In [1]:
```
# import necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

#import urlib.requesst to open URLs
from urllib.request import urlopen

#import Beautiful Soup package to extract data from html fles
from bs4 import BeautifulSoup
import re

#import necessary modules for data visualization
from pylab import rcParams
import seaborn as sns, numpy as np
```

In [2]:
```
from requests import get
url = 'http://www.imdb.com/search/title?release_date=2019&sort=num_votes,desc&pag
response = get(url)
```

In [3]:
```
from bs4 import BeautifulSoup
html_soup = BeautifulSoup(response.text, 'html.parser')
type(html_soup)
```

Out[3]:
```
bs4.BeautifulSoup
```

In [4]:
```
mv_containers = html_soup.find_all('div', class_ = 'lister-item mode-advanced')
```

In [5]:
```
headers = {"Accept-Language": "en-US, en;q=0.5"}
```

In [6]:
```
pages = [str(i) for i in range(1,10)]
years_url = [str(i) for i in range(2010,2019)]
```

In [7]:
```
from time import sleep
from random import randint
```

In [8]:
```
from time import time;start_time = time()
from datetime import timedelta
requests = 0
for _ in range(5):
# A request goes here
    requests += 1
    sleep(randint(1,3))
    elapsed_time = time() - start_time
    print('Request: {}; Frequency: {} requests/s'.format(requests, requests/elaps
```

```
Request: 1; Frequency: 0.333296885767395 requests/s
Request: 2; Frequency: 0.33249152907629426 requests/s
Request: 3; Frequency: 0.33230358028647183 requests/s
Request: 4; Frequency: 0.33231872808323115 requests/s
Request: 5; Frequency: 0.3834375258886133 requests/s
```

In [9]:
```
from IPython.core.display import clear_output
# start_time = time()requests = 0
for _ in range(10):
# A request would go here
    requests += 1
    sleep(randint(1,3))
    current_time = time()
    elapsed_time = current_time - start_time
    print('Request: {}; Frequency: {} requests/s'.format(requests, requests/elaps
clear_output(wait = True)
```

```
Request: 6; Frequency: 0.3983508356348098 requests/s
Request: 7; Frequency: 0.3875384799547849 requests/s
Request: 8; Frequency: 0.3796001732829742 requests/s
Request: 9; Frequency: 0.37382423390253505 requests/s
Request: 10; Frequency: 0.3693248118763715 requests/s
Request: 11; Frequency: 0.39177691546230153 requests/s
Request: 12; Frequency: 0.41269024659816783 requests/s
Request: 13; Frequency: 0.40508031757385815 requests/s
Request: 14; Frequency: 0.39882979555572556 requests/s
Request: 15; Frequency: 0.40427067809823497 requests/s
```

In [10]:
```
from warnings import warn
warn("Warning Simulation")
```

```
C:\Users\gladies\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: UserWarni
ng: Warning Simulation
```

In [11]:
```
# Redeclaring the lists to store data in
names = []
years = []
imdb_ratings = []
metascores = []
votes = []
grade_class = []
runing_time = []
moviegenre = []

# Preparing the monitoring of the loop
start_time = time()
requests = 0

# For every year in the interval 2010-2019
for year_url in years_url:

    # For every page in the interval 1-4
    for page in pages:

        # Make a get request
        response = get('http://www.imdb.com/search/title?release_date=' + year_u
        '&sort=num_votes,desc&page=' + page, headers = headers)

        # Pause the loop
        sleep(randint(8,15))

        # Monitor the requests
        requests += 1
        elapsed_time = time() - start_time
        print('Request:{}; Frequency: {} requests/s'.format(requests, requests/e
        clear_output(wait = True)

        # Throw a warning for non-200 status codes
        if response.status_code != 200:
            warn('Request: {}; Status code: {}'.format(requests, response.status_

        # Break the loop if the number of requests is greater than expected
        if requests > 100:
            warn('Number of requests was greater than expected.')
            break

        # Parse the content of the request with BeautifulSoup
        page_html = BeautifulSoup(response.text, 'html.parser')

        # Select all the 50 movie containers from a single page
        imdb_containers = page_html.find_all('div', class_ = 'lister-item mode-a

        # For every movie of these 50
        for container in imdb_containers:
            # If the movie has a Metascore, then:
            if container.find('div', class_ = 'ratings-metascore') is not None:

                # Scrape the name
                name = container.h3.a.text
                names.append(name)
```

```
                # Scrape the year
                year = container.h3.find('span', class_ = 'lister-item-year').tex
                years.append(year)

                # Scrape the IMDB rating
                imdb = float(container.strong.text)
                imdb_ratings.append(imdb)

                # Scrape the Metascore
                m_score = container.find('span', class_ = 'metascore').text
                metascores.append(int(m_score))

                # Scrape the number of votes
                vote = container.find('span', attrs = {'name':'nv'})['data-value
                votes.append(int(vote))

                # Scrape the grade
                grade = container.find('span', class_ = 'certificate').text
                grade_class.append(grade)

                # Scrape the runtime
                runtime = container.find('span', class_ = 'runtime').text
                runing_time.append(runtime)

                # Scrape the genre
                genre = container.find('span', class_ = 'genre').text
                moviegenre.append(genre)
```

```
Request:81; Frequency: 0.07684425983252377 requests/s
```

In [12]:
```
import pandas as pd
imdb_ratings = pd.DataFrame({'movie': names,
'year': years,
'imdb': imdb_ratings,
'metascore': metascores,
'votes': votes,
'grade': grade_class,
'runtime': runing_time,
'genre': moviegenre
})
print(movie_ratings.info())
imdb_ratings.tail(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3690 entries, 0 to 3689
Data columns (total 8 columns):
movie        3690 non-null object
year         3690 non-null object
imdb         3690 non-null float64
metascore    3690 non-null int64
votes        3690 non-null int64
grade        3690 non-null object
runtime      3690 non-null object
genre        3690 non-null object
dtypes: float64(1), int64(2), object(5)
memory usage: 230.8+ KB
None
```

Out[12]:

	movie	year	imdb	metascore	votes	grade	runtime	genre
3680	Bad Times at the El Royale	(2018)	7.1	60	103738	R	141 min	\nCrime, Drama, Mystery
3681	Ralph Breaks the Internet	(2018)	7.1	71	103648	PG	112 min	\nAnimation, Adventure, Comedy
3682	The Predator	(2018)	5.4	48	102950	R	107 min	\nAction, Adventure, Sci-Fi
3683	The Nun	(2018)	5.3	46	100137	R	96 min	\nHorror, Mystery, Thriller
3684	A Simple Favor	(2018)	6.8	67	98625	R	117 min	\nComedy, Crime, Drama
3685	Halloween	(I) (2018)	6.6	67	98280	R	106 min	\nHorror, Thriller
3686	The Ballad of Buster Scruggs	(2018)	7.3	79	96276	R	133 min	\nComedy, Drama, Musical
3687	Maze Runner: The Death Cure	(2018)	6.2	50	96176	PG-13	143 min	\nAction, Sci-Fi, Thriller
3688	Pacific Rim: Uprising	(2018)	5.6	44	93515	PG-13	111 min	\nAction, Adventure, Sci-Fi
3689	Tag	(I) (2018)	6.5	56	93443	R	100 min	\nComedy

In [13]:
```
imdb_ratings = imdb_ratings[['movie', 'year', 'imdb', 'metascore', 'votes', 'gra
imdb_ratings.head()
```

Out[13]:

	movie	year	imdb	metascore	votes	grade	runtime	genre
0	Inception	(2010)	8.8	74	1884162	PG-13	148 min	\nAction, Adventure, Sci-Fi
1	Shutter Island	(2010)	8.1	63	1031297	R	138 min	\nMystery, Thriller
2	Toy Story 3	(2010)	8.3	92	704012	G	103 min	\nAnimation, Adventure, Comedy
3	Iron Man 2	(2010)	7.0	57	674309	PG-13	124 min	\nAction, Adventure, Sci-Fi
4	Black Swan	(2010)	8.0	79	659578	R	108 min	\nDrama, Thriller

In [14]:
```
imdb_ratings['year'].unique()
```

Out[14]:
```
array(['(2010)', '(I) (2010)', '(2011)', '(I) (2011)', '(2012)',
       '(I) (2012)', '(2013)', '(I) (2013)', '(2014)', '(I) (2014)',
       '(II) (2014)', '(2015)', '(I) (2015)', '(II) (2015)', '(2016)',
       '(II) (2016)', '(I) (2016)', '(IX) (2016)', '(2017)', '(I) (2017)',
       '(2018)', '(I) (2018)', '(III) (2018)'], dtype=object)
```

In [15]:
```
imdb_ratings.loc[:, 'year'] = movie_ratings['year'].str[-5:-1].astype(int)
```

In [16]:
```
imdb_ratings['year'].tail(3)
```

Out[16]:
```
3687    2018
3688    2018
3689    2018
Name: year, dtype: int32
```

In [17]:
```
imdb_ratings.describe().loc[['min', 'max'], ['imdb', 'metascore']]
```

Out[17]:

	imdb	metascore
min	4.1	27.0
max	8.8	100.0

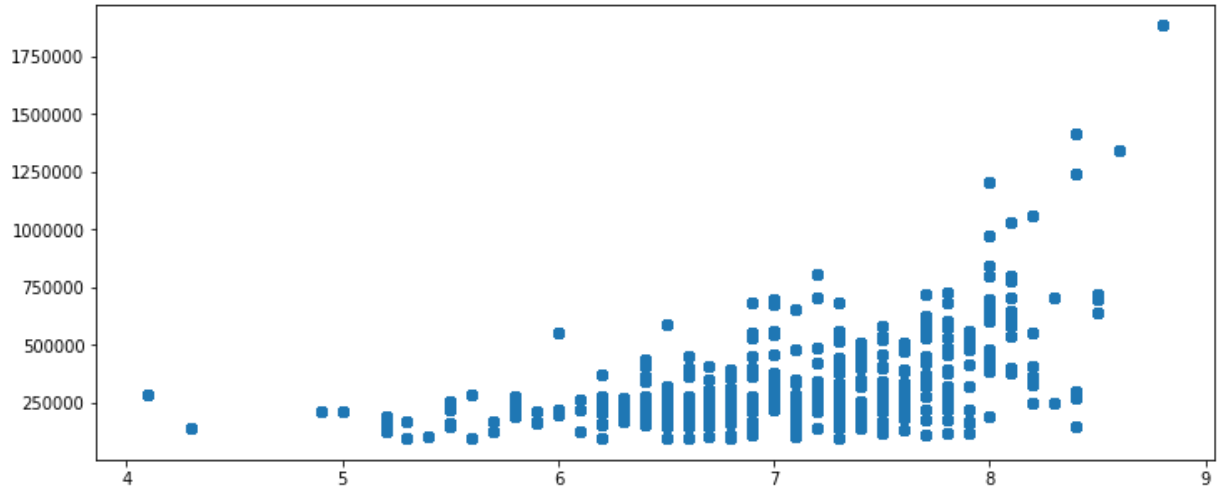
In [18]:
```
imdb_ratings['n_imdb'] = movie_ratings['imdb'] * 10
imdb_ratings.head(3)
```

Out[18]:

	movie	year	imdb	metascore	votes	grade	runtime	genre	n_imdb
0	Inception	2010	8.8	74	1884162	PG-13	148 min	\nAction, Adventure, Sci-Fi	88.0
1	Shutter Island	2010	8.1	63	1031297	R	138 min	\nMystery, Thriller	81.0
2	Toy Story 3	2010	8.3	92	704012	G	103 min	\nAnimation, Adventure, Comedy	83.0

In [19]:
```
imdb_ratings.to_csv('movie_ratings_2019.csv')
```

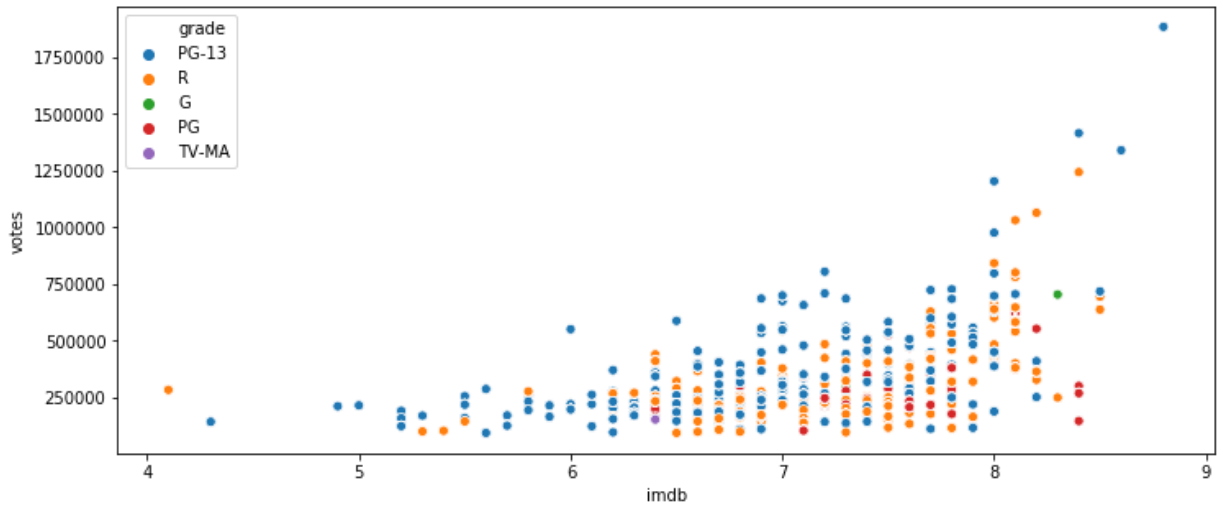
In [20]:
```
#Start to make charts and do statistical analysis
import seaborn as sns, numpy as np
from matplotlib import pyplot as plt
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12,5)
plt.scatter(movie_ratings.imdb, movie_ratings.votes)
plt.show()
```



In [21]:
```
df = movie_ratings
print(df['runtime'])
```

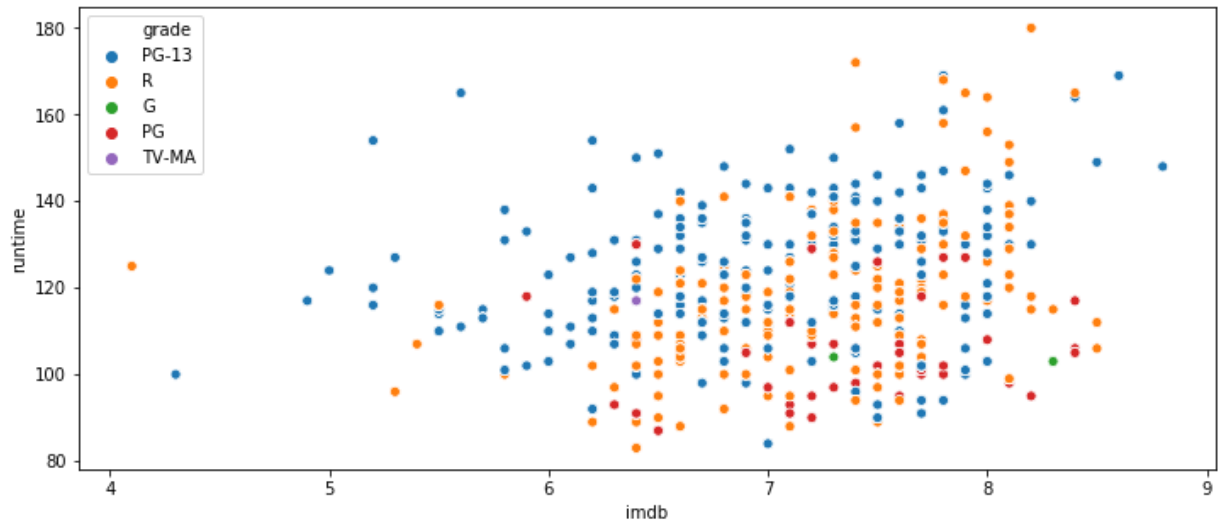
```
0       148 min
1       138 min
2       103 min
3       124 min
4       108 min
         ...
3685    106 min
3686    133 min
3687    143 min
3688    111 min
3689    100 min
Name: runtime, Length: 3690, dtype: object
```

In [22]:
```
ax = sns.scatterplot(x="imdb", y="votes", hue="grade",data=df)
```



In [23]:
```
#Need to change format of Volume from string to numeric
runtime_num_list = df['runtime'].tolist()
runtime_num = []
for i in runtime_num_list:
    if i.endswith('min'):
        num = i.strip("min")
        runtime_num.append(num)
#pass num back to Volume_list
df['runtime'] = runtime_num
df.head(5)
df['runtime'] = pd.to_numeric(df['runtime'])
```

In [24]:
```
ax = sns.scatterplot(x="imdb", y="runtime", hue="grade",data=df)
yticks=np.arange(120,170,5)
```



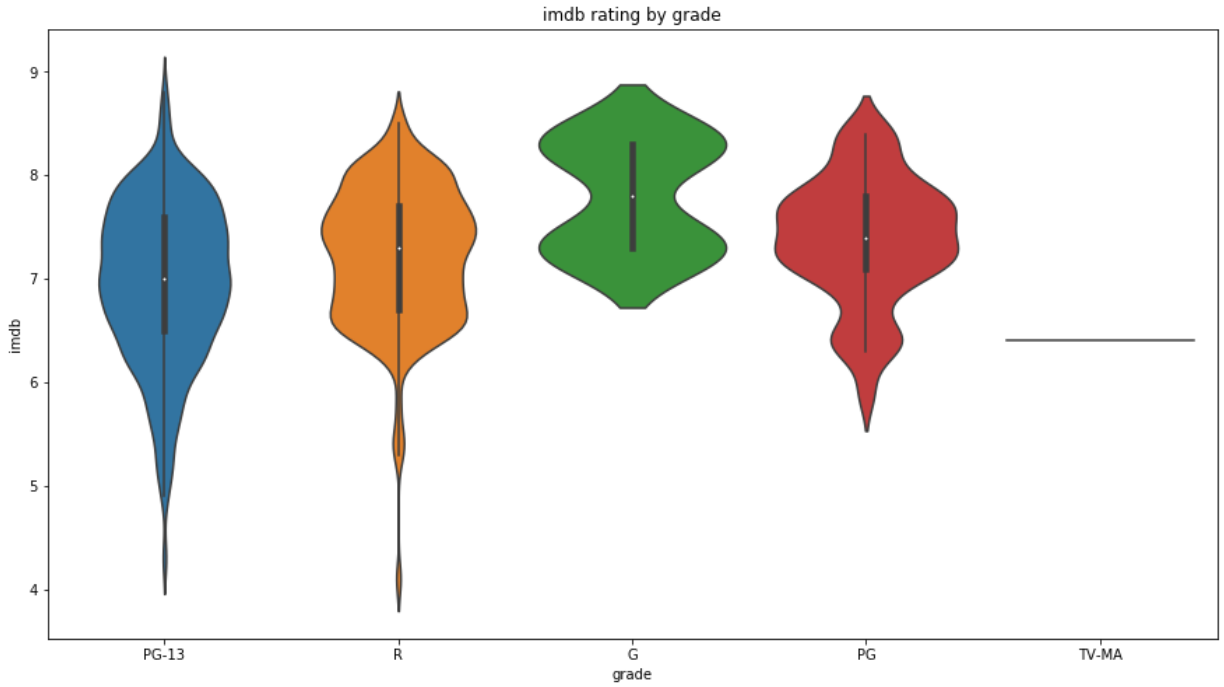
In [25]:
```
df = movie_ratings
df.describe()
```

Out[25]:

	year	imdb	metascore	votes	runtime	n_imdb
count	3690.000000	3690.000000	3690.000000	3.690000e+03	3690.000000	3690.000000
mean	2013.985366	7.107561	64.485366	3.353521e+05	119.797561	71.075610
std	2.591100	0.730896	15.330229	2.102535e+05	17.408287	7.308964
min	2010.000000	4.100000	27.000000	9.344300e+04	83.000000	41.000000
25%	2012.000000	6.600000	53.000000	1.982840e+05	107.000000	66.000000
50%	2014.000000	7.200000	65.000000	2.681510e+05	118.000000	72.000000
75%	2016.000000	7.700000	75.000000	4.092850e+05	131.000000	77.000000
max	2018.000000	8.800000	100.000000	1.884162e+06	180.000000	88.000000

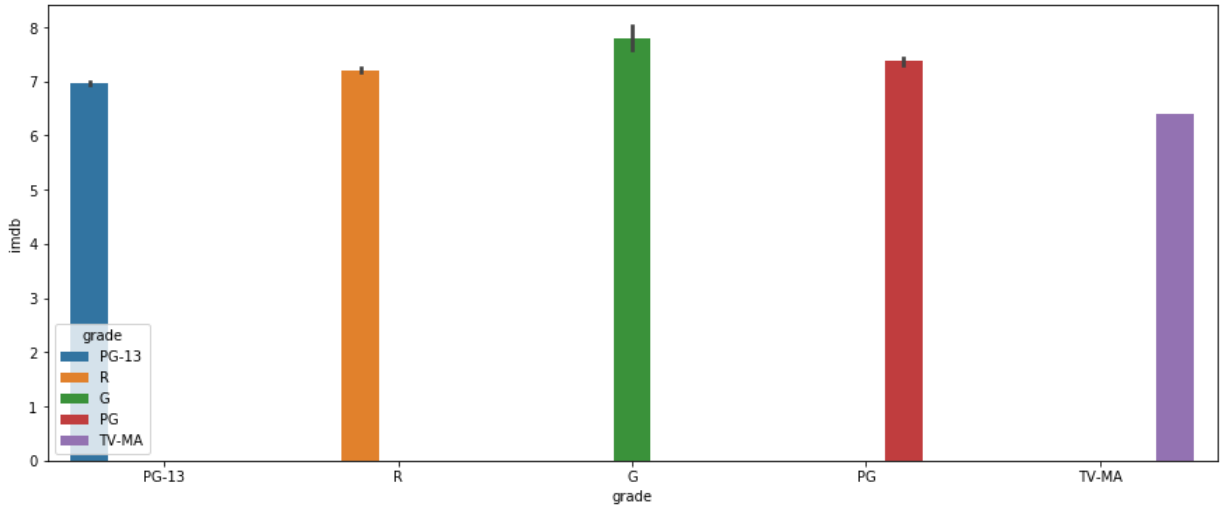
In [26]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
#
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(15, 8))
#using violinplot to showcase density and distribtuion of prices
viz_2=sns.violinplot(data=df, x='grade', y='imdb')
viz_2.set_title('imdb rating by grade')
```

Out[26]: Text(0.5, 1.0, 'imdb rating by grade')

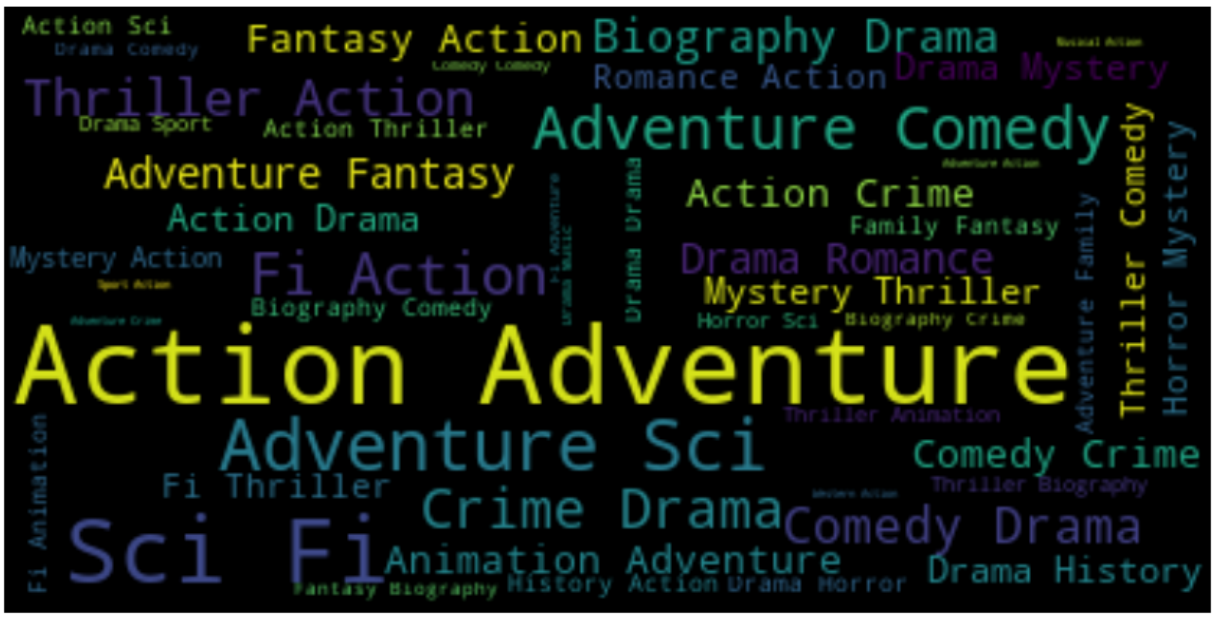


In [27]:
```
df = movie_ratings
plt.figure(figsize=(15, 6))
sns.barplot(x='grade', y='imdb', hue='grade',data=df)
```

Out[27]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1acd77902b0>
```



In [28]:
```
#word cloud
from wordcloud import WordCloud, ImageColorGenerator
text = " ".join(str(each) for each in df.genre)
# Create and generate a word cloud image:
wordcloud = WordCloud(max_words=400).generate(text)
plt.figure(figsize=(15,10))
# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



In [29]:
```
import sys
print(sys.executable)
```

```
C:\Users\gladies\Anaconda3\python.exe
```

In [ ]:
```

```

In [ ]:
```

```