Romantic films versus Horror films: Exploring differences in box-office values, audience ratings and sentiment of scripts

Programming with Data Midterms

1. Introduction

Most projects on movies use data analytics to identify audience preferences and predict commercial success. Natural language processing is applied on scripts to create genre classifiers and to analyse sentiments expressed in scripts. These projects either focus on the movie industry as a whole, or focus on only a specific genre.

No major study was done to pit two genres of movies against one another, or to compare and contrast the two different markets. Analysing two genres of films can allow us to appreciate why an individual may prefer one genre over another and help us compare the commercial prospects in the industry.

I want to compare the horror and romance movie industries using data analysis - to identify differences between the two genres and quantify this disparity using three main parameters - box-office numbers (budget and revenue), audience ratings and the sentiments of scripts.

I decided to focus on romance and horror as they are on opposing ends of the genre spectrum, with both evoking strong feelings from the audience. I will be focusing on three aspects - the profitability of the two markets, the popularity of the genre and the sentiment of horror versus romantic scripts.

1.1 Aims and objectives

Research question

My research question is as follows: What are the key differences between horror and romance films - in terms of commercial success and the sentiments of the movie scripts?

With this project, I will like to explore:

- Market of romance and horror films
    - Are film-makers spending more on movies over the years?
    - Is there any difference in the budget and revenue of the two genres?
    - How strong is the relationship between revenue and budget?
- Popularity of genre
    - Which of the genre is generally more popular?
- Analysing romance and horror scripts
    - Are romance and horror scripts equally provocative?
    - Are romance and horror scripts semantically disparate?

At the end of the project, I will have a more definitive answer in the differences between the horror and romance movie markets and their scripts.

Brief data processing pipeline

- In Section 2 will cover the data gathering process, which involves the use of APIs and webscraping.
- In Section 3, I will cover data cleaning and processing, where I will handle any missing values and inconsistencies. In this section, raw data gathered from the three sources will be combined into a format that is suitable for further analysis.
- In Section 4, I will conduct the exploratory data analysis to gleam some understanding from the data set.

1.2 Data Collection

In order to compare the two markets, I have to gather reliable data on romance and horror movies. I need numbers such as box-office earnings and movie ratings, along with accurate versions of the scripts for sentiment analysis.

This project uses data primarily from three sources:

- The Internet Movie Script Database (IMSDb)
- The Movie Database (TMDB)

- Rotten Tomatoes

IMSDb was chosen over other databases of movie scripts, like Simply Scripts or Go Into the Story (official blog of The Blacklist). This is because it has the largest corpus, with frequent updates. Multiple NLP academic projects such as a Stanford student study on film success prediction [1] and a data science project to study gendered dialogue [2] use this corpus as the main dataset. This lends credibility to the scripts as a resource for NLP projects. I have scraped 196 romantic film scripts and 154 horror film scripts from IMSDb.

Movie information such as budget, runtime and box-office numbers are gathered from TMDB. It provides accurate numbers and has a native API that facilitates gathering of data. It was chosen overs sites such as Box Office Mojo (by IMDbPro) or The Numbers, as it provides succinct and accurate box-office values, as well as audience ratings.

I chose to supplement the dataset with audience scores from Rotten Tomatoes. Rotten Tomatoes is the go-to site for movie reviews and is reputable for its accurate reflection of audience reactions. It is chosen over websites such as Screen Rant and MovieWeb, as these sites use

1.2.1 Limitations of dataset

Limited number of movies analysed

The datasets gathered for this project include fewer than 200 movies from each genre for the year of 1931 to 2023. This means that on average, only 1 to 3 horror and romance movies are covered per year. The analysis performed on this data may not be representative of the genre as a whole. However, given the limited volume of open-to-public scripts available, we have to accept this as an acceptable constraint for this study.

Inaccuracies in script

Many of the scripts analysed are early drafts of the movies and may not follow the dialogue or plot of the final release. As it is impossible to check each script given the limited amount of time, the scripts taken from IMSDb are assumed to be accurate.

1.3 Ethical considerations

1.3.1 Use of data

TMDB for market information

I initially chose IMDb as a resource as it is the most robust site for information on movies. It has a public Python package, Cinemagoer, that allows for easy retrieval of data. However, IMDb Conditions of Use explicitly prohibits data extraction tools on its site without written explicit consent. I have written code with Cinemagoer to extract information from IMDb (included in Appendix A) but did not execute it as I am unable to acquire a response from the licensing department. I decided, instead, to switch to TMDB, which provides similar data.

TMDB has its own native API that supports querying of data with a personal API key. I created a personal account with TMDB and acquired an API key, which I used for the length of my project. TMDB encourages the use of its API to gather metadata on movies. Since I am using the data solely for a student project and not for commercial use, I can legally save a copy of the data on my machine. This makes TMDB the ideal source of data for box-office numbers.

IMSDb for movie scripts

IMSDb released the movie scripts under the fair use exception. This means that the material can be used without seeking consent, under certain conditions such as for educational purposes. I have no desire to use the scripts for anything outside of this project, or to republish the scripts elsewhere.

Rotten Tomatoes for ratings

There is no clause in Rotten Tomatoes site that prevents the use of web-scrapping. However, upon inspection of reddit's robot.txt, I found out that it disallows the use of scraping on comments and reviews. I took care to avoid these and extracted only overall audience scores from the site. These scores do not identify any individuals and only reflect average ratings.

1.3.2 Usage of final analysis

Future users of the data derived from this project will need to comply with the updated Conditions of Use by the above sites at their time of use. There is a potential for the project to create a bias against either genre of film. Hence, it is important to emphasise that the results should not be used in place of an authority on the subject. All analysis are purely my own conjecture, and should not be used to influence commercial decisions. While there is an objective comparison of box office figures between the two genres, it is important to note that it should not be the sole yardstick of success for a genre of film.

1.3.3 Impact of final analysis

This project poses a few potentially harms:

- The resultant analysis of the movie market may be misleading due to inaccuracies from limited sources of data.
- Sentiment analysis on the scripts may perpetuate biases inherent to the genre and may not be representative of the genre as a whole.
- Scripts are often used to train machine learning models. Usage of scripts may increase valid concerns of replacing screenwriters with generative AI.

I have taken these few points into consideration:

- The negative impact of an inaccurate analysis is minimal as this project serves to provide a cursory preview on the market. The results are not used to make any business decisions - rather, it serves as cause for future studies. By carefully cultivating data from reputable sources, we have minimised the risk of inaccuracies.
- The sentiment analysis on the scripts are not used to represent a person or social group and the diverse range of scripts reduces bias.
- The scripts will not be fed into any machine learning algorithms and the sentiment analysis from this project will not be sufficient to imitate a screenwriter, or replace the role of artists in the industry. I do not endorse any generative AI that uses art from content creators without consent.

1.4 Modules, libraries and packages used in the project

The following code block contains the necessary imports of packages for this project. Most notably, I used:

- Scrapy and TMDB's API for webscraping
- Pandas, Regex and Numpy for data wrangling
- NLTK for launguage processing of scripts

In [3]:
```
#---------------------------------------------------------------
# for webscraping
import sys
import scrapy
from scrapy.crawler import CrawlerProcess

# to gather tmdb data
import tmdbsimple as tmdb
!{sys.executable} -m pip install tmdbsimple
import requests
tmdb.API_KEY = '5959a1e90d29ba2c3492253da68352ad'
tmdb.REQUESTS_SESSION = requests.Session()
```

```
#-----------------------------------------------------------------
# for data wrangling
import pandas as pd
import regex as re
import xml.etree.ElementTree as ET
import numpy as np
import csv
import ast

#-----------------------------------------------------------------
# for language processing of scripts
import nltk
from nltk import word_tokenize
from nltk.corpus import stopwords
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('omw-1.4')

from nltk.stem import WordNetLemmatizer
from nltk.probability import FreqDist
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from wordcloud import WordCloud
from collections import Counter

#-----------------------------------------------------------------
# for plotting of figures in data analysis
import matplotlib.pyplot as plt
import seaborn as sns
```

```
Requirement already satisfied: tmdbsimple in /Users/main/anaconda3/lib/python3.11/site-packages (2.9.1)
Requirement already satisfied: requests in /Users/main/anaconda3/lib/python3.11/site-packages (from tmdbsimple) (2.3
1.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/main/anaconda3/lib/python3.11/site-packages (from r
equests->tmdbsimple) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /Users/main/anaconda3/lib/python3.11/site-packages (from requests->tmd
bsimple) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/main/anaconda3/lib/python3.11/site-packages (from request
s->tmdbsimple) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in /Users/main/anaconda3/lib/python3.11/site-packages (from request
s->tmdbsimple) (2023.7.22)
```

```
[nltk_data] Downloading package wordnet to /Users/main/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /Users/main/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /Users/main/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /Users/main/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

2. Data Gathering

I have gathered data from these three main sources:

1. The Internet Movie Script Database: Scraping the movie scripts for 196 romance movies and 153 horror movies.
2. TMDB: For ratings and box office numbers
3. Rotten Tomatoes: For ratings and audience scores

Do note: In order to get data on the romance and horror genres, the code below must be run twice - once for romance titles and once for horror titles. Sections 2 (Data Gathering) and 3 (Data Cleaning and Processing) is executed twice with the exact same code as below, with the exception of line 6 of the ScriptSpider taking on either 'https://imsdb.com/genre/Romance' or 'https://imsdb.com/genre/Horror' as the root url.

In addition, the csv filenames are altered with each run of the code (replacing romance with horror).

2.1 Webscraping the IMSDb website for scripts

Scrapy was used in this project for web-scraping as it works as an efficient standalone framework that allows us to crawl through webpages and collect data.

By the end of the entire run, we will have a list named 'scripts' that contains all 193 scripts of romance film used in the project.

To get the horror films, a root URL of 'https://imsdb.com/genre/Horror' is used instead to get the 153 horror scripts.

Please note that this webscraping code from IMSDb block may take up to 5 minutes on each run.

2.1 Internet Movie Script Database (IMSDb) data

I have written a crawler with Scrapy's spider module to automate the process of downloading the scripts. I start with the root url, which is a page of film titles of each genre, with links to info page on specific movie titles. On the individual info page, another link will lead us to the script.

Hence, the spider requires 3 parse calls:

1. to follow each link to individual movie pages from the main genre page
2. to follow each link to the script from an individual movie page
3. to extract the text in the script and append to a list.

In order to get the urls in each parse call, I used either a xpath or a css selector. Since each page has the same html format, the spider can successfully crawl through the website.

In [ ]:
```
class ScriptSpider(scrapy.Spider):   # create a spider with scrapy module
    name = 'script_spider'

    # requests start with imsdb's webpage for romantic movies
    def start_requests(self):
        yield scrapy.Request(url = 'https://imsdb.com/genre/Romance', callback = self.parse)
#         yield scrapy.Request(url = 'https://imsdb.com/genre/Horror', callback = self.parse)

    # spider follows each link to a romantic movie
    def parse(self,response):
        movie_titles = response.xpath('//p/a/@title').extract()
        movie_links = response.xpath('//p/a/@href').extract()

        for link in movie_links:
            yield response.follow(url = link, callback = self.parse_movie)

    # spider follows the 'script link' to another webpage containing script
    def parse_movie(self,response):
        movie_script_links = response.css('.script-details > tr:nth-of-type(2) > td:nth-of-type(2) > a:last-child::att

        for link in movie_script_links:
            yield response.follow(url = link, callback = self.parse_script)

    # spider parses all the text that makes up the script and appends it to a list of scripts
    def parse_script(self, response):

        script = response.xpath('//td[@class="scrtext"]//text()').extract()
        scripts.append(script)

        url = response.request.url
        urls.append(url)

# initialising the lists outside of the spider
scripts = list()
urls = list()

# running the spider
process = CrawlerProcess()
process.crawl(ScriptSpider)
process.start()
```

In the blocks below, I save each scraped script as a text file. While performing a quick check on the files, I have discovered that the scripts do not always reflect the title of the previous webpage. *(For example, the webpage of 'Bonnie and Clyde' may bring us to the script of 'Bruce Almighty' instead.)* In order to ensure the quality and consistency of the data, I have to make sure that the title of the txt file matches the script inside.

The most accurate way to get the title of the script is to use the title directly in the html of the script page. To do this, I used python string method removeprefix and removesuffix. A new list 'titles' is created instead, that is used later to query TMDB.

In [ ]:
```
titles = [] # used to query TMDB later
files = list()

for n in range(len(scripts)):
    title = ((urls[n].removeprefix('https://imsdb.com/scripts/').removesuffix('.html'))).lower()   # we use the title
    titles.append(title)

    file_name = title + '.txt' # saves the file with the movie_title
    files.append(file_name) # saves it as a list of files
    f = open(file_name,'w') # creates a file
    f.writelines(scripts[n]) # write the entire script into the file
    f.close() #closes the file
```

In [ ]:
```
# saving all the romance movie titles, title changed for horror films
with open("romance_titles.csv", mode = "w", newline = "") as file:
    writer = csv.writer(file)
    writer.writerow(titles)
```

2.2 TMDB Data

In this section, I will be using version 3 of The Movie Database (TMDB) API. This API provides methods to find details on movies, television series and actors. In order to access the API, I created a personal account with TMDB and generated an API key. I imported tmdbsimple, a python package that streamlines the workflow with the API, as tmdb.

Please do note that the code takes around 10 minutes to run on each round. As with the above section, I ran the code twice - once for the romance films and another for the horror films - by changing the csv file called. The code to run the horror films have been commented out.

2.2.1 Storing TMDB data

The tmdbsimple module allows us to fetch movie data by parsing a string that forms the query, along with my temporary api key.

For example, the '/movie?query=' substring allows us to find a particular movie. I looped through all the movie titles, appending the title to the end of the substring. I also used 'credits?' to query for the movie cast.

TMDB API only supports JSON format for its data. Since the data gathered is in the form of a key-value pair, I chose to use a nested dictionary is to contain all the tmdb information.

For each individual movie, I retrieved the necessary information from the json object and stored it in a dictionary, which I then stored in the nested dictionary 'tmdb_info'. Lastly, I saved the output as a csv file as it contains mostly values that are needed for numerical analysis at a later stage.

In [ ]:
```
tmdb_info = {} # nested dictionary to contain information of all films
no_info = [] # list of titles with no information
for title in titles: # list from Section 2.1
    try:
        url = "https://api.themoviedb.org/3/search/movie?query=" + title + "&api_key=" + api_key
        response = requests.get(url)
        response_json = response.json() # get the json object

        movie_id = response_json['results'][0]['id'] # get id of the movie
        url_details = 'https://api.themoviedb.org/3/movie/' + str(movie_id) + "?api_key=" + api_key
        movie_details = requests.get(url_details)
        details_json = movie_details.json()

        url_crew = 'https://api.themoviedb.org/3/movie/' + str(movie_id) + "/" + "credits" + "?api_key=" + api_key
        crew_details = requests.get(url_crew)
        crew_json = crew_details.json()

        title = details_json['original_title']
        cast_lst = [member['name'] for member in crew_json['cast'] if member['known_for_department'] == 'Acting']
        company_lst = [company['name'] for company in details_json['production_companies']]
        genre_lst = [genre['name'] for genre in details_json['genres']]

        # creating a dictionary of all the elements extracted
        new_dict = {
        "name" : details_json['original_title'],
        "imdb_id" : details_json['imdb_id'],
        "release_date" : details_json['release_date'],
        "runtime" : details_json['runtime'],
        "budget" : details_json['budget'],
        "revenue" : details_json['revenue'],
         "rating" : details_json['vote_average'],
         "vote_count" : details_json['vote_count'],
        "popularity" : details_json['popularity'],
        "cast" : cast_lst,
        "companies" : company_lst,
        "genres" : genre_lst
        }
        tmdb_info[title] = new_dict
    except:
        no_info.append(title) # no_info will have all the titles that have missing titles
```

I saved the nested dictionary as a dataframe and exported it into a csv. The CSV files will be used later as direct import into the program for subsequent data analysis processes. Saving the files in a flat-file format will save us from web-scraping everytime we open the project. Furthermore, webpages can change in format over time. This is a fool-proof way of making sure that the data extracted is intact in the future.

In [ ]:
```
tmdb_info_df = pd.DataFrame.from_dict(tmdb_info, orient = "index")
tmdb_info_df.to_csv("romance_tmdb.csv", header = True, index = True)
```

2.3 Rotten Tomatoes

Rotten tomatoes is a website that hosts movie reviews and ratings from audience. I used the rottentomaties-python 0.6.5 package to scrape the data from the site. I have imported rottentomatoes as rt, and I used the standalone functions to retrieve the data. I stored the information for each movie in a dictionary. I saved the resultant 'rotten_tomatoes' nested dictionary as a csv file.

In [ ]:
```python
rotten_tomatoes = {}

for title in titles: # from section 2.1
    try:
        weighted_score = rt.weighted_score(title)
        tomatometer = rt.tomatometer(title)
        audience_score = rt.audience_score(title)
    except:
        weighted_score = np.nan
        tomatometer = np.nan
        audience_score = np.nan

    # store element as a dict
    new_dict = {
        "weighted_score" : weighted_score,
        "tomatometer" : tomatometer,
        "audience_score" : audience_score
    }

    rotten_tomatoes[title] = new_dict # the resultant data will be a nested dictionary

rotten_tomatoes_df = pd.DataFrame.from_dict(rotten_tomatoes, orient = "index")
rotten_tomatoes_df.to_csv("romance_rt.csv", header = True, index = True)
# rotten_tomatoes_df.to_csv("horror_rt.csv", header = True, index = True)
```

2.4 Horror movie dataset

In the code below, I will repeat the same process from the last three sections for horror films. The code is exactly the same, with the exception of the root url for the spider - 'https://imsdb.com/genre/Horror' - and the filenames of the CSV saved.

In [ ]:
```python
# running again for horror films
class ScriptSpider(scrapy.Spider):   # create a spider with scrapy module
    name = 'script_spider'

    # requests start with imsdb's webpage for horror films.
    def start_requests(self):
        yield scrapy.Request(url = 'https://imsdb.com/genre/Horror', callback = self.parse)

    # spider follows each link to a romantic movie
    def parse(self,response):
        movie_titles = response.xpath('//p/a/@title').extract()
        movie_links = response.xpath('//p/a/@href').extract()

        for link in movie_links:
            yield response.follow(url = link, callback = self.parse_movie)

    # spider follows the 'script link' to another webpage containing script
    def parse_movie(self,response):
        movie_script_links = response.css('.script-details > tr:nth-of-type(2) > td:nth-of-type(2) > a:last-child::att

        for link in movie_script_links:
            yield response.follow(url = link, callback = self.parse_script)

    # spider parses all the text that makes up the script and appends it to a list of scripts
    def parse_script(self, response):

        script = response.xpath('//td[@class="scrtext"]//text()').extract()
        scripts.append(script)

        url = response.request.url
        urls.append(url)

# initialising the lists outside of the spider
scripts = list()
urls = list()

# running the spider
process = CrawlerProcess()
process.crawl(ScriptSpider)
process.start()
```

In the code below, I save scripts for the horror films as text files.

In [ ]:
```
titles = [] # used to query TMDB later
files = list()

for n in range(len(scripts)):
    title = ((urls[n].removeprefix('https://imsdb.com/scripts/').removesuffix('.html'))).lower()   # we use the title
    titles.append(title)

    file_name = title + '.txt' # saves the file with the movie_title
    files.append(file_name) # saves it as a list of files
    f = open(file_name,'w') # creates a file
    f.writelines(scripts[n]) # write the entire script into the file
    f.close() #closes the file
```

In [ ]:

```
# same as code above, title for file changed for horror films
with open("horror_titles.csv", mode = "w", newline = "") as file:
    writer = csv.writer(file)
    writer.writerow(titles)
```

Much like the romance titles above, I proceeded to gather information from tmdb and stored it as a csv file for further processing later.

In [ ]:

```
# getting tmdb info for horror films

tmdb_info = {} # nested dictionary to contain information of all films
no_info = [] # list of titles with no information
for title in titles: # list from Section 2.1
    try:
        url = "https://api.themoviedb.org/3/search/movie?query=" + title + "&api_key=" + api_key
        response = requests.get(url)
        response_json = response.json() # get the json object

        movie_id = response_json['results'][0]['id'] # get id of the movie
        url_details = 'https://api.themoviedb.org/3/movie/' + str(movie_id) + "?api_key=" + api_key
        movie_details = requests.get(url_details)
        details_json = movie_details.json()

        url_crew = 'https://api.themoviedb.org/3/movie/' + str(movie_id) + "/" + "credits" + "?api_key=" + api_key
        crew_details = requests.get(url_crew)
        crew_json = crew_details.json()

        title = details_json['original_title']
        cast_lst = [member['name'] for member in crew_json['cast'] if member['known_for_department'] == 'Acting']
        company_lst = [company['name'] for company in details_json['production_companies']]
        genre_lst = [genre['name'] for genre in details_json['genres']]

        # creating a dictinary of all the elements extracted
        new_dict = {
        "name" : details_json['original_title'],
        "imdb_id" : details_json['imdb_id'],
        "release_date" : details_json['release_date'],
        "runtime" : details_json['runtime'],
        "budget" : details_json['budget'],
        "revenue" : details_json['revenue'],
         "rating" : details_json['vote_average'],
         "vote_count" : details_json['vote_count'],
        "popularity" : details_json['popularity'],
        "cast" : cast_lst,
        "companies" : company_lst,
        "genres" : genre_lst
        }
        tmdb_info[title] = new_dict
    except:
        no_info.append(title) # no_info will have all the titles that have missing titles
```

In [ ]:
```
tmdb_info_df = pd.DataFrame.from_dict(tmdb_info, orient = "index")
tmdb_info_df.to_csv("horror_tmdb.csv", header = True, index = True)
```

I also extracted the data from Rotten Tomatoes for horror films.

In [ ]:
```
rotten_tomatoes = {}

for title in titles: # from section 2.1
    try:
        weighted_score = rt.weighted_score(title)
        tomatometer = rt.tomatometer(title)
        audience_score = rt.audience_score(title)
    except:
        weighted_score = np.nan
        tomatometer = np.nan
        audience_score = np.nan

    # store element as a dict
    new_dict = {
        "weighted_score" : weighted_score,
        "tomatometer" : tomatometer,
        "audience_score" : audience_score
    }

    rotten_tomatoes[title] = new_dict # the resultant data will be a nested dictionary

rotten_tomatoes_df = pd.DataFrame.from_dict(rotten_tomatoes, orient = "index")
rotten_tomatoes_df.to_csv("horror_rt.csv", header = True, index = True)
```

3. Data cleaning and processing

Before analysis, I would like to see if there any of the data has missing values, or needs further manipulation in order to derive greater meaning from it.

3.1 Cleaning TMDB data

I created a new dataframe from the csv so that I will not write over the raw data. In the next few steps, I will remove any movie's infoset if they have missing values or inappopriate data type. I would like to keep the original copy on hand, so that I know which data were removed in the process.

```
In [ ]: tmdb_raw_df = pd.read_csv("romance_tmdb.csv").copy()
```

Since it is a dataframe, the sum() method must be applied twice or 'na_sum' will be a pandas series object and not an integer. If there are less than five missing values, I will drop the corresponding rows). If 5 rows are dropped, we will only lose less than 5% of our data set - an acceptable level of loss.

```
In [ ]: na_sum = tmdb_raw_df.isna().sum().sum()
        check_na = "There are {sum} NA values in this dataframe.".format(sum = na_sum)
        if(na_sum < 5): tmdb_raw_df.dropna(inplace = True)
```

I converted all the dates from strings into datetime objects, so that we can analyse the data based on a time-series in the later parts of the project.

```
In [ ]: raw_dates = list(tmdb_raw_df['release_date'])
        dates = [datetime.strptime(str(date),'%Y-%m-%d') for date in raw_dates if date is not np.nan]
```

In the following section, I transformed the budget, revenue and runtime into np arrays. I then wrote a custom function "get_ratio" that will take divide an np array by another np array and format all the values as a float. I then used the function to get the profit, budget-to-runtime and revenue-over-runtime ratios.

```
In [ ]: box_office_raw_df = tmdb_raw_df.iloc[:,[4,5,6]]

        budget = np.array(box_office_raw_df.iloc[:,1])
        revenue = np.array(box_office_raw_df.iloc[:,2])
        runtime = box_office_raw_df.iloc[:,0]

        def get_ratio(dividend, divisor):
```

```
    raw_ratio = np.divide(dividend, divisor)
    lst = raw_ratio.tolist()
    float_lst = [round(value,2) for value in lst]
    return float_lst

pnl_ratio = get_ratio(revenue,budget)
budget_rt_ratio = get_ratio(budget,runtime)
revenue_rt_ratio = get_ratio(revenue,runtime)
```

In the code block above, I received a runtime warning, as some of the values are 0 and dividing by 0 generates an exception. In order to avoid this in the future, I stored the box office values as another dataframe. I then replaced all 0 values with numpy's nan and applied dropna. The resultant dataframe is significantly smaller as more than 30 rows are dropped. I saved the dataframe as a csv file

In [ ]:
```
box_office_dict = {
    "budget": budget,
    "revenue": revenue,
    "runtime": runtime,
    "pnl_ratio": pnl_ratio,
    "budget_rt_ratio" : budget_rt_ratio,
    "revenue_rt_ratio" : revenue_rt_ratio
}

box_office_df = pd.DataFrame(box_office_dict)
box_office_cleaned = box_office_df.replace(0, np.nan).dropna(axis = 'index')
box_office_cleaned.to_csv("romance_box_office.csv", index = False)
```

I took the newly calculated dates and ratio and combined it with the raw data, saving it as a cleaned csv. I dropped the first column of tmdb_raw_df as it contains a duplicate name column. I combined the new information with the raw dataframe and saved it into a csv file.

In [ ]:
```
new_tmdb_data = {
    "dates" : dates,
    "pnl_ratio": pnl_ratio,
    "budget_rt_ratio" : budget_rt_ratio,
    "revenue_rt_ratio" : revenue_rt_ratio
}

tmdb_raw_df.drop(columns = tmdb_raw_df.columns[0], axis = 1, inplace = True)
new_tmdb_df = pd.DataFrame(new_tmdb_data)
revised_tmdb_df = tmdb_raw_df.join(new_tmdb_df, how = 'inner') # could also use inner join
revised_tmdb_df.to_csv('romance_tmdb_cleaned.csv', index = False)
```

3.2 Extracting meaningful features from movie scripts

Since we have extracted the TMDB infoset for the movies and saved it in a CSV file, we can now start language processing on the corresponding scripts. This project focuses on finding commanality among the script and hence, we require common words used in the scripts and their sentiment polarity score.

In [ ]:
```
romance_titles_df = pd.read_csv("romance_titles.csv", header = None)
titles = list(romance_titles_df.iloc[0,:])
```

We use a for-loop to process each script. The code below takes above an hour to execute.

In [ ]:
```
scripts_analysis = {} # nested dictionary to contain processed data of all scripts

for title in titles:             # for each loop, one script file is opened
    script_title = title + '.txt'
    script = open(script_title)
    lines = script.read()     # read the file containing script
    script.close()            # close the file

    # declaring aliases
    analyzer = SentimentIntensityAnalyzer()
    tokens = nltk.tokenize.word_tokenize(lines)
    raw_polarity_score = analyzer.polarity_scores(lines)
    lemmatizer = WordNetLemmatizer()

    # only get words that are not a stopword and not a number
    filtered_tokens = [word for word in tokens if not word in stopwords.words() and word.isalnum()]
    # apply lowercase to all words
    lowered_case_ft = [word.lower() for word in filtered_tokens]

    # each part-of-speech tag contain the word and the tag: eg. "best" , ADJ (adjective)
    pairs = nltk.pos_tag(lowered_case_ft)
    verbs = [a for (a,b) in pairs if b == ('VBG' or 'VBD' or 'VBN' or 'VBP' or 'VBZ')] # verbs of all forms
    adjs = [a for (a,b) in pairs if b == ('JJ' or 'JJR' or 'JJS')] # adjective (comparative and superlative)

    # lemmatize each verb and adjectives
    verbs_lemmatized = [lemmatizer.lemmatize(word) for word in verbs]
    adjs_lemmatized = [lemmatizer.lemmatize(word) for word in adjs]

    # count the number of verbs and adjectives
    count_verbs = Counter(verbs_lemmatized)
    count_adjs = Counter(adjs_lemmatized)

    # get the most common verbs and adjectives
    common_verbs = [a for (a,b) in count_verbs.most_common(10)]
    common_adjs = [a for (a,b) in count_adjs.most_common(10)]

    # get the adjectives as a single string and apply sentiment analysis
    string_of_adj = "".join([" " + i for i in adjs]).strip()
    analyzer = SentimentIntensityAnalyzer()

    adjs_polarity_score = analyzer.polarity_scores(string_of_adj) # polarity based on adjectives

    # get the information as a dictionary
    script_analysis = {
    "common_verbs": common_verbs,
    "common_adjs": common_adjs,
    "raw_polarity_score": raw_polarity_score,
    "adjs_polarity_score" : adjs_polarity_score
    }

    # script_analysis will be a dictionary of dictoraries
    scripts_analysis[title] = script_analysis

scripts_analysis_df = pd.DataFrame.from_dict(scripts_analysis, orient = "index")
scripts_analysis_df.to_csv("romance_scripts_analysis.csv",index = True, header = True)
```

At the end of the process, I will have a csv file that contains polarity scores and common verbs and adjectives of the script.

3.3 Repeating for horror films

In this section, I will run the same code from 3.1 and 3.2 again to clean and process the data gathered for horror films. I will be referring to the csv files that contains teh data gathered on horror films.

In [ ]:
```
tmdb_raw_df = pd.read_csv("horror_tmdb.csv").copy()

# dropping rows with missing values
na_sum = tmdb_raw_df.isna().sum().sum()
check_na = "There are {sum} NA values in this dataframe.".format(sum = na_sum)
if(na_sum < 5): tmdb_raw_df.dropna(inplace = True)
```

```python
# creating datetime objects
raw_dates = list(tmdb_raw_df['release_date'])
dates = [datetime.strptime(str(date),'%Y-%m-%d') for date in raw_dates if date is not np.nan]

# creating np arrays
box_office_raw_df = tmdb_raw_df.iloc[:,[4,5,6]]

budget = np.array(box_office_raw_df.iloc[:,1])
revenue = np.array(box_office_raw_df.iloc[:,2])
runtime = box_office_raw_df.iloc[:,0]

def get_ratio(dividend, divisor):
    raw_ratio = np.divide(dividend, divisor)
    lst = raw_ratio.tolist()
    float_lst = [round(value,2) for value in lst]
    return float_lst

pnl_ratio = get_ratio(revenue,budget)
budget_rt_ratio = get_ratio(budget,runtime)
revenue_rt_ratio = get_ratio(revenue,runtime)

# creating a dictionary and saving the data into a csv
box_office_dict = {
    "budget": budget,
    "revenue": revenue,
    "runtime": runtime,
    "pnl_ratio": pnl_ratio,
    "budget_rt_ratio" : budget_rt_ratio,
    "revenue_rt_ratio" : revenue_rt_ratio
}

# creating dataframe with only box office figures
box_office_df = pd.DataFrame(box_office_dict)
box_office_cleaned = box_office_df.replace(0, np.nan).dropna(axis = 'index')
box_office_cleaned.to_csv("horror_box_office.csv", index = False)

new_tmdb_data = {
    "dates" : dates,
    "pnl_ratio": pnl_ratio,
    "budget_rt_ratio" : budget_rt_ratio,
    "revenue_rt_ratio" : revenue_rt_ratio
}

tmdb_raw_df.drop(columns = tmdb_raw_df.columns[0], axis = 1, inplace = True)
```

```
new_tmdb_df = pd.DataFrame(new_tmdb_data)
revised_tmdb_df = tmdb_raw_df.join(new_tmdb_df, how = 'inner') # could also use inner join
revised_tmdb_df.to_csv('horror_tmdb_cleaned.csv', index = False)
```

Extracting meaningful features from movie scripts for horror films

In [ ]:
```
horror_titles_df = pd.read_csv("horror_titles.csv", header = None)
titles = list(horror_titles_df.iloc[0,:])
```

In [ ]:
```
scripts_analysis = {} # nested dictionary to contain processed data of all scripts

for title in titles:             # for each loop, one script file is opened
    script_title = title + '.txt'
    script = open(script_title)
    lines = script.read()    # read the file containing script
    script.close()           # close the file

    # declaring aliases
    analyzer = SentimentIntensityAnalyzer()
    tokens = nltk.tokenize.word_tokenize(lines)
    raw_polarity_score = analyzer.polarity_scores(lines)
    lemmatizer = WordNetLemmatizer()

    # only get words that are not a stopword and not a number
    filtered_tokens = [word for word in tokens if not word in stopwords.words() and word.isalnum()]
    # apply lowercase to all words
    lowered_case_ft = [word.lower() for word in filtered_tokens]

    # each part-of-speech tag contain the word and the tag: eg. "best" , ADJ (adjective)
    pairs = nltk.pos_tag(lowered_case_ft)
    verbs = [a for (a,b) in pairs if b == ('VBG' or 'VBD' or 'VBN' or 'VBP' or 'VBZ')] # verbs of all forms
    adjs = [a for (a,b) in pairs if b == ('JJ' or 'JJR' or 'JJS')] # adjective (comparative and superlative)

    # lemmatize each verb and adjectives
    verbs_lemmatized = [lemmatizer.lemmatize(word) for word in verbs]
    adjs_lemmatized = [lemmatizer.lemmatize(word) for word in adjs]

    # count the number of verbs and adjectives
    count_verbs = Counter(verbs_lemmatized)
    count_adjs = Counter(adjs_lemmatized)

    # get the most common verbs and adjectives
    common_verbs = [a for (a,b) in count_verbs.most_common(10)]
    common_adjs = [a for (a,b) in count_adjs.most_common(10)]

    # get the adjectives as a single string and apply sentiment analysis
    string_of_adj = "".join([" " + i for i in adjs]).strip()
    analyzer = SentimentIntensityAnalyzer()

    adjs_polarity_score = analyzer.polarity_scores(string_of_adj) # polarity based on adjectives

    # get the information as a dictionary
    script_analysis = {
    "common_verbs": common_verbs,
    "common_adjs": common_adjs,
    "raw_polarity_score": raw_polarity_score,
    "adjs_polarity_score" : adjs_polarity_score
    }

    # script_analysis will be a dictionary of dictoraries
    scripts_analysis[title] = script_analysis

scripts_analysis_df = pd.DataFrame.from_dict(scripts_analysis, orient = "index")
scripts_analysis_df.to_csv("horror_scripts_analysis.csv",index = True, header = True)
```

4. Exploratory data analysis

We have cleaned our dataset and can now proceed to gleam some understanding of the dataset. Most of the visualisations will utilise seaborn to plot the figures.

In [4]:
```
# set seaborn stype for this section
sns.set_theme(style="dark") # dark background
```

4.1 Market analysis for Romance and Horror films

In this section, I will use descriptive statistics to summarise the box-office values for Romance and Horror films. Then, I will compare the box-office values of romance and horror films to see if the two markets are equally profitable. I will also study the strength and direction of relationships between the box office values.

In [4]:
```
# read romance and horror box office figures into pandas dataframes
romance_box_df = pd.read_csv('romance_box_office.csv', header = 0)
```

```
horror_box_df = pd.read_csv('horror_box_office.csv', header = 0)

# combine romance and horror dataset (for plotting of graphs)
concatenated_box_df = pd.concat([romance_box_df.assign(dataset='romance'),horror_box_df.assign(dataset='horror')])
```

4.1.1 Descriptive statistics of individual industries

In order to get an overview of the two markets, I created a dataframe 'box_office_means' which shows the mean budget, revenue and runtime for each genre of movies. The third column in the dataframe, 'percent_diff', shows the calculated percentage difference between the two means.

In [5]:
```
pd.options.display.float_format = '{:,.2f}'.format  # formats values to have 2 decimal places.

# get the mean of budget, revenue and runtime
romance_mean = romance_box_df[['budget','revenue','runtime']].mean()
horror_mean = horror_box_df[['budget','revenue','runtime']].mean()

# concatenate romance and horror data
box_office_means = pd.concat([romance_mean, horror_mean], axis = 1)
box_office_means.columns = ['romance','horror']

# calculate percentage difference
box_office_means['percent_diff'] = (box_office_means['romance'] - box_office_means['horror'])/ box_office_means['horro
box_office_means
```

Out[5]:

	romance	horror	percent_diff
budget	34,456,970.64	28,005,610.69	23.04
revenue	146,703,948.96	110,629,572.36	32.61
runtime	116.22	105.14	10.54

From the table above, it is clear that romance films typically have a higher budget, of 23% more than horror films. Romance films also bring in more revenue, at an average of 32% more than horror films. On average, romance films run 11 minutes longer than horror films.

This discriptive statistics suggest that romance films are cost-heavy, with a higher budget and longer runtime, but may bring in a higher revenue compared to horror films.

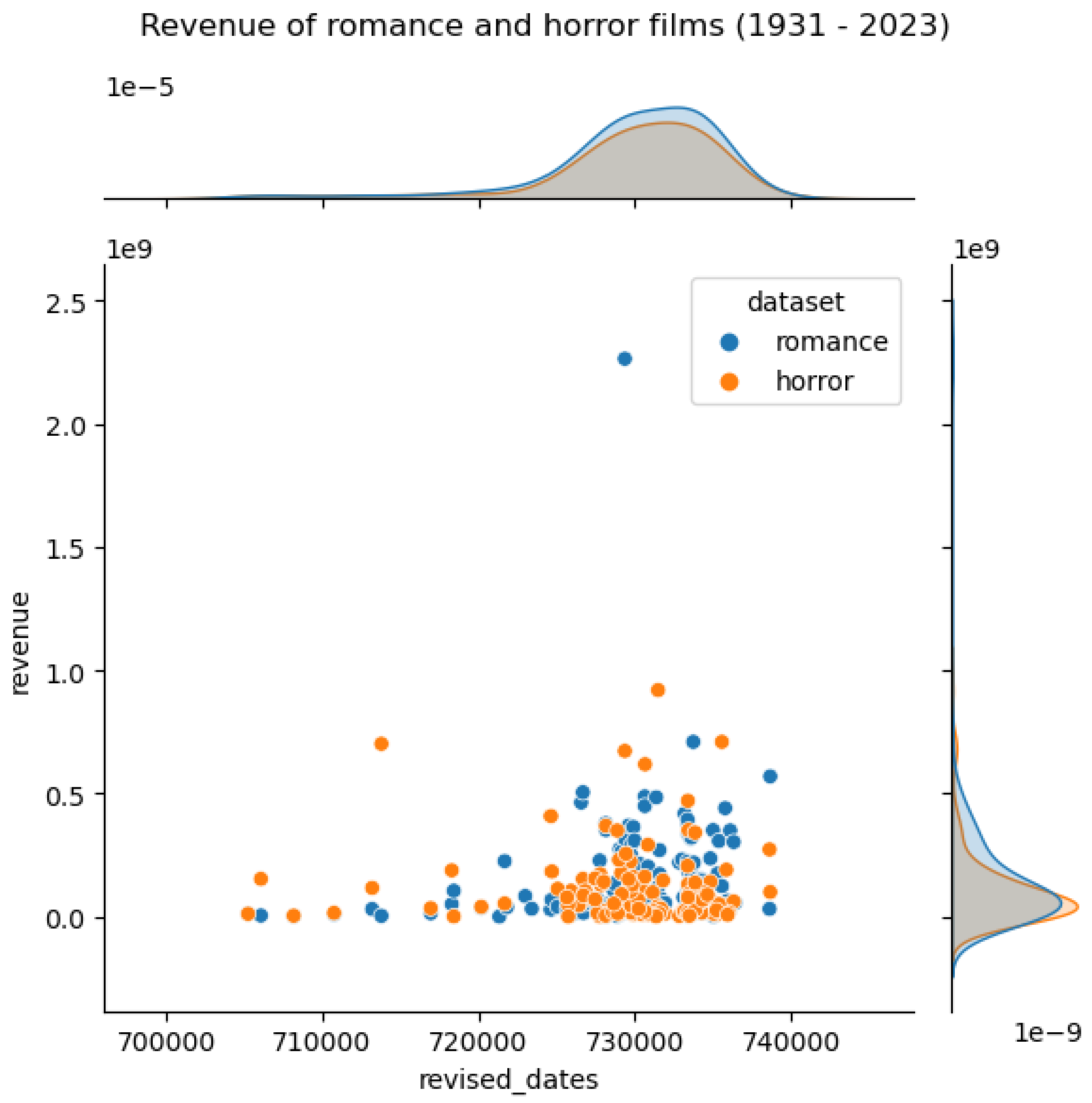
4.1.2 Temporal trends

I would like to explore how movie revenue and budget have evolved for the two genres over the past nine decades. I have noticed that seaborn does not take datetime objects as appropriate data type for its axes. Hence, I have to first convert all datetime values into Gregorian ordinal value.

In [6]:
```
# sorting by dates
r_dates = romance_box_df.sort_values('dates')
h_dates = horror_box_df.sort_values('dates')
```

In [7]:
```
# changing date values to ordinal values
r_dates['revised_dates'] = pd.to_datetime(r_dates['dates']).apply(lambda datetime : datetime.toordinal())
h_dates['revised_dates'] = pd.to_datetime(r_dates['dates']).apply(lambda datetime : datetime.toordinal())

# combine romance and horror data into a single dataframe
concatenated = pd.concat([r_dates.assign(dataset='romance'),h_dates.assign(dataset='horror')])
ax = sns.jointplot(x = 'revised_dates', y = 'revenue', data = concatenated, hue = 'dataset')
ax.fig.suptitle("Revenue of romance and horror films (1931 - 2023)")
ax.fig.tight_layout()   # making rom to display the title
```

```
/var/folders/pc/fblgrjpn2ng04364nyk3lgjc0000gn/T/ipykernel_42674/1407834942.py:9: UserWarning: The figure layout has
changed to tight
  ax.fig.tight_layout()   # making rom to display the title
```



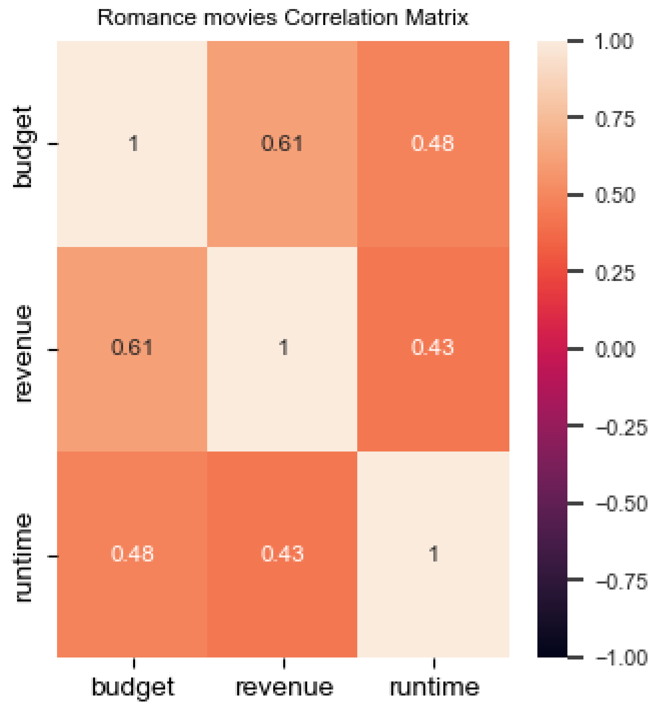
The average revenue for romance films is higher than horror films (bell curve to right), with romance films seeing higher revenues, while horror films see a larger standard deviation. Romance films tend to perform more consistently, with a steeper bell curve, and all of the data points cluttered around the same range and only one obvious outlier. Overall, revenue has increased for both genre of films over the years.

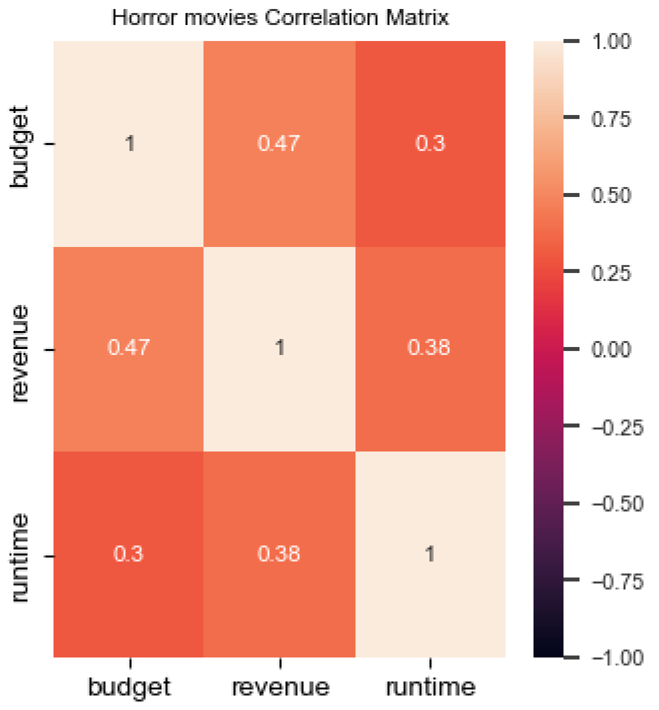
4.1.3 Correlation matrix of box-office numbers

My hypothesis is that there is a positive relationship between budget, runtime and revenue. To prove my hypothesis, I created a correlation matrix for each genre and plotted a heatmap.

In [8]:
```python
# creating correlation matrix dataframe
r_corr_matrix = romance_box_df.iloc[:,[1,2,3]].corr()
h_corr_matrix = horror_box_df.iloc[:,[1,2,3]].corr()

fig, axes = plt.subplots(1,2,figsize=(8,4)) # size of the subplot

sns.set(font_scale = 0.7) # text size

# minimum and maximum of -1 and 1 to anchor the colourmap, setting annotation to true
sns.heatmap(r_corr_matrix, vmin = -1, vmax = 1, annot = True, ax = axes[0])
sns.heatmap(h_corr_matrix, vmin = -1, vmax = 1, annot = True, ax = axes[1])

axes[0].set_title("Romance movies Correlation Matrix")
axes[1].set_title("Horror movies Correlation Matrix")
```

Out[8]:
```
Text(0.5, 1.0, 'Horror movies Correlation Matrix')
```





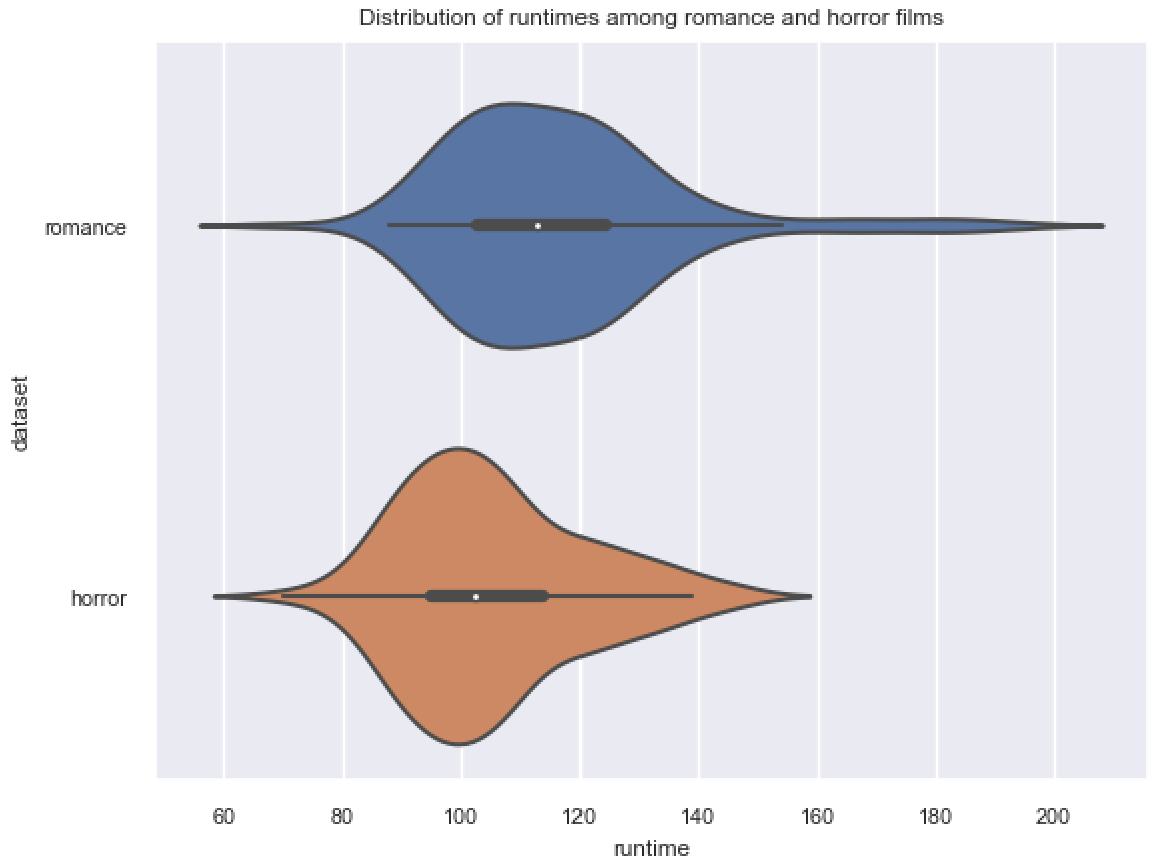
From the correlation matrix above, it is obvious that there is a positive correlation between runtime, revenue and budget for both romance and horror movies. There is a higher correlation between revenue and budget for romance movies, as compared to horror movies. This suggests that a higher budget will have a bigger impact in romance films than horror films. A longer runtime also have a more significant impact on romance movies than horror ones.

4.1.4 Runtimes

While it is expected for a high-budget film to garner higher revenues, it is interesting to see in the correlation matrix that runtime also has a positive correlation with revenue. I would like to explore the distribution of runtime within each genre to further investigate this relationship.

```
In [9]: concatenated_box_office = pd.concat([romance_box_df.assign(dataset='romance'),horror_box_df.assign(dataset='horror')]
        ax = sns.violinplot(data = concatenated_box_df, x = 'runtime', y = 'dataset')
        ax.set_title("Distribution of runtimes among romance and horror films")
```

```
Out[9]: Text(0.5, 1.0, 'Distribution of runtimes among romance and horror films')
```



Romance films seeem to have a longer run time, with a median between 100 and 120 minutes, and the longest taking more than 3 hours. The plot shows that romance films have a more elongated distribution, which suggests a long-tail distribution of films, with some extreme long runtimes. Horror films have more typical runtimes, with most running around 100 minutes.

4.1.5 Top 50 highest grossing films in this dataset

Based on the data above, runtime has a positive relationship with revenue and romance films have a longer runtime. I am interested to see if the top 10 highest grossing films in the dataset have long run-times.

```
In [10]: top_films = concatenated_box_office.nlargest(50,'revenue') # run to get the dataframe
         top_films.head(10) # get the top 10
```

Out[10]:

	dates	budget	revenue	runtime	pnl_ratio	budget_rt_ratio	revenue_rt_ratio	dataset
28	1997-11-18	200,000,000.00	2,264,162,353.00	194.00	11.32	1,030,927.84	11,670,939.96	romance
75	1993-06-11	63,000,000.00	920,100,000.00	127.00	14.60	496,062.99	7,244,881.89	horror
22	2009-11-18	50,000,000.00	709,827,462.00	131.00	14.20	381,679.39	5,418,530.24	romance
24	2009-11-18	50,000,000.00	709,827,462.00	131.00	14.20	381,679.39	5,418,530.24	horror
76	2017-09-06	40,000,000.00	701,800,000.00	135.00	17.55	296,296.30	5,198,518.52	horror
28	1999-08-06	40,000,000.00	672,800,000.00	107.00	16.82	373,831.78	6,287,850.47	horror
68	1997-05-23	73,000,000.00	618,638,999.00	129.00	8.47	565,891.47	4,795,651.16	horror
77	2023-05-18	297,000,000.00	569,600,000.00	135.00	1.92	2,200,000.00	4,219,259.26	romance
108	1990-07-13	22,000,000.00	505,000,000.00	127.00	22.95	173,228.35	3,976,377.95	romance
36	2001-05-18	60,000,000.00	487,900,000.00	90.00	8.13	666,666.67	5,421,111.11	romance

In [11]:

```
top_films['dataset'].value_counts() # get count of romance/horror films within 50 films
```

Out[11]:

```
dataset
romance    35
horror     15
Name: count, dtype: int64
```

It is worthy to note that the top 10 films are split equally between romance and horor, with horror films taking the top 7 spots.

However, there is significantly more romance films than horror films in the top 50 highest grossing films in this dataset. This suggests that romance films are significantly more likely to generate a higher revenue.

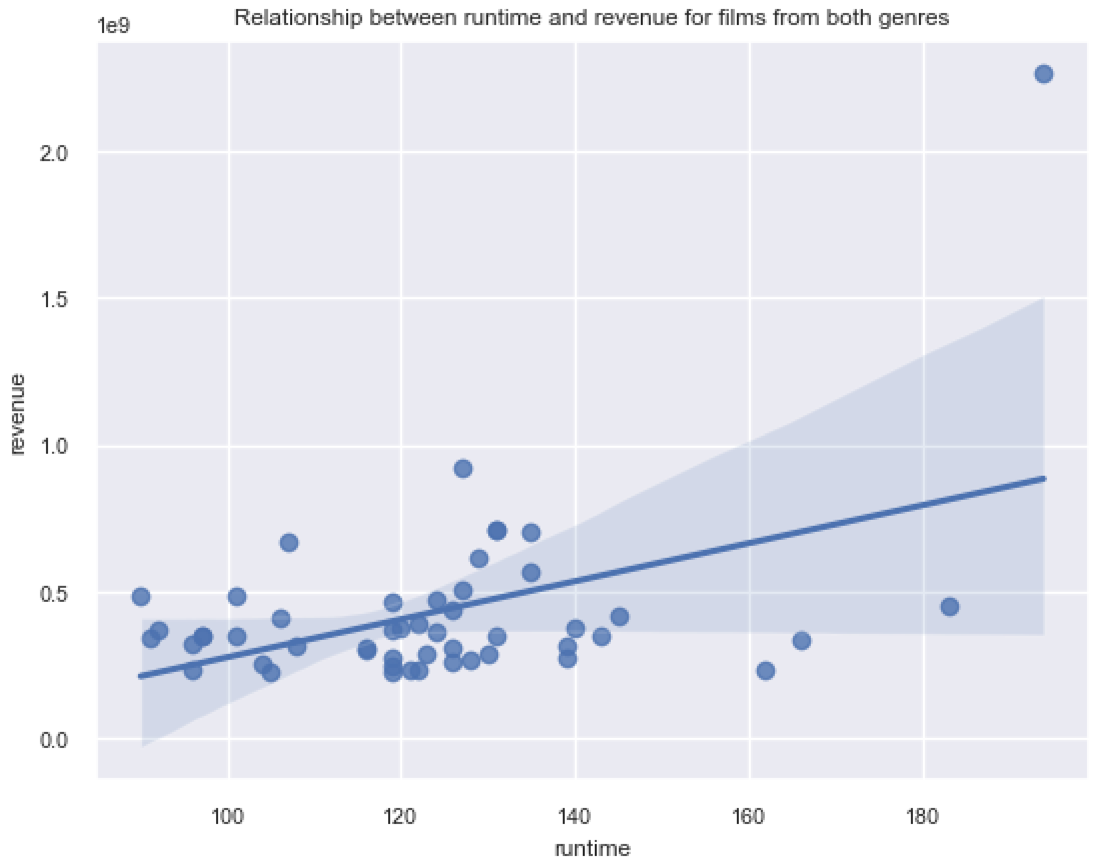
From the dataframe, it is evident that most of the films last longer than 120 minutes. I decided to do a scatter plot with regression line for runtime against revenue to see the impact of runtime on revenue for the top grossing films.

In [12]:

```
ax = sns.regplot(top_films, x = "runtime", y = "revenue")
ax.set_title("Relationship between runtime and revenue for films from both genres")
```

Out[12]:

```
Text(0.5, 1.0, 'Relationship between runtime and revenue for films from both genres')
```



From the plot above, it is obvious that the general trend is for films with longer runtime to have a higher revenue.

4.2 Analysis of audience ratings for romance and horror films.

In this section, I will be analysing the datasets retrieved from tmdb and rotten tomatoes. When I initially created the combined dataframe without this keyword, the resultant df has indexes with duplicate values, which raises a 'ValueError: cannot reindex on an axis with duplicate labels' error message. To prevent this, I concatenated the dataframes together, setting the 'ignore_index' keyword argument to true.

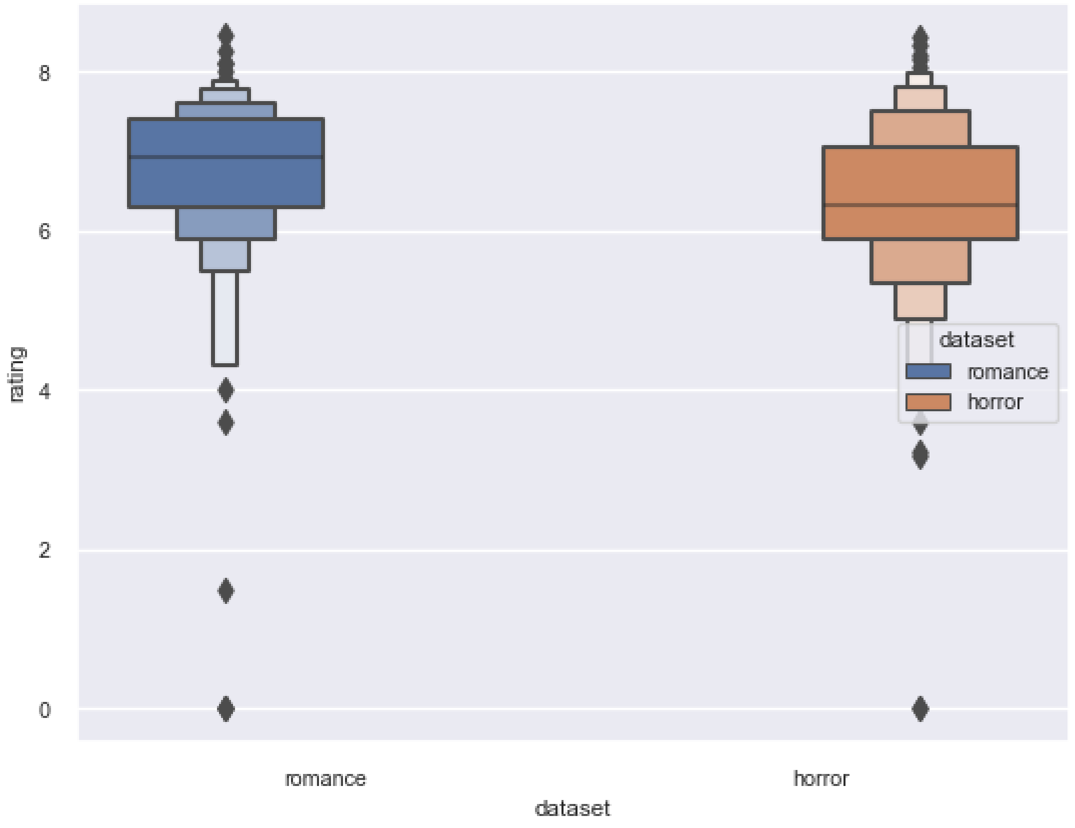
In [13]:
```
romance_tmdb_df = pd.read_csv('romance_tmdb_cleaned.csv', header = 0)
romance_rt_df = pd.read_csv('romance_rt.csv', header = 0)
horror_tmdb_df = pd.read_csv('horror_tmdb_cleaned.csv', header = 0)
horror_rt_df = pd.read_csv('horror_rt.csv', header = 0)
romance_df = romance_tmdb_df.join(romance_rt_df, how = 'inner')
```

```
horror_df = horror_tmdb_df.join(horror_rt_df, how = 'inner')

concatenated_ratings_df = pd.concat([romance_df.assign(dataset='romance'),horror_df.assign(dataset='horror')], ignore_
```

4.2.1 Ratings for romance and horror films

```
In [14]: sns.boxenplot(data = concatenated_ratings_df, x = 'dataset', y = 'rating', hue = 'dataset')
```

```
Out[14]: <Axes: xlabel='dataset', ylabel='rating'>
```

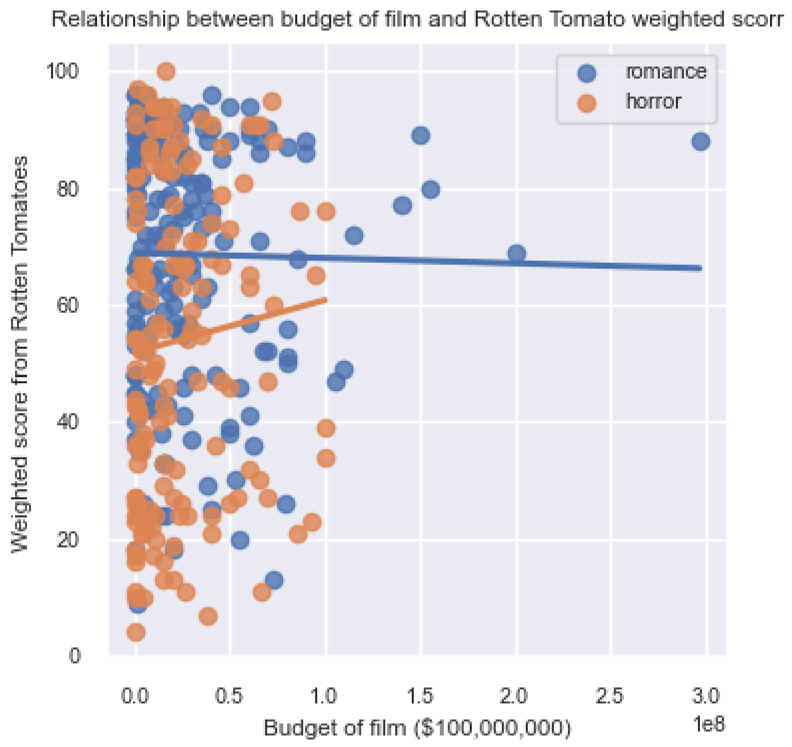


From the categorical plot above, we can see that the rating of romance film is clustered around the score of 6 - 8 out of 10 while horror movies see slightly lower scores. Horror movies also have a wider distribution than romance movies - which suggests that audience rating is less predictable for horror films.

4.2.2 Budget vs Rating

It is intuitive to think that a higher budget will translate to high audience ratings for films, since a more costly production will have a bigger team and better materials. I would like to see if that is true for both romance and horror films, and whether the positive coorelation is a strong one.

In [15]:
```
fig, ax = plt.subplots(figsize=(4, 4))
# add the plots for each dataframe
sns.regplot(data=romance_df, x='budget', y='weighted_score',  fit_reg=True, ci=None, ax=ax, label='romance')
sns.regplot(data=horror_df, x='budget', y='weighted_score',  fit_reg=True, ci=None, ax=ax, label='horror')

ax.set(ylabel='Weighted score from Rotten Tomatoes', xlabel='Budget of film ($100,000,000)')
ax.set_title("Relationship between budget of film and Rotten Tomato weighted scorr")
ax.legend()
plt.show()
```



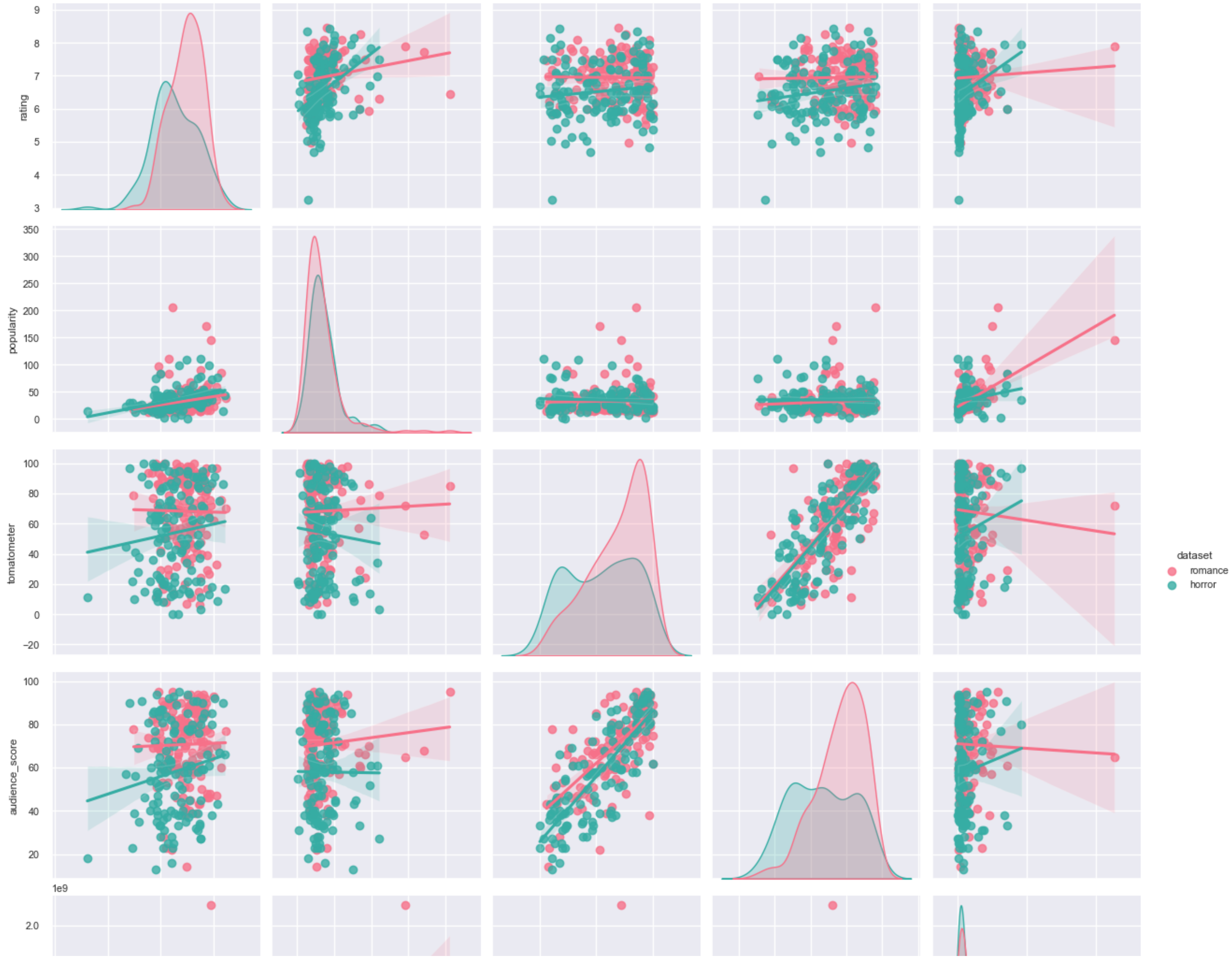
The linear regression model fit provides a very surprising answer. We can see the relationship between budget of film and ratings from Rotten tomatoes is positive for horror films but negative for romance films. This suggests that audience do not find high-budgetted romance films more attractive than low-budget ones, possibly because a good romance films do not depend on a big cast, CGI, or costly marketing. On the other hand, there is a noticable positive relationship between budget and ratings for horror films - perhaps a truly terrifying film require a higher quality special effects team.

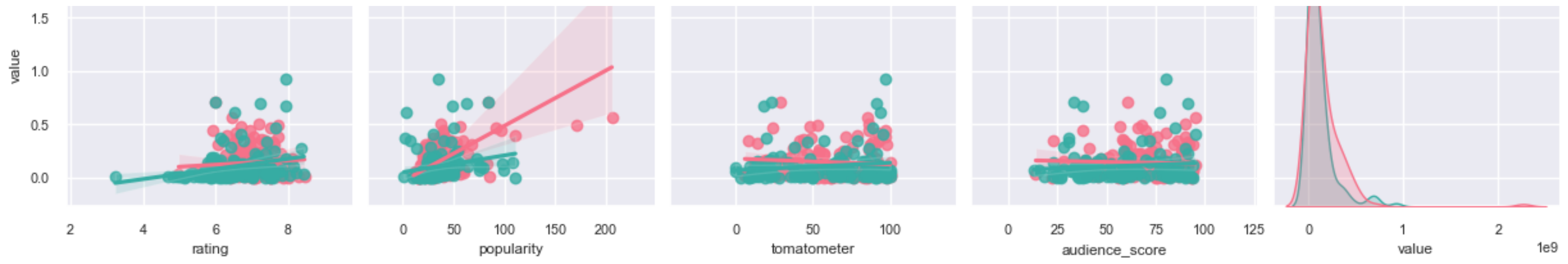
4.2.3 Ratings from two different sites and revenue

I would like to see if there are any major disprecancies between ratings from TMDB and Rotten Tomatoes. I would also like to see the impact of ratings on revenue. In order to check if the ratings from two sites differ, I chose to use a pair plot.

```
In [27]: # ratings_df = concatenated_ratings_df.melt()
         ratings_df = pd.melt(concatenated_ratings_df,
                             id_vars = ['rating','popularity','tomatometer','audience_score','dataset'],
                             value_vars = ['revenue'])

         # using inverted mask to get only rows that do not have 0 values
         ratings_df_cleaned = ratings_df[~(ratings_df[['value']] == 0).any(axis=1)]

         ax = sns.pairplot(ratings_df_cleaned, hue = 'dataset', markers = None, palette= 'husl', kind = 'reg')
```

```
/Users/main/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has change
d to tight
  self._figure.tight_layout(*args, **kwargs)
```


rating
popularity
tomatometer
audience_score
1e9
dataset
romance
horror
9
8
7
6
5
4
3
350
300
250
200
150
100
50
0
100
80
60
40
20
0
−20
2.0



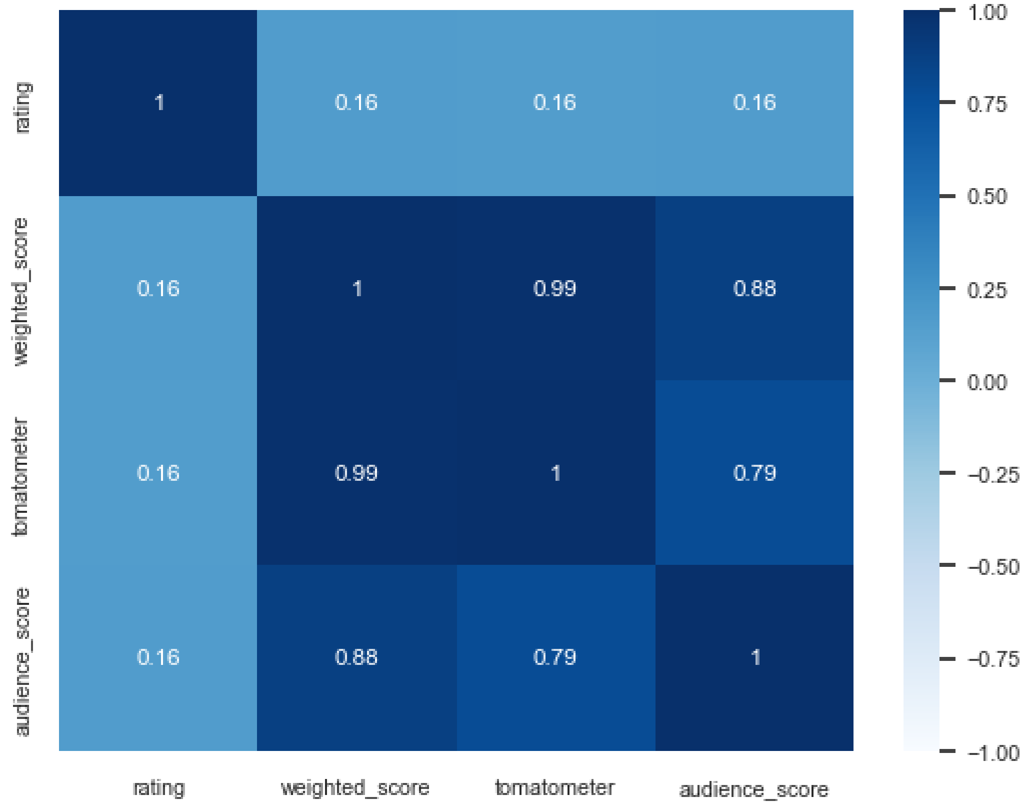
From this pairplot, we can see that there is largely a positive relationship between the ratings from TMDB and the scores from Rotten Tomatoes, which suggests that the ratings are well-balanced from each site.

Based on the bottom row, there is a slight positive relationship (based on Rotten Tomato), and a strong positive relationship (based on TMDB's popularity score) between revenue and ratings. The only disprecancy is found between the poplarity score from TMDB and tomatomer for horror movies, with a slight inverse relationship.

4.2.4 Discrepencies between TMDB and Rotten Tomatoes audience scores

In order to make sure that there is no major contradiction between the audience ratings from TMDB and Rotten Tomatoes, I decided to create a correlation matrix and plot a heat map.

In [17]:
```
ratings_corr_df = concatenated_ratings_df.iloc[:,[6,17,18,19]].corr()
ax = sns.heatmap(ratings_corr_df, vmin = -1, vmax = 1, annot = True, cmap="Blues")
```



From this heatmap, we can see that none of the scores have an inverse relationship with another score. This means that the ratings gathered from the two sites do not contradict each other and showcases the same attitude towards an individual film.

4.3 Analysis of language used in scripts

In the following section, I will be using the data gained from the sentiment analysis done in Section 3 to explore the language used in the movie scripts.

For the purpose of this exercise, I created dataframes from the sentiment analysis data stored in the csv.

```
In [18]: romance_scripts_df = pd.read_csv('romance_scripts_analysis.csv', header = 0)
         horror_scripts_df = pd.read_csv('horror_scripts_analysis.csv', header = 0)
```

4.3.1 Common Verbs and Adjectives

In the following code blocks, I will be looking at all the common verbs and adjectives among the scripts and seeing if there are any similarities between the two genres.

Each movie has two sets of top 10 most common words in the script - for verbs and adjectives. In order to visualise the common verbs among all of the movies, I defined a function to collate all the words into a single string. From this single string, I plotted a word cloud to represent the collated data.

In [19]:
```
def words_to_string(col, to_remove):
    words_collated = []
    for lst in col:
        words = lst.split(',')
        cleaned_lst = [word.strip("[' ']") for word in words]
        cleaned_lst_meaningful = [word for word in cleaned_lst if not word in to_remove]
        words_collated.extend(cleaned_lst_meaningful)
        words_as_str = ' '.join(words_collated)

    return words_as_str
```

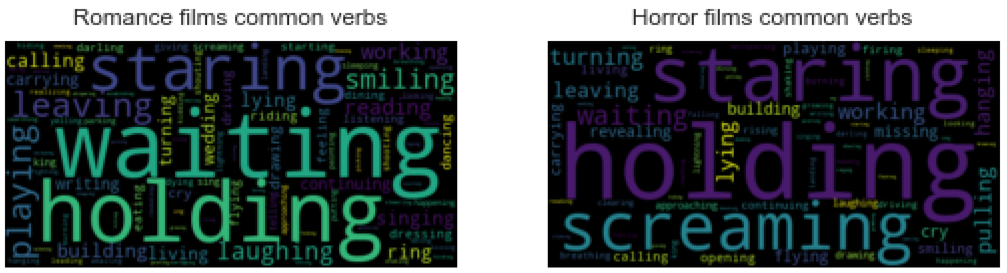
Common Verbs The initial word cloud formed have words that are either common actions that do not necessarily lend any meaning to the script content or expletives not appropriate for a data analysis project. I have removed these verbs from the string.

In [20]:
```
romance_common_verbs = romance_scripts_df.get("common_verbs")
horror_common_verbs = horror_scripts_df.get("common_verbs")

verbs_to_exclude = ['standing','coming','watching','walking','running','sitting','making','taking','moving','wearing',

romance_verbs_str = words_to_string(romance_common_verbs, verbs_to_exclude)
horror_verbs_str = words_to_string(horror_common_verbs, verbs_to_exclude)


plt.subplot(1,2,1)
plt.title("Romance films common verbs")
wordcloud = WordCloud().generate(romance_verbs_str)
plt.axis("off")
plt.imshow(wordcloud, interpolation='bilinear')

plt.subplot(1,2,2)
plt.title("Horror films common verbs")
```

```
wordcloud = WordCloud().generate(horror_verbs_str)
plt.axis("off")
plt.imshow(wordcloud, interpolation='bilinear')
```

Out[20]:

```
<matplotlib.image.AxesImage at 0x167768c90>
```



It is interesting to see that both romance films and horror films share common verbs, such as 'waiting' and 'holding'. These actions, while common among genres, can hold different connotations in different contexts. The verbs gathered are mainly from stage directions and scene setting included in the scripts - which means that it is up to the actors' discretion as to how to deliver these actions.

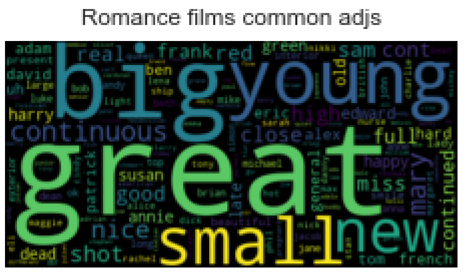
While words such as 'screaming' and 'pulling' are among more common words for romance films, they are lacking in horror films. On the other hand, 'playing' and 'laughing' are prominent among romance films but not in horror films.

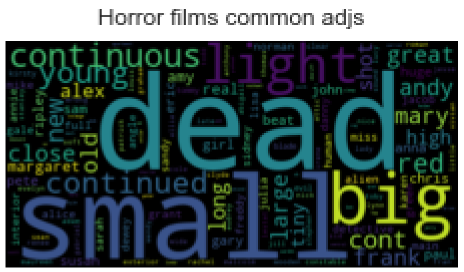
Common Adjectives

In [26]:

```
romance_common_adjs = romance_scripts_df.get("common_adjs")
horror_common_adjs = horror_scripts_df.get("common_adjs")

adjs_to_exclude = ['ext','int','white','black','open','front','table','oh']

romance_adjs_str = words_to_string(romance_common_adjs, adjs_to_exclude)
horror_adjs_str = words_to_string(horror_common_adjs, adjs_to_exclude)


plt.subplot(1,2,1)
plt.title("Romance films common adjs")
wordcloud = WordCloud().generate(romance_adjs_str)
plt.axis("off")
plt.imshow(wordcloud, interpolation='bilinear')


plt.subplot(1,2,2)
plt.title("Horror films common adjs")
wordcloud = WordCloud().generate(horror_adjs_str)
plt.axis("off")
plt.imshow(wordcloud, interpolation='bilinear')
```

Out[26]:
```
<matplotlib.image.AxesImage at 0x1679b6010>
```





From the word clouds, it is evident that the scripts, regardless of genres, are focused on physical characteristics such as size. eg. big, tiny, small. While the adjectives in romance films have more positive connotations, such as 'nice', 'happy' and 'good', horror films has the word 'dead' and 'small' as its two most common adjectives. This is as to be expected of the two opposing genres of films.

4.3.2 Sentiment Polarity Scores

In the following code block, I will be looking at how the polarity scores are distributed in each genre of film. Polarity scores lie between -1 and 1, where -1 indicates negative sentiment (gathered from words such as bad or dead) and 1 indicates positive sentiment (gathered from words such as great, nice and laughing). From the analysis of common verbs and adjectives above, I predict that horror movies will have a low polarity score while romance films will have a high polarity score.

The compound score is stored as a dictionary in the raw_polarity_score and adjs_polarity_score columns.

Since the data is acquired as a JSON object and saved as a CSV, the compound scores are stored in strings of this format: "{'neg': 0.1, 'neu': 0.783, 'pos': 0.117, 'compound': 0.9999}"

I have to define a function to isolate the compound scores and add them in as new columns. In order to do this, I used regex validation to get the string before the last comma.

In [32]:
```
def get_compound(s):
    before_comma = re.findall("(.*),[^,]*$",s)[0] # Find string before last comma
    compound = s.replace(before_comma,'').split(':')[-1].strip('}')
    return compound

merged_romance_df = romance_scripts_df.merge(romance_df, how = "inner")
romance_compound = merged_romance_df['raw_polarity_score'].apply(get_compound)
romance_adj_compound = merged_romance_df['adjs_polarity_score'].apply(get_compound)


merged_horror_df = horror_scripts_df.merge(horror_df, how = "inner")
horror_compound =  merged_horror_df['raw_polarity_score'].apply(get_compound)
horror_adj_compound =  merged_horror_df['adjs_polarity_score'].apply(get_compound)


romance_polarity_dict = {
    'raw_polarity' : romance_compound,
    'adj_polarity': romance_adj_compound
}

horror_polarity_dict = {
    'raw_polarity' : horror_compound,
    'adj_polarity' : horror_adj_compound
}

full_romance = merged_romance_df.assign(**romance_polarity_dict)
full_horror = merged_horror_df.assign(**horror_polarity_dict)
```

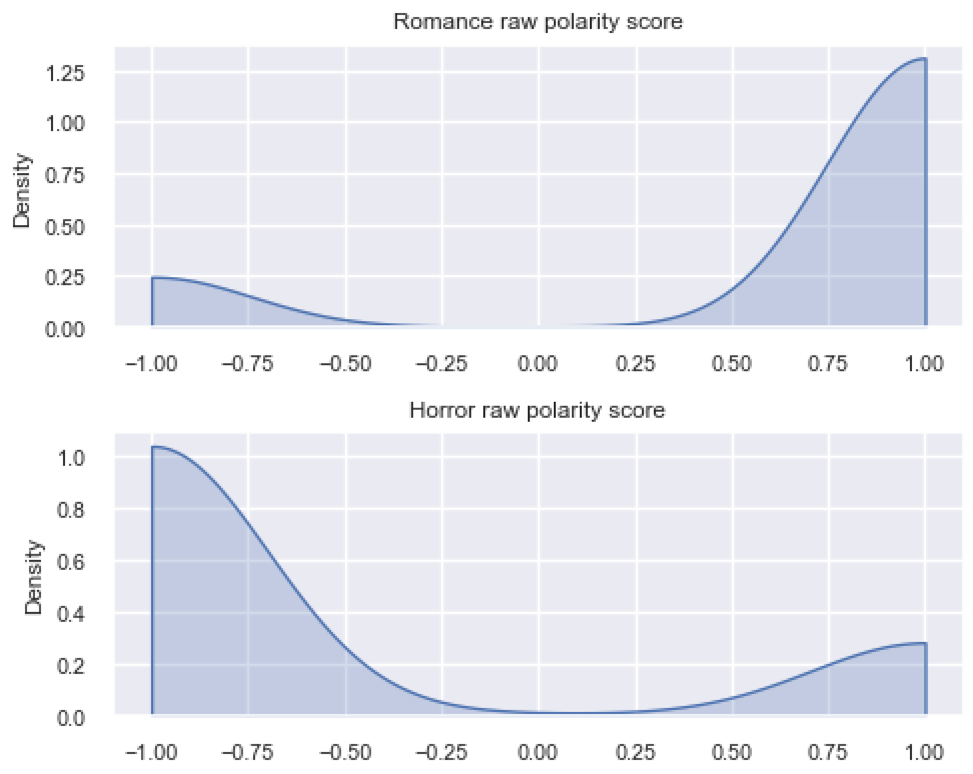
In [33]:
```
full_df = pd.concat([full_romance.assign(dataset='romance'),full_horror.assign(dataset='horror')])
full_df.to_csv('full_dataset.csv', header = True, index = True)
```

Raw polarity scores

The raw polarity scores is calculated indiscriminately based on all of the words of the script. I predict that romance films will have a high score while horror films will have a low score.

In [34]:
```
romance_polarity_scores = romance_scripts_df.get("raw_polarity_score")
horror_polarity_scores = horror_scripts_df.get("raw_polarity_score")

# Compound score is in a dictionary
def get_average_polarity(scores):
    scores_values = []
    compound_scores = scores.to_dict()
    for item in compound_scores.items():
        compound_value = ast.literal_eval(item[1])['compound']
```

```
            scores_values.append(compound_value)
        return np.array(scores_values)


romance_polarity = get_average_polarity(romance_polarity_scores)
horror_polarity = get_average_polarity(horror_polarity_scores)

fig, (ax1,ax2) = plt.subplots(2,1, figsize=(5,4))
sns.kdeplot(romance_polarity, ax=ax1, cut = 0, fill = True)
sns.kdeplot(horror_polarity, ax=ax2, cut = 0, fill = True)
ax1.set_title("Romance raw polarity score")
ax2.set_title("Horror raw polarity score")
plt.tight_layout()
```



I used a kernel density estimate (KDE) plot to visualise the distribution of the raw polarity scores in romance movies versus horror movies respectively. As expected, romance movies tend to have a high polarity score while horror movies tend to have a negative polarity score. The two genres tend to have very extreme sentiments, with a significant amount of films at the opposite end.

4.3.3 Polarity scores on rating

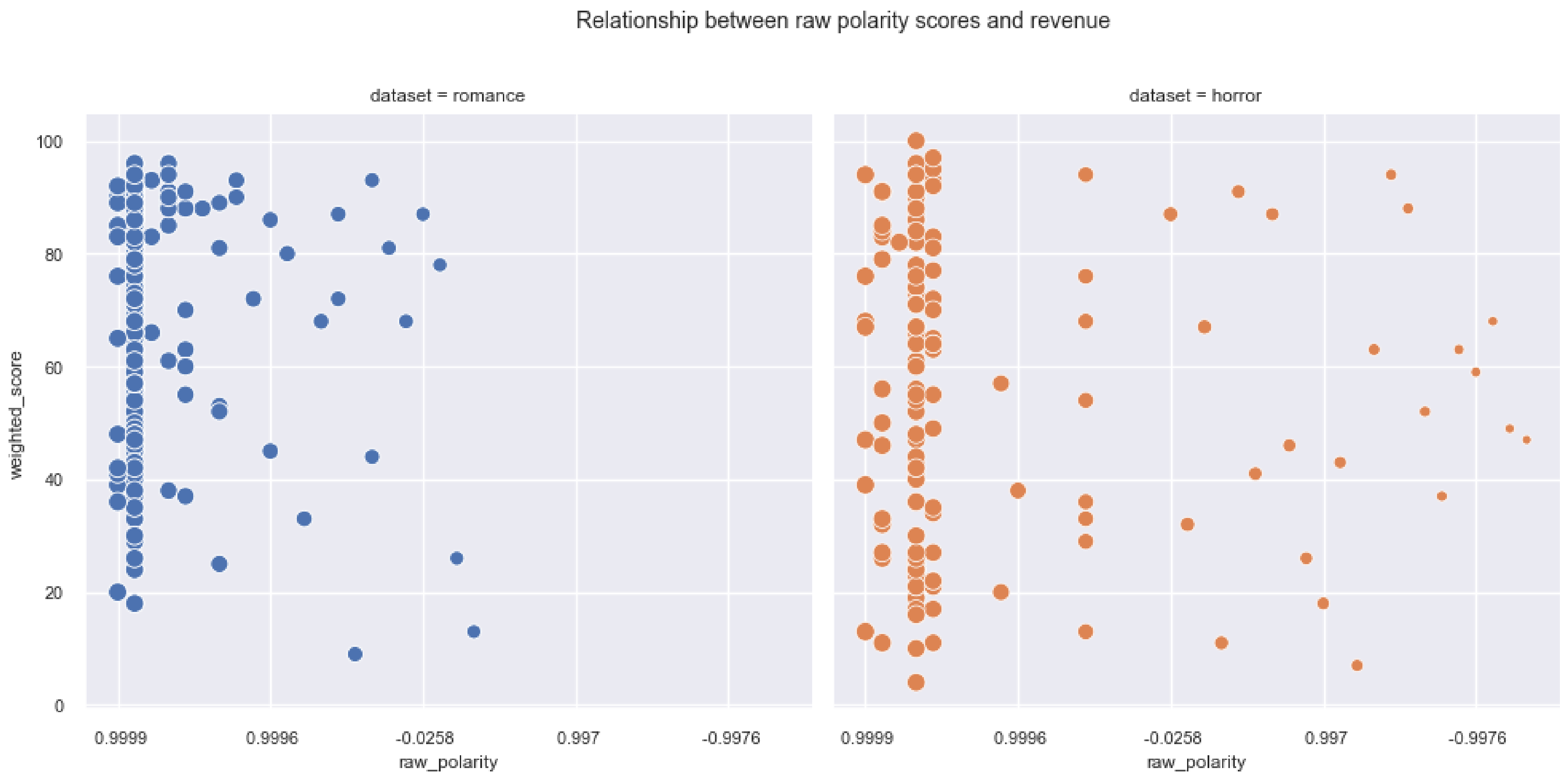
In order to explore the relationship between polarity score and ratings, I have to combine the two dataframes of data from:

- box-office values and audience ratings from TMDB and Rotten Tomatoes
- sentiment Analysis on scripts from IMSDB

In [35]:
```
g = sns.relplot(data = full_df, x = 'raw_polarity', y = 'weighted_score', col = 'dataset',
                    size = 'raw_polarity', hue = 'dataset')

g.set_xticklabels(labels = None, step = 3)
g.legend.remove()
g.fig.subplots_adjust(top = 0.85)
g.fig.suptitle('Relationship between raw polarity scores and revenue')
```

```
/Users/main/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has change
d to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[35]:
```
Text(0.5, 0.98, 'Relationship between raw polarity scores and revenue')
```



There seem to be no strong relationship between the polarity scores and the revenue of a film within both the romance and horror genres. This suggests that the strengh of 'positive' and 'negative' sentiments do not have a major sway over how much the audience likes a movie.

In the code below, I created a full dataset (of box-office numbers, ratings and sentiment analysis) and saved it into a CSV flat file format for ease of use for further studies.

```
In [36]: full_df = pd.concat([full_romance.assign(dataset='romance'),full_horror.assign(dataset='horror')])
         full_df.to_csv('full_dataset.csv', header = True, index = True)
```

5. Summary

Based on the exploratory data analysis above, we can tell that romance films tend to have a higher budget and a higher revenue, but horror films tend to perform more consistently in box-office. Romance and horror films do not share a huge difference in ratings and audience scores. TMDB and Rotten Tomatoes provide similar ratings and revenue has a positive relationship with ratings in geenral. Romance films tend to have more positive sentiments while horror movies tend to have more negative ones. Both genre uses common verbs and adjectives in the scripts. There is no noticable relationship between polarity scores and ratings from audience.

Full Dataset

The full dataset used in this project can be found as 'full_dataset.csv'. It consists of the data gathered from TMDB and Rotten Tomatoes, as well as the sentiment analysis of the script.

5.2 Further studies

The project generated some unexpected results, which provides several possible areas for future research.

Questions raised from the project:

- Is there a difference in the number of romance versus horror films produced each year?
- Why do romance films generally have a higher budget and revenue?
- Since romance and horror scripts share similar vocabulary, do they also share overarching themes?
- What would sway an individual to prefer one genre over the other?
- In this project, box-office vlues and audience ratings are held as the yardstick for commercial success. A future study taking into account movie reviews or awards and accolades won by films may shed light on audience ratings.

References

Data Sources

[1] The Internet Movie Script Database(IMSDb), 2023. Romance Movie Scripts. [Online] Available at: https://imsdb.com/genre/Romance [Accessed 1 December 2023].

[2] The Internet Movie Script Database(IMSDb), 2023. Horror Movie Scripts. [Online] Available at: https://imsdb.com/genre/Horror [Accessed 1 December 2023].

[3] TMDB, 2023. The Movie Database. [Online] Available at: https://www.themoviedb.org/ [Accessed 10 December 2023].

[4] Rotten Tomatoes, 2024. Rotten Tomatoes. [Online] Available at: https://www.rottentomatoes.com/ [Accessed 27 December 2023].

Inspiration

[1] Gross, J. A., Roberson, W. C. & Foley-Cox, J. B., 2021. CS 230: Film Success Prediction Using NLP Techniques. [Online] Available at: https://cs230.stanford.edu/projects_winter_2021/reports/70992925.pdf [Accessed 12 December 2023].

[2] Maz, Cassie., 2019. Animated-Movie-Gendered-Dialogue. [Online] Available at: https://github.com/Data-Science-for-Linguists-2019/Animated-Movie-Gendered-Dialogue/tree/master [Accessed 12 December 2023].

[3] Agarwal, A., 2020. Movie Genre Classifier using Natural Language Processing. [Online] Available at: https://medium.com/@aagarwal691/movie-genre-classifier-using-natural-language-processing-4d5e73da7b78 [Accessed 12 December 2023].

[4] Pan, K. (., 2018. Movie Recommender System Based On Natural Language Processing. [Online] Available at: https://sites.northwestern.edu/msia/2018/03/16/movie-recommender-system-based-on-natural-language-processing/ [Accessed 12 December 2023].

Data gathering, processing and visualising

[1] Bird, S., Klein, E. & Loper, E., 2019. 5. Categorizing and Tagging Words. [Online] Available at: https://www.nltk.org/book/ch05.html [Accessed 12 December 2023].

[2] Waskom, M., 2023. Scatterplot with continuous hues and sizes. [Online] Available at: https://seaborn.pydata.org/examples/scatterplot_sizes.html [Accessed 27 December 2023].

[3] Waskom, M., 2023. Overview of seaborn plotting functions. [Online] Available at: https://seaborn.pydata.org/tutorial/function_overview.html [Accessed 5 January 2024].

[4] Steinman, A., 2020. Navigating my first API: the TMDb Database. [Online] Available at: https://adinasteinman.medium.com/navigating-my-first-api-the-tmdb-database-d8d2975b0df4 [Accessed 30 December 2023].