Woods (2014)	3.5	3.4	13055	2014)

TASK: How many movies have zero votes?

In [11]:
```
no_votes = fandango['VOTES']==0
no_votes.sum()
```

Out [11]: 69

TASK: Create DataFrame of only reviewed films by removing any films that have zero votes.

In [12]:
```
fan_reviewed = fandango[fandango['VOTES']>0]
```

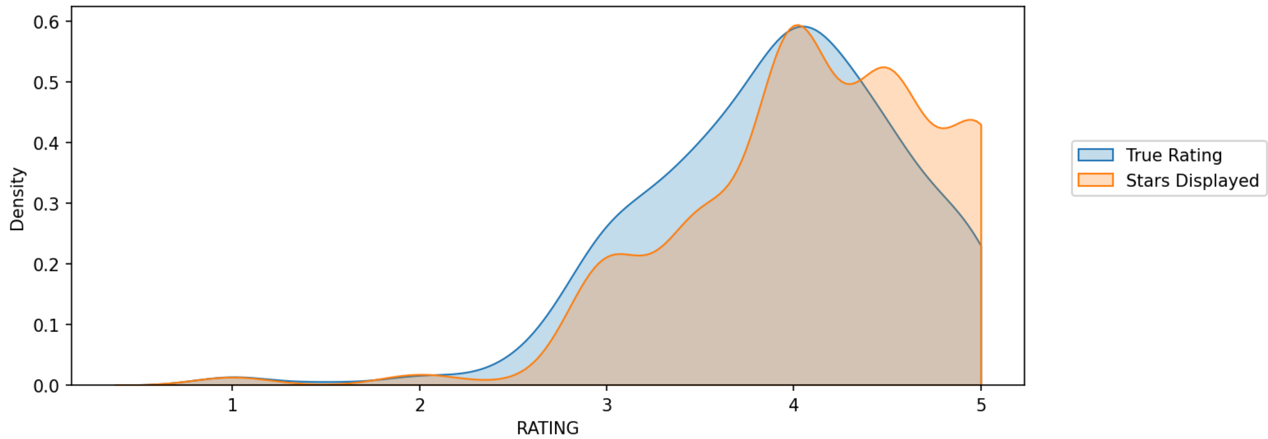
As noted in the article, due to HTML and star rating displays, the true user rating may be slightly different than the rating shown to a user. Let's visualize this difference in distributions.

TASK: Create a KDE plot (or multiple kdeplots) that displays the distribution of ratings that are displayed (STARS) versus what the true rating was from votes (RATING). Clip the KDEs to 0-5.

In [13]:
```
plt.figure(figsize=(10,4), dpi=150)
sns.kdeplot(data=fan_reviewed, x='RATING', clip=[0,5], fill= True, label='True Rating')
sns.kdeplot(data=fan_reviewed, x='STARS', clip=[0,5], fill=True, label='Stars Displayed')

plt.legend(loc=(1.05,0.5))
```

Out [13]: <matplotlib.legend.Legend at 0x2659e1032d0>



TASK: Let's now actually quantify this discrepancy. Create a new column of the different between STARS displayed versus true RATING. Calculate this difference with STARS-RATING and round these differences to the nearest decimal point.

In [14]:
```
fan_reviewed['STARS_DIFF'] = fan_reviewed['STARS'] - fan_reviewed['RATING']
fan_reviewed['STARS_DIFF'] = fan_reviewed['STARS_DIFF'].round(2)
```

```
C:\Users\vigne\AppData\Local\Temp\ipykernel_17808\3768807957.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  fan_reviewed['STARS_DIFF'] = fan_reviewed['STARS'] - fan_reviewed['RATING']
C:\Users\vigne\AppData\Local\Temp\ipykernel_17808\3768807957.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  fan_reviewed['STARS_DIFF'] = fan_reviewed['STARS_DIFF'].round(2)
```

In [15]:
```
fan_reviewed
```

Out [15]:

	FILM	STARS	RATING	VOTES	YEAR	STARS_DIFF
0	Fifty Shades of Grey (2015)	4.0	3.9	34846	2015)	0.1
1	Jurassic World (2015)	4.5	4.5	34390	2015)	0.0
2	American Sniper (2015)	5.0	4.8	34085	2015)	0.2
3	Furious 7 (2015)	5.0	4.8	33538	2015)	0.2

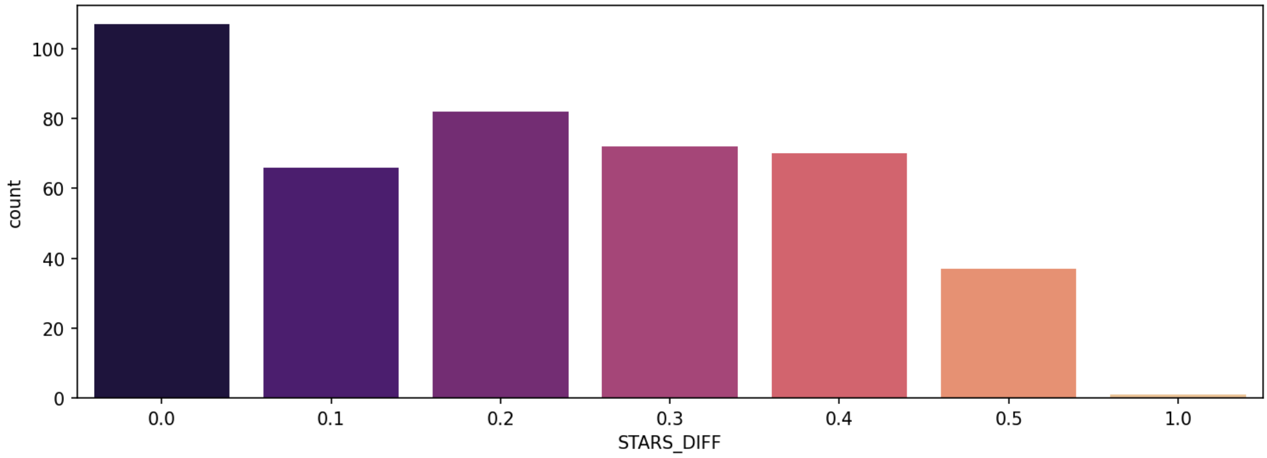
	FILM	STARS	RATING	VOTES	YEAR	STARS_DIFF
4	Inside Out (2015)	4.5	4.5	15749	2015)	0.0
...	...	...	...	...	...	...
430	That Sugar Film (2015)	5.0	5.0	1	2015)	0.0
431	The Intern (2015)	5.0	5.0	1	2015)	0.0
432	The Park Bench (2015)	5.0	5.0	1	2015)	0.0
433	The Wanted 18 (2015)	5.0	5.0	1	2015)	0.0
434	Z For Zachariah (2015)	5.0	5.0	1	2015)	0.0

435 rows × 6 columns

TASK: Create a count plot to display the number of times a certain difference occurs:

In [16]:
```
plt.figure(figsize=(12,4), dpi=150)
sns.countplot(data=fan_reviewed, x='STARS_DIFF', palette='magma')
```

Out [16]: <Axes: xlabel='STARS_DIFF', ylabel='count'>



TASK: We can see from the plot that one movie was displaying over a 1 star difference than its true rating! What movie had this close to 1 star differential?

In [17]:
```
fan_reviewed[fan_reviewed['STARS_DIFF'] == 1]
```

Out [17]:

	FILM	STARS	RATING	VOTES	YEAR	STARS_DIFF
381	Turbo Kid (2015)	5.0	4.0	2	2015)	1.0

Part Three: Comparison of Fandango Ratings to Other Sites

Let's now compare the scores from Fandango to other movies sites and see how they compare.

TASK: Read in the "all_sites_scores.csv" file by running the cell below

In [18]:
```
all_sites = pd.read_csv('all_sites_scores.csv')
```

TASK: Explore the DataFrame columns, info, description.

In [19]:
```
all_sites.head()
```

Out [19]:

	FILM	RottenTomatoes	RottenTomatoes_User	Metacritic	Metacritic_User	IMDB	Metacritic_user_vote_count	IMDB_user_vote_count
0	Avengers: Age of Ultron (2015)	74	86	66	7.1	7.8	1330	271107
1	Cinderella (2015)	85	80	67	7.5	7.1	249	65709
2	Ant-Man (2015)	80	90	64	8.1	7.8	627	103660
3	Do You Believe? (2015)	18	84	22	4.7	5.4	31	3136
4	Hot Tub Time Machine 2 (2015)	14	28	29	3.4	5.1	88	19560

In [20]:
```python
all_sites.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146 entries, 0 to 145
Data columns (total 8 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   FILM                        146 non-null    object 
 1   RottenTomatoes              146 non-null    int64  
 2   RottenTomatoes_User         146 non-null    int64  
 3   Metacritic                  146 non-null    int64  
 4   Metacritic_User             146 non-null    float64
 5   IMDB                        146 non-null    float64
 6   Metacritic_user_vote_count  146 non-null    int64  
 7   IMDB_user_vote_count        146 non-null    int64  
dtypes: float64(2), int64(5), object(1)
memory usage: 9.3+ KB
```

In [21]:
```python
all_sites.describe()
```

Out [21]:

	RottenTomatoes	RottenTomatoes_User	Metacritic	Metacritic_User	IMDB	Metacritic_user_vote_count	IMDB_user_vote_count
count	146.000000	146.000000	146.000000	146.000000	146.000000	146.000000	146.000000
mean	60.849315	63.876712	58.808219	6.519178	6.736986	185.705479	42846.205479
std	30.168799	20.024430	19.517389	1.510712	0.958736	316.606515	67406.509171
min	5.000000	20.000000	13.000000	2.400000	4.000000	4.000000	243.000000
25%	31.250000	50.000000	43.500000	5.700000	6.300000	33.250000	5627.000000
50%	63.500000	66.500000	59.000000	6.850000	6.900000	72.500000	19103.000000
75%	89.000000	81.000000	75.000000	7.500000	7.400000	168.500000	45185.750000
max	100.000000	94.000000	94.000000	9.600000	8.600000	2375.000000	334164.000000

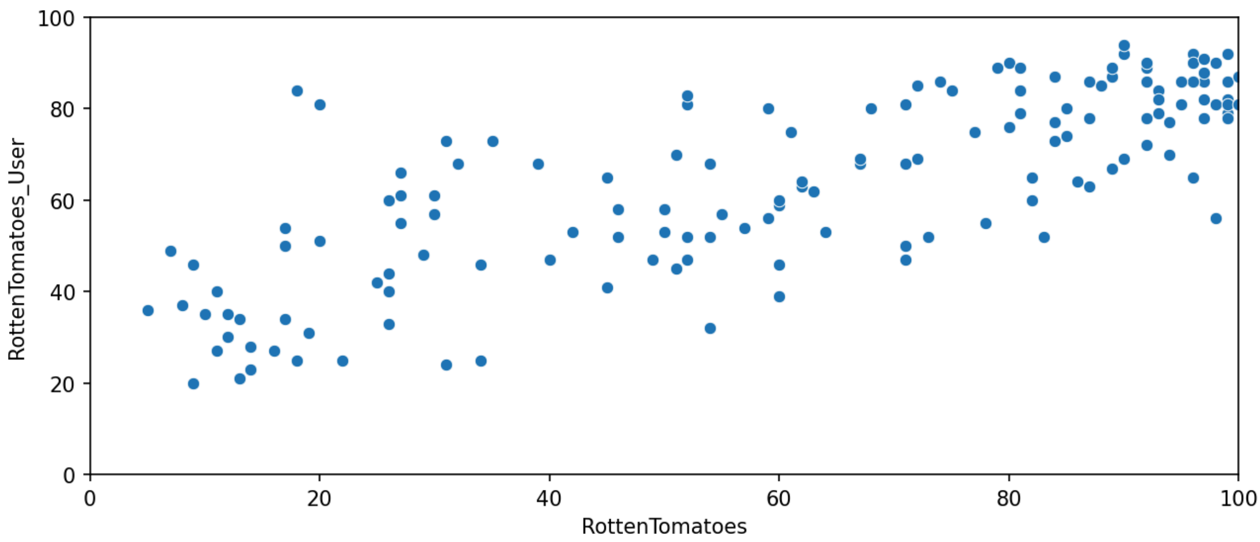
Rotten Tomatoes

Let's first take a look at Rotten Tomatoes. RT has two sets of reviews, their critics reviews (ratings published by official critics) and user reviews.

TASK: Create a scatterplot exploring the relationship between RT Critic reviews and RT User reviews.

In [23]:
```python
plt.figure(figsize=(10,4), dpi=150)
sns.scatterplot(data=all_sites, x='RottenTomatoes', y='RottenTomatoes_User')
plt.xlim(0,100)
plt.ylim(0,100)
```

Out [23]: (0.0, 100.0)



Let's quantify this difference by comparing the critics ratings and the RT User ratings. We will calculate this with RottenTomatoes-RottenTomatoes_User. Note: Rotten_Diff here is Critics - User Score. So values closer to 0 means aggrement between Critics and Users. Larger positive values means critics rated much higher than users. Larger negative values means users rated much higher than critics.

TASK: Create a new column based off the difference between critics ratings and users ratings for Rotten Tomatoes. Calculate this with RottenTomatoes-RottenTomatoes_User

In [26]:
```python
all_sites['Rotten_Diff'] = all_sites['RottenTomatoes'] - all_sites['RottenTomatoes_User']
```

Let's now compare the overall mean difference. Since we're dealing with differences that could be negative or positive, first take the absolute value of all the differences, then take the mean. This would report back on average to absolute difference between the critics rating versus the user rating.

TASK: Calculate the Mean Absolute Difference between RT scores and RT User scores as described above.

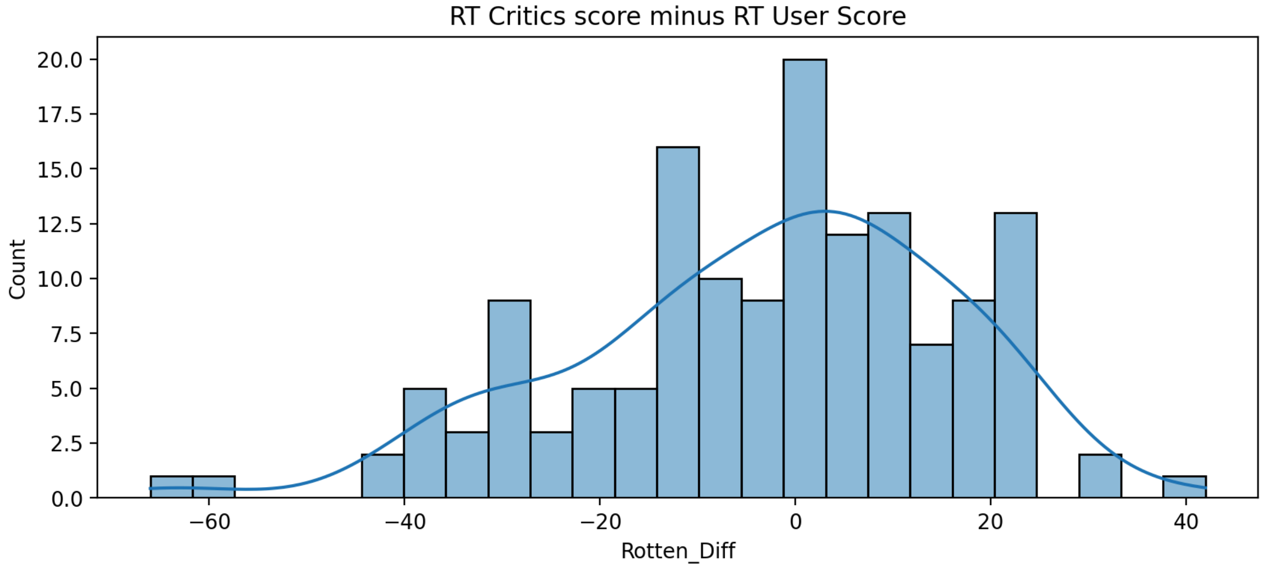
In [27]:
```python
all_sites['Rotten_Diff'].apply(abs).mean()
```

Out [27]: 15.095890410958905

TASK: Plot the distribution of the differences between RT Critics Score and RT User Score. There should be negative values in this distribution plot. Feel free to use KDE or Histograms to display this distribution.

In [28]:
```python
plt.figure(figsize=(10,4), dpi=200)
sns.histplot(data=all_sites, x='Rotten_Diff', kde=True, bins=25)
plt.title("RT Critics score minus RT User Score")
```

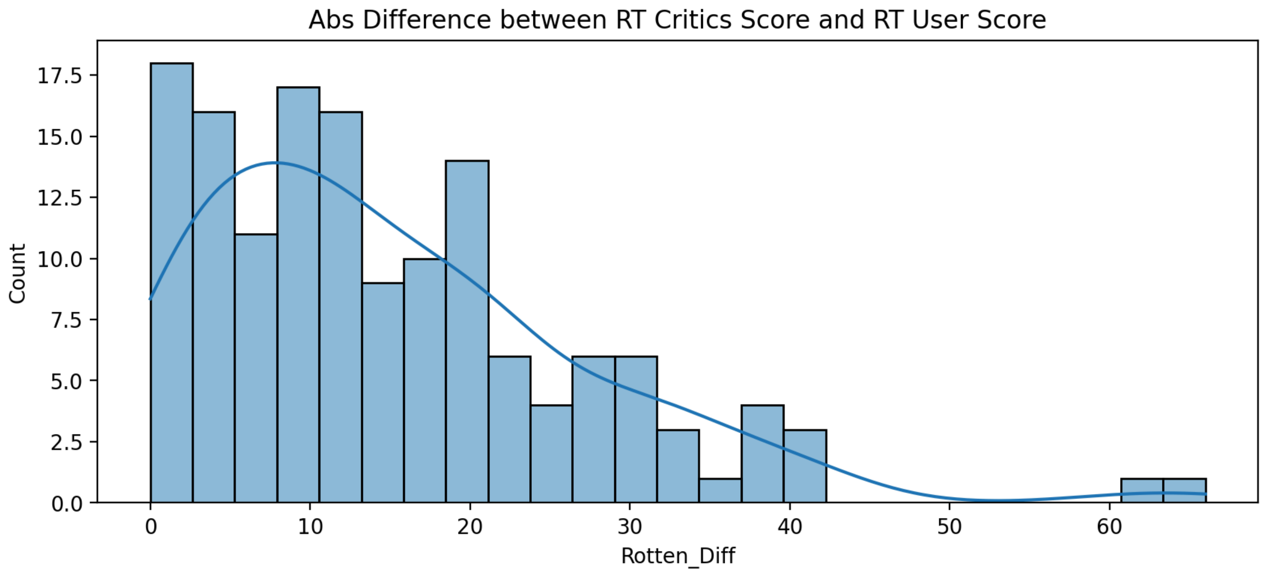
Out [28]: Text(0.5, 1.0, 'RT Critics score minus RT User Score')



TASK: Now create a distribution showing the absolute value difference between Critics and Users on Rotten Tomatoes.

In [31]:
```python
plt.figure(figsize=(10,4), dpi=200)
sns.histplot(x=all_sites['Rotten_Diff'].apply(abs), bins=25, kde=True)
plt.title("Abs Difference between RT Critics Score and RT User Score")
```

Out [31]: Text(0.5, 1.0, 'Abs Difference between RT Critics Score and RT User Score')



Let's find out which movies are causing the largest differences. First, show the top 5 movies with the largest negative difference between Users and RT critics. Since we calculated the difference as Critics Rating - Users Rating, then large negative values imply the users rated the movie much higher on average than the critics did.

TASK: What are the top 5 movies users rated higher than critics on average:

In [32]:
```python
print("Users love but critics hate")
all_sites.nsmallest(5, 'Rotten_Diff')[['FILM', 'Rotten_Diff']]
```

Users love but critics hate

Out [32]:

	FILM	Rotten_Diff
3	Do You Believe? (2015)	-66
85	Little Boy (2015)	-61
105	Hitman: Agent 47 (2015)	-42
134	The Longest Ride (2015)	-42

	FILM	Rotten_Diff
125	The Wedding Ringer (2015)	-39

TASK: Now show the top 5 movies critics scores higher than users on average.

In [33]:
```
print("Critics love, but Users Hate")
all_sites.nlargest(5, 'Rotten_Diff')[['FILM', 'Rotten_Diff']]
```

```
Critics love, but Users Hate
```

Out [33]:

	FILM	Rotten_Diff
69	Mr. Turner (2014)	42
112	It Follows (2015)	31
115	While We're Young (2015)	31
37	Welcome to Me (2015)	24
40	I'll See You In My Dreams (2015)	24

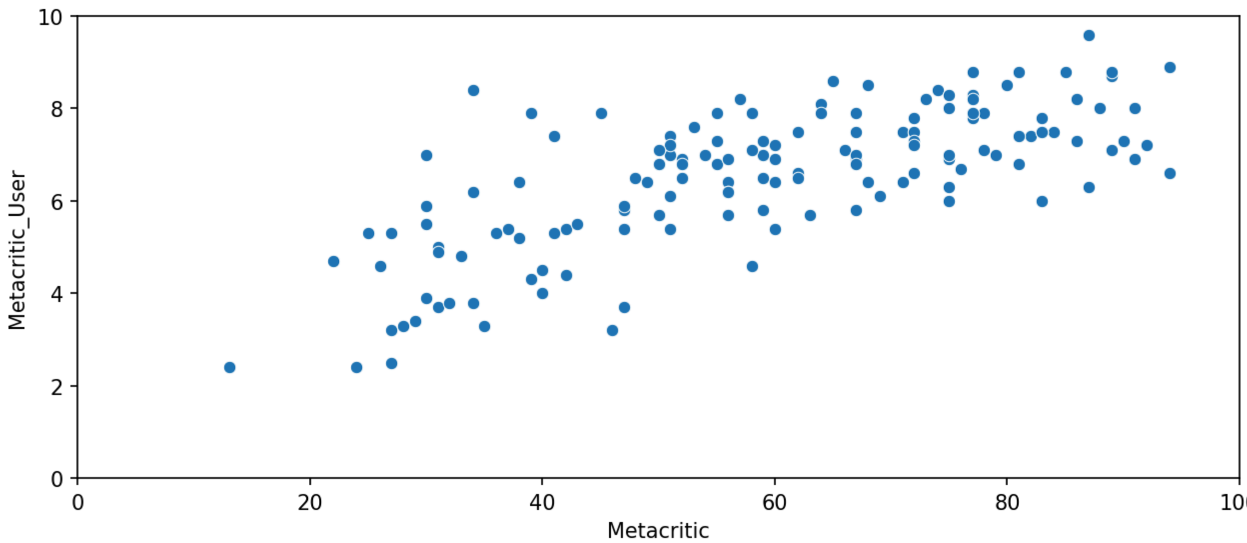
MetaCritic

Now let's take a quick look at the ratings from MetaCritic. Metacritic also shows an average user rating versus their official displayed rating.

TASK: Display a scatterplot of the Metacritic Rating versus the Metacritic User rating.

In [35]:
```
plt.figure(figsize=(10,4), dpi=150)
sns.scatterplot(data=all_sites, x='Metacritic', y='Metacritic_User')
plt.xlim(0,100)
plt.ylim(0,10)
```

Out [35]: (0.0, 10.0)



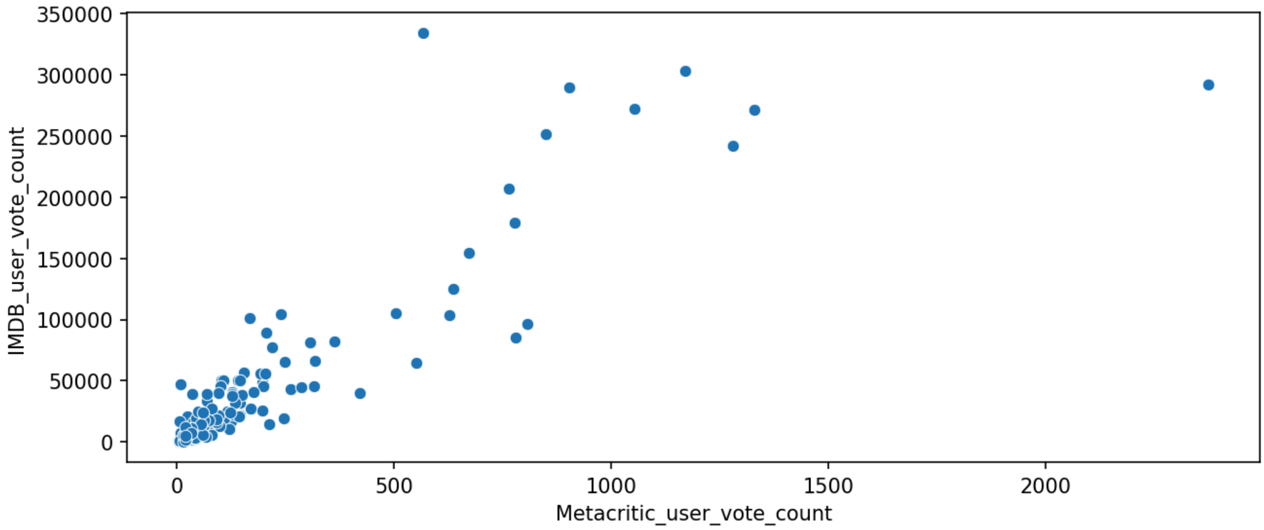
IMDB

Finally let's explore IMDB. Notice that both Metacritic and IMDB report back vote counts. Let's analyze the most popular movies.

TASK: Create a scatterplot for the relationship between vote counts on MetaCritic versus vote counts on IMDB.

In [36]:
```
plt.figure(figsize=(10,4), dpi=150)
sns.scatterplot(data=all_sites, x='Metacritic_user_vote_count', y='IMDB_user_vote_count')
```

Out [36]: <Axes: xlabel='Metacritic_user_vote_count', ylabel='IMDB_user_vote_count'>



Notice there are two outliers here. The movie with the highest vote count on IMDB only has about 500 Metacritic ratings. What is this movie?

TASK: What movie has the highest IMDB user vote count?

In [37]:
```
all_sites.nlargest(1, 'IMDB_user_vote_count')
```

Out [37]:

	FILM	RottenTomatoes	RottenTomatoes_User	Metacritic	Metacritic_User	IMDB	Metacritic_user_vote_count	IMDB_user_vote_count	Rott
14	The Imitation Game (2014)	90	92	73	8.2	8.1	566	334164	-2

TASK: What movie has the highest Metacritic User Vote count?

In [38]:
```
all_sites.nlargest(1, 'Metacritic_user_vote_count')
```

Out [38]:

	FILM	RottenTomatoes	RottenTomatoes_User	Metacritic	Metacritic_User	IMDB	Metacritic_user_vote_count	IMDB_user_vote_count	Rottten_
88	Mad Max: Fury Road (2015)	97	88	89	8.7	8.3	2375	292023	9

Fandago Scores vs. All Sites

Finally let's begin to explore whether or not Fandango artificially displays higher ratings than warranted to boost ticket sales.

TASK: Combine the Fandango Table with the All Sites table. Not every movie in the Fandango table is in the All Sites table, since some Fandango movies have very little or no reviews. We only want to compare movies that are in both DataFrames, so do an inner merge to merge together both DataFrames based on the FILM columns.

In [39]:
```
df = pd.merge(fandango,all_sites, on='FILM', how='inner')
```

In [40]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145 entries, 0 to 144
Data columns (total 14 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   FILM                        145 non-null    object
 1   STARS                       145 non-null    float64
 2   RATING                      145 non-null    float64
 3   VOTES                       145 non-null    int64
 4   YEAR                        145 non-null    object
 5   RottenTomatoes              145 non-null    int64
 6   RottenTomatoes_User         145 non-null    int64
 7   Metacritic                  145 non-null    int64
 8   Metacritic_User             145 non-null    float64
 9   IMDB                        145 non-null    float64
 10  Metacritic_user_vote_count  145 non-null    int64
 11  IMDB_user_vote_count        145 non-null    int64
 12  Rottten_Diff                145 non-null    int64
 13  Rotten_Diff                 145 non-null    int64
dtypes: float64(4), int64(8), object(2)
memory usage: 16.0+ KB
```

In [41]:
```
df.head()
```

Out [41]:

	FILM	STARS	RATING	VOTES	YEAR	RottenTomatoes	RottenTomatoes_User	Metacritic	Metacritic_User	IMDB	Metacritic_user_vote
0	Fifty Shades	4.0	3.9	34846	2015)	25	42	46	3.2	4.2	778

	FILM	STARS	RATING	VOTES	YEAR	RottenTomatoes	RottenTomatoes_User	Metacritic	Metacritic_User	IMDB	Metacritic_user_vote
	of Grey (2015)										
1	Jurassic World (2015)	4.5	4.5	34390	2015)	71	81	59	7.0	7.3	1281
2	American Sniper (2015)	5.0	4.8	34085	2015)	72	85	72	6.6	7.4	850
3	Furious 7 (2015)	5.0	4.8	33538	2015)	81	84	67	6.8	7.4	764
4	Inside Out (2015)	4.5	4.5	15749	2015)	98	90	94	8.9	8.6	807

Normalize columns to Fandango STARS and RATINGS 0-5

Notice that RT,Metacritic, and IMDB don't use a score between 0-5 stars like Fandango does. In order to do a fair comparison, we need to normalize these values so they all fall between 0-5 stars and the relationship between reviews stays the same.

TASK: Create new normalized columns for all ratings so they match up within the 0-5 star range shown on Fandango. There are many ways to do this.

Hint link: https://stackoverflow.com/questions/26414913/normalize-columns-of-pandas-data-frame

Easier Hint:

Keep in mind, a simple way to convert ratings:

- 100/20 = 5
- 10/2 = 5

In [42]:
```
df['RT_Norm'] = np.round(df['RottenTomatoes']/20, 1)
df['RTU_Norm'] = np.round(df['RottenTomatoes_User']/20, 1)
```

In [47]:
```
df['Meta_Norm'] = np.round(df['Metacritic']/20, 1)
df['Meta_U_Norm'] = np.round(df['Metacritic_User']/2, 1)
```

In [44]:
```
df['IMDB_Norm'] = np.round(df['IMDB']/2, 1)
```

In [48]:
```
df.head()
```

Out [48]:

	FILM	STARS	RATING	VOTES	YEAR	RottenTomatoes	RottenTomatoes_User	Metacritic	Metacritic_User	IMDB	Metacritic_user_vote
0	Fifty Shades of Grey (2015)	4.0	3.9	34846	2015)	25	42	46	3.2	4.2	778
1	Jurassic World (2015)	4.5	4.5	34390	2015)	71	81	59	7.0	7.3	1281
2	American Sniper (2015)	5.0	4.8	34085	2015)	72	85	72	6.6	7.4	850
3	Furious 7 (2015)	5.0	4.8	33538	2015)	81	84	67	6.8	7.4	764
4	Inside Out (2015)	4.5	4.5	15749	2015)	98	90	94	8.9	8.6	807

TASK: Now create a norm_scores DataFrame that only contains the normalizes ratings. Include both STARS and RATING from the original Fandango table.

In [49]:
```
norm_scores = df[['STARS', 'RATING', 'RT_Norm', 'Meta_Norm', 'Meta_U_Norm', "IMDB_Norm"]]
```

In [50]:
```
norm_scores.head()
```

Out [50]:

	STARS	RATING	RT_Norm	Meta_Norm	Meta_U_Norm	IMDB_Norm
0	4.0	3.9	1.2	2.3	1.6	2.1
1	4.5	4.5	3.6	3.0	3.5	3.6
2	5.0	4.8	3.6	3.6	3.3	3.7
3	5.0	4.8	4.0	3.4	3.4	3.7
4	4.5	4.5	4.9	4.7	4.4	4.3

Comparing Distribution of Scores Across Sites

Now the moment of truth! Does Fandango display abnormally high ratings? We already know it pushs displayed RATING higher than STARS, but are the ratings themselves higher than average?

TASK: Create a plot comparing the distributions of normalized ratings across all sites. There are many ways to do this, but explore the Seaborn KDEplot docs for some simple ways to quickly show this. Don't worry if your plot format does not look exactly the same as ours, as long as the differences in distribution are clear.

Quick Note if you have issues moving the legend for a seaborn kdeplot: https://github.com/mwaskom/seaborn/issues/2280

In [51]:
```
def move_legend(ax, new_loc, **kws):
    old_legend = ax.legend_
    handles - old_legend.legendHandles
    labels = [t.get_text() for t in old_legend.get_texts()]
    title = old_legend.get_title().get_text()
    ax.legend(handles, labels, loc=new_loc, title=title, **kws)
```

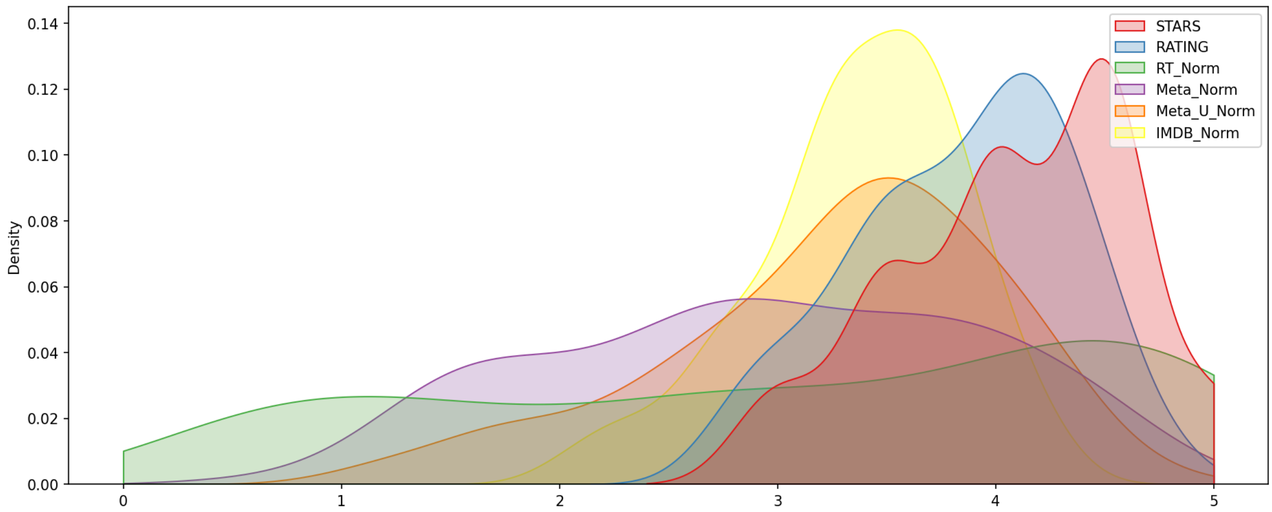
In [52]:
```
fig, ax = plt.subplots(figsize=(15,6), dpi=150)
sns.kdeplot(data=norm_scores, clip=[0,5], shade=True, palette='Set1', ax=ax)
move_legend(ax, "upper left")
```

```
C:\Users\vigne\AppData\Local\Temp\ipykernel_17808\3564412342.py:2: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=norm_scores, clip=[0,5], shade=True, palette='Set1', ax=ax)
```

```
---------------------------------------------------------------------------NameError                                 Traceback (most recent call las
      1 fig, ax = plt.subplots(figsize=(15,6), dpi=150)
      2 sns.kdeplot(data=norm_scores, clip=[0,5], shade=True, palette='Set1', ax=ax)
----> 3 move_legend(ax, "upper left")
Cell In[51], line 3, in move_legend(ax, new_loc, **kws)
      1 def move_legend(ax, new_loc, **kws):
      2     old_legend = ax.legend_
----> 3     handles - old_legend.legendHandles
      4     labels = [t.get_text() for t in old_legend.get_texts()]
      5     title = old_legend.get_title().get_text()
NameError: name 'handles' is not defined
```



Clearly Fandango has an uneven distribution. We can also see that RT critics have the most uniform distribution. Let's directly compare these two.

TASK: Create a KDE plot that compare the distribution of RT critic ratings against the STARS displayed by Fandango.

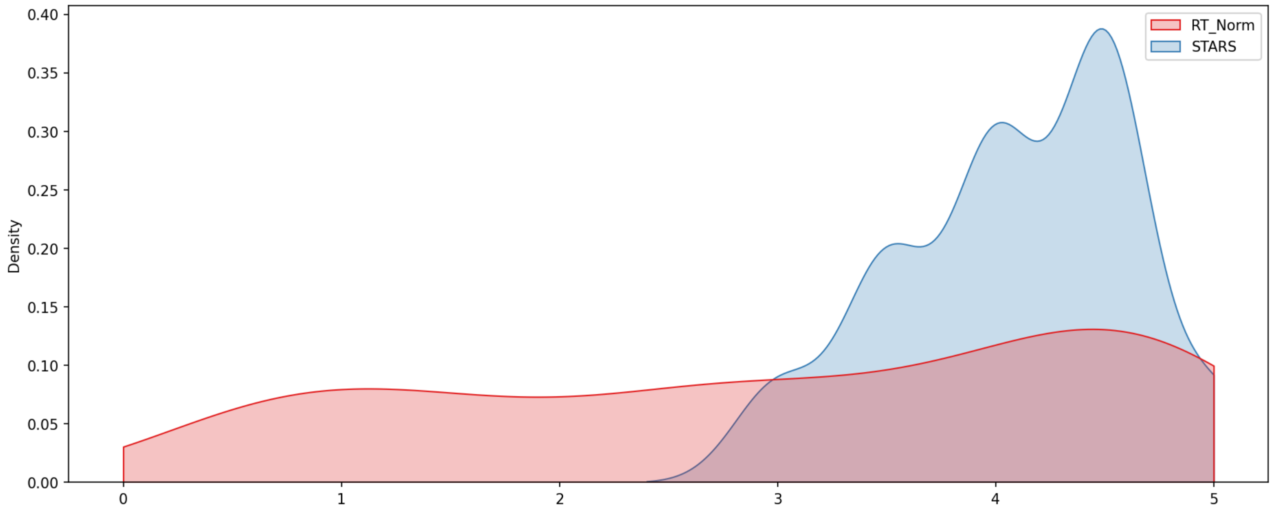
In [53]:
```
fig, ax = plt.subplots(figsize=(15,6), dpi=150)
sns.kdeplot(data=norm_scores[['RT_Norm', 'STARS']], clip=[0,5], shade=True, palette='Set1', ax=ax)
move_legend(ax, "upper left")
```

```
C:\Users\vigne\AppData\Local\Temp\ipykernel_17808\4272579469.py:2: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=norm_scores[['RT_Norm', 'STARS']], clip=[0,5], shade=True, palette='Set1', ax=ax)
```

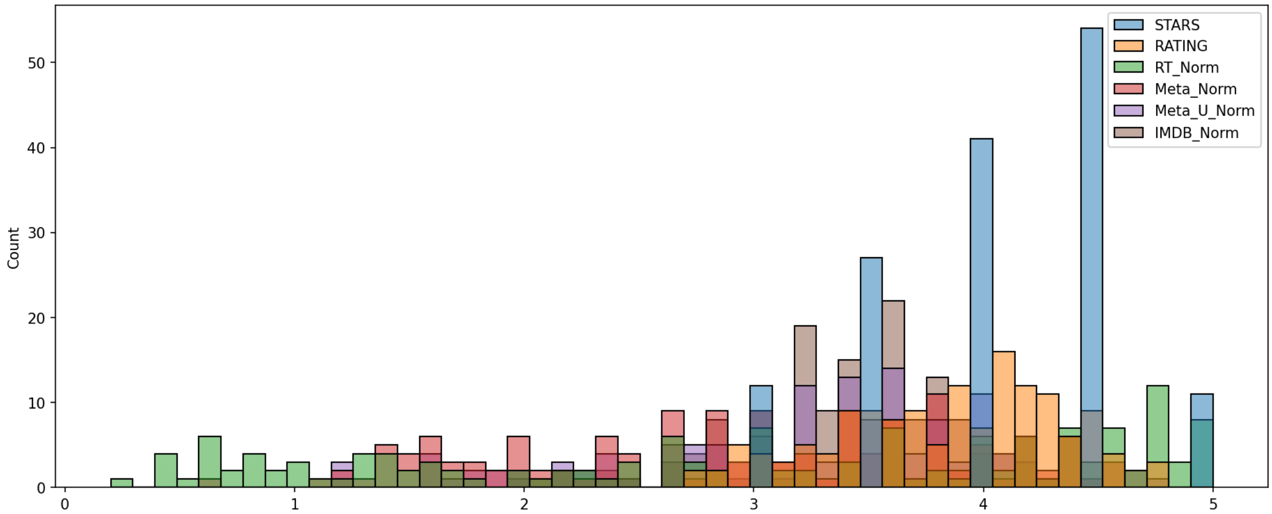
```
---------------------------------------------------------------------------NameError                                 Traceback (most recent call las
      1 fig, ax = plt.subplots(figsize=(15,6), dpi=150)
      2 sns.kdeplot(data=norm_scores[['RT_Norm', 'STARS']], clip=[0,5], shade=True, palette='Set1', ax=ax)
----> 3 move_legend(ax, "upper left")
Cell In[51], line 3, in move_legend(ax, new_loc, **kws)
      1 def move_legend(ax, new_loc, **kws):
      2     old_legend = ax.legend_
----> 3     handles - old_legend.legendHandles
      4     labels = [t.get_text() for t in old_legend.get_texts()]
      5     title = old_legend.get_title().get_text()
NameError: name 'handles' is not defined
```



OPTIONAL TASK: Create a histplot comparing all normalized scores.

In [55]:
```
plt.subplots(figsize=(15,6), dpi=150)
sns.histplot(norm_scores, bins=50)
```

Out [55]: <Axes: ylabel='Count'>

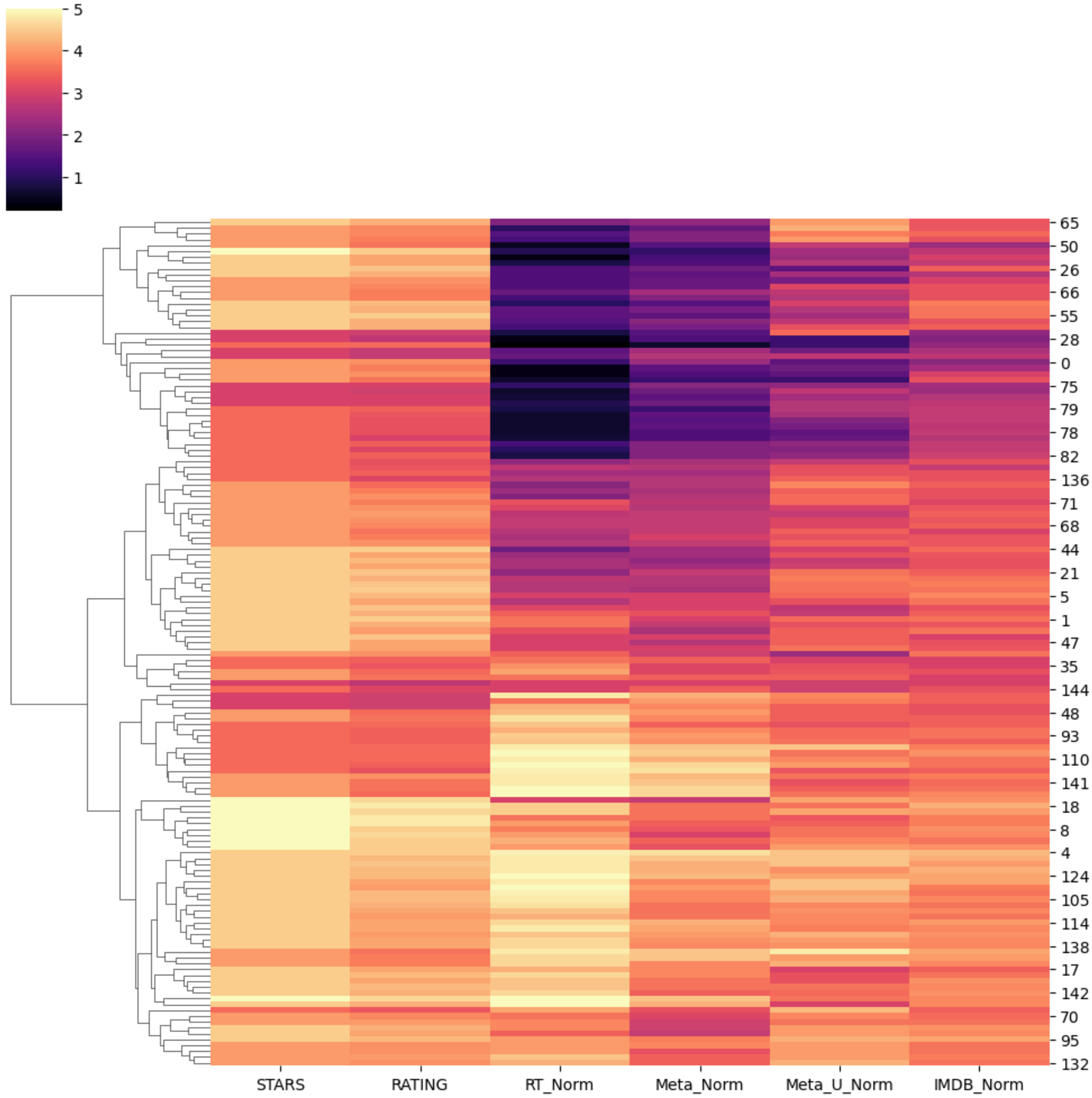


How are the worst movies rated across all platforms?

TASK: Create a clustermap visualization of all normalized scores. Note the differences in ratings, highly rated movies should be clustered together versus poorly rated movies. Note: This clustermap does not need to have the FILM titles as the index, feel free to drop it for the clustermap.

In [56]:
```
sns.clustermap(norm_scores, cmap='magma', col_cluster=False)
```

Out [56]: <seaborn.matrix.ClusterGrid at 0x265a7848350>



TASK: Clearly Fandango is rating movies much higher than other sites, especially considering that it is then displaying a rounded up version of the rating. Let's examine the top 10 worst movies. Based off the Rotten Tomatoes Critic Ratings, what are the top 10 lowest rated movies? What are the normalized scores across all platforms for these movies? You may need to add the FILM column back in to your DataFrame of normalized scores to see the results.

In [57]:
```
norm_films = df[['STARS', 'RATING', 'RT_Norm','Meta_Norm', 'Meta_U_Norm', 'IMDB_Norm', 'FILM']]
```

In [58]:
```
norm_films.nsmallest(10, 'RT_Norm')
```

Out [58]:

	STARS	RATING	RT_Norm	Meta_Norm	Meta_U_Norm	IMDB_Norm	FILM
49	3.5	3.5	0.2	0.6	1.2	2.2	Paul Blart: Mall Cop 2 (2015)
25	4.5	4.1	0.4	1.3	2.3	3.0	Taken 3 (2015)
28	3.0	2.7	0.4	1.4	1.2	2.0	Fantastic Four (2015)
54	4.0	3.7	0.4	1.6	1.8	2.4	Hot Pursuit (2015)
84	4.0	3.9	0.4	1.4	1.6	3.0	Hitman: Agent 47 (2015)
50	4.0	3.6	0.5	1.5	2.8	2.3	The Boy Next Door (2015)
77	3.5	3.2	0.6	1.5	2.0	2.8	Seventh Son (2015)
78	3.5	3.2	0.6	1.4	1.6	2.8	Mortdecai (2015)
83	3.5	3.3	0.6	1.6	2.5	2.8	Sinister 2 (2015)
87	3.5	3.2	0.6	1.6	1.9	2.7	Unfinished Business (2015)

FINAL TASK: Visualize the distribution of ratings across all sites for the top 10 worst movies.

In [59]:
```
plt.figure(figsize=(15,6), dpi=150)
worst_films = norm_films.nsmallest(10, 'RT_Norm').drop('FILM', axis=1)
sns.kdeplot(data=worst_films, clip=[0,5], shade=True, palette='Set1')
plt.title("Ratings for Rt Critic's 10 worst reviewed films")
```

```
C:\Users\vigne\AppData\Local\Temp\ipykernel_17808\2172007374.py:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=worst_films, clip=[0,5], shade=True, palette='Set1')
```

Out [59]: Text(0.5, 1.0, "Ratings for Rt Critic's 10 worst reviewed films")