student

November 6, 2024

0.1 Final Project Submission

Please fill out: * Student name: Robert Sheynin * Student pace: self paced * Scheduled project review date/time: * Instructor name: * Blog post URL:

1 Introduction

This project is an analysis of several datasets that contain information about movies. The goal of this project is to provide insights and business recommendations about what types of movies are currently doing the best at the box office. The analysis will focus on the following: - genres - reviews (rotten tomatoes and imdb) - ROI and profit

We will also focus on investigating any relationships between the above factors to answer: - What genres are currently the most popular? - What genres are the most profitable? - What genres have the best reviews? - What is the relationship between reviews and profit?

TODO - maybe include actors, directors, and production companies in the analysis

1.1 Import the necessary libraries

[1]:
```
# setup
import pandas as pd
import sqlite3
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import plotly.graph_objs as go
import plotly.express as px
import requests
import statsmodels.api as sm
from sklearn.preprocessing import KBinsDiscretizer, LabelEncoder
from bs4 import BeautifulSoup



# set pandas options
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', 100)
```

```
pd.set_option('display.precision', 2)
pd.set_option('display.large_repr', 'truncate')
pd.set_option('display.expand_frame_repr', False)
pd.set_option('display.memory_usage', 'deep')

# api key for tmdb
api_key = 'afb631d3b4cac582d777c74aeab9c37e'
```

```
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_67205/4230152606.py:2
: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of
pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better
interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

1.2 Data Cleaning and Aggregation

- Inspect the various data sources and determine what data is available and how it can be combined to create a single dataset

1.2.1 Process The Movie Database data

- load the data
- rename columns in preparation for merging
- drop extraneous columns

[2]:
```
# read the other files
tmdb_df = pd.read_csv('./data/tmdb.movies.csv')

# rename vote_average to movie_score_tmdb
tmdb_df.rename(columns={'vote_average': 'movie_score_tmdb'}, inplace=True)

# drop columns
tmdb_df.drop(columns=['Unnamed: 0', 'genre_ids', 'original_language',␣
↪'release_date', 'vote_count', 'popularity', 'id'], inplace=True)
```

1.2.2 Process Movie Gross Data

- Clean up and process the data from Box Office Mojo
- Aggregate the data with The Numbers
- Feature engineer the data to create a new column for ROI (Return on Investment)

[3]:
```
# load the revenue data
gross_df = pd.read_csv('./data/bom.movie_gross.csv')
```

```python
budgets_df = pd.read_csv('./data/tn.movie_budgets.csv')

# rename columns to have a common column name for merging
budgets_df.rename(columns={'movie': 'title', 'domestic_gross':␣
↪'domestic_gross_tn'}, inplace=True)
gross_df.rename(columns={ 'domestic_gross': 'domestic_gross_bom'}, inplace=True)

# merge the data frames on the title column
money_df = pd.merge(budgets_df, gross_df, on='title', how='inner')
money_df['domestic_gross_bom'] = money_df['domestic_gross_bom'].astype(float)
money_df['worldwide_gross'] = money_df['worldwide_gross'].str.replace(',', '').
↪str.replace('$', '').astype(float)
money_df['domestic_gross_tn'] = money_df['domestic_gross_tn'].str.replace(',',␣
↪'').str.replace('$', '').astype(float)
money_df['production_budget'] = money_df['production_budget'].str.replace(',',␣
↪'').str.replace('$', '').astype(float)

# merge the tmdb data
money_df = pd.merge(money_df, tmdb_df, on='title', how='inner')

# remove duplicates
money_df.drop_duplicates(subset='title', inplace=True)

# normalize the release date
money_df['release_date'] = pd.to_datetime(money_df['release_date'], format='%b␣
↪%d, %Y', errors='coerce')


# extract release month and year
money_df['release_month'] = money_df['release_date'].dt.strftime('%b')
money_df['release_year'] = money_df['release_date'].dt.year


# drop columns
# money_df.drop(columns=['domestic_gross_bom', 'domestic_gross_tn',␣
↪'foreign_gross', 'id'], inplace=True)

# add columns for profit and roi
money_df['profit'] = money_df['worldwide_gross'] - money_df['production_budget']
money_df['roi'] = (money_df['worldwide_gross'] - money_df['production_budget'])␣
↪/ money_df['production_budget']


money_df.head()
```

[3]:
```
   id release_date                                        title  
production_budget  domestic_gross_tn  worldwide_gross studio  domestic_gross_bom  
foreign_gross  year                                     original_title  
movie_score_tmdb release_month  release_year    profit   roi  
0   2   2011-05-20  Pirates of the Caribbean: On Stranger Tides  
4.11e+08            2.41e+08          1.05e+09      BV            2.41e+08  
804600000  2011  Pirates of the Caribbean: On Stranger Tides               6.4  
May           2011  6.35e+08  1.55  
1   4   2015-05-01                      Avengers: Age of Ultron  
3.31e+08            4.59e+08          1.40e+09      BV            4.59e+08  
946400000  2015                      Avengers: Age of Ultron               7.3  
May           2015  1.07e+09  3.24  
2   7   2018-04-27                       Avengers: Infinity War  
3.00e+08            6.79e+08          2.05e+09      BV            6.79e+08  
1,369.5  2018                       Avengers: Infinity War               8.3  
Apr           2018  1.75e+09  5.83  
3   9   2017-11-17                               Justice League  
3.00e+08            2.29e+08          6.56e+08      WB            2.29e+08  
428900000  2017                               Justice League               6.2  
Nov           2017  3.56e+08  1.19  
5  10   2015-11-06                                      Spectre  
3.00e+08            2.00e+08          8.80e+08   Sony             2.00e+08  
680600000  2015                                      Spectre               6.4  
Nov           2015  5.80e+08  1.93
```

1.2.3 Process Movie Reviews

Clean up and aggregate the movie rotten tomato review data and rotten tomato review metadata. - load the movie data - impute missing ratings using 'fresh' and 'rotten' ratings as 8 and 2 respectively - remove reviews with no rating and no fresh or rotten rating - normalize the ratings to a 0-10 scale - remove extraneous columns ('critic', 'top_critic', 'publisher', 'date', 'box_office', 'genre', 'director', 'studio') - remove all attributes except 'id' and 'rating' - note that 'review' which contains the review text is not included in the final data but could have been used for sentiment analysis to infer a missing rating as positive or negative - create a function to query tmdb for movie titles by director and year - combine the review data with the movie metadata

[4]:
```python
# load data
reviews_df = pd.read_csv('./data/rt.reviews.tsv', delimiter='\t',␣
 ↪encoding='latin1')
movie_info_df = pd.read_csv('./data/rt.movie_info.tsv', delimiter='\t',␣
 ↪encoding='latin1')

# drop rows that do not have both a rating and a 'fresh' or 'rotten' value
reviews_df = reviews_df[reviews_df['fresh'].isin(['fresh', 'rotten']) |␣
 ↪reviews_df['rating'].notnull()]

# normalize ratings
```

```python
def convert_rating(rating):
    letter_grade_map = {
        'A+': 10, 'A': 9.5, 'A-': 9,
        'B+': 8.5, 'B': 8, 'B-': 7.5,
        'C+': 7, 'C': 6.5, 'C-': 6,
        'D+': 5.5, 'D': 5, 'D-': 4.5,
        'F': 3
    }

    def parse_fraction(fraction_str):
        try:
            if ' ' in fraction_str:
                integer_part, fraction_part = fraction_str.split(' ')
                num, denom = fraction_part.split('/')
                return float(integer_part) + (float(num) / float(denom))
            elif '/' in fraction_str:
                num, denom = fraction_str.split('/')
                return float(num) / float(denom)
            else:
                return float(fraction_str)
        except ValueError:
            print(f'Warning: Could not convert fraction {fraction_str}')
            return None

    try:
        if isinstance(rating, str):
            if '/' in rating or ' ' in rating:
                return parse_fraction(rating) * 10
            elif rating in letter_grade_map:
                return letter_grade_map[rating]
            else:
                return None
        else:
            return None

    except ValueError:
        print(f'Warning: Could not convert rating {rating}')
        return None


# apply the conversion
reviews_df['rating'] = reviews_df['rating'].apply(convert_rating)

# convert rating to float
reviews_df['rating'] = reviews_df['rating'].astype(float)

# map the rotten and fresh ratings to integers to impute the missing ratings
```

```python
fresh_rotten_mapping = {
    'rotten': 5,
    'fresh': 8
}

# infer ratings based on 'fresh' and 'rotten' values
reviews_df['inferred_rating'] = reviews_df['fresh'].map(fresh_rotten_mapping)

# if 'review_score' is missing, use the inferred rating
reviews_df['rating'] = reviews_df['rating'].fillna(reviews_df['inferred_rating'])

# drop the 'inferred_rating' column as it was just used for imputation
reviews_df.drop(columns=['inferred_rating'], inplace=True)

# rename the 'rating' column in reviews_df to 'review_score'
reviews_df.rename(columns={'rating': 'review_score'}, inplace=True)

# merge the DataFrames on the 'id' column
rotten_tomato_df = pd.merge(movie_info_df, reviews_df, on='id', how='inner')

# normalize the release date
rotten_tomato_df['theater_date'] = pd.to_datetime(rotten_tomato_df['theater_date'], format='%b %d, %Y', errors='coerce')

# extract release month and year
rotten_tomato_df['month'] = rotten_tomato_df['theater_date'].dt.strftime('%b')
rotten_tomato_df['year'] = rotten_tomato_df['theater_date'].dt.year

# process the genre column
rotten_tomato_df['genre'] = rotten_tomato_df['genre'].str.split('|')
#rotten_tomato_df['genre'] = rotten_tomato_df['genre'].apply(lambda x: [genre.strip().lower() for genre in x] if x is not None else x)

# function to query the TMDb API for a movie's title by director and release date
def search_movie_by_director_release(director_name, release_date):
    # Step 1: Search for the director to get their TMDb ID
    director_search_url = f"https://api.themoviedb.org/3/search/person?api_key={api_key}&query={director_name}"
    director_response = requests.get(director_search_url).json()

    if director_response['results']:
        director_id = director_response['results'][0]['id']

        # Step 2: Use the director ID to search for movies
```

```python
        movie_search_url = f"https://api.themoviedb.org/3/discover/movie?
↪api_key={api_key}&with_people={director_id}&primary_release_year={release_date[:
↪4]}"
        movie_response = requests.get(movie_search_url).json()

        if movie_response['results']:
            return movie_response['results'][0]['title']  # Return the first␣
↪matched title
        else:
            return "No Title Found"
    else:
        return "Director Not Found"

# remove duplicates based on 'id'
rotten_tomato_df.drop_duplicates(subset='id', inplace=True)

# create a new column 'title' by making an API call for each row
rotten_tomato_df['title'] = rotten_tomato_df.apply(
    lambda row: search_movie_by_director_release(row['director'],␣
↪str(row['theater_date'])), axis=1
)

# drop columns
rotten_tomato_df.drop(columns=['dvd_date', 'date', 'box_office', 'synopsis',␣
↪'critic', 'publisher', 'top_critic', 'review'], inplace=True)

rotten_tomato_df.head()
```

[4]:
```
     id rating                                   genre          director  \
writer theater_date currency      runtime                 studio  review_score  \
fresh month    year              title
0     3      R  [Drama, Science Fiction and Fantasy]  David Cronenberg     David
Cronenberg|Don DeLillo   2012-08-17         $  108 minutes      Entertainment One
6.0   fresh   Aug  2012.0         Cosmopolis
163   5      R  [Drama, Musical and Performing Arts]    Allison Anders
Allison Anders   1996-09-13      NaN  116 minutes                        NaN
8.0   fresh   Sep  1996.0  Grace of My Heart
186   6      R         [Drama, Mystery and Suspense]    Barry Levinson  Paul
Attanasio|Michael Crichton   1994-12-09      NaN  128 minutes
NaN            5.0  rotten   Dec  1994.0          Quiz Show
243   8     PG              [Drama, Kids and Family]       Jay Russell
Gail Gilchriest   2000-03-03      NaN   95 minutes  Warner Bros. Pictures
8.0   fresh   Mar  2000.0        My Dog Skip
318  10  PG-13                               [Comedy]       Jake Kasdan
Mike White   2002-01-11        $   82 minutes     Paramount Pictures
6.0   fresh   Jan  2002.0      Orange County
```

[5]:
```
## note that we have some missing values.

# print the number of rows that contain 'No Title Found' in the 'title' column␣
↪as a ratio of the total number of rows
print(rotten_tomato_df[rotten_tomato_df['title'] == 'No Title Found'].shape[0] /
↪ rotten_tomato_df.shape[0])
# print the total number of rows
print(rotten_tomato_df.shape[0])
```

```
0.1656387665198238
1135
```

1.2.4 Process IMDB Data

- load the IMDB data
- query all tables and load them into a data frame
- merge data

[6]:
```
# load sql data
conn = sqlite3.connect('./data/im.db')

# load all tables into a dataframe
tables = pd.read_sql_query("SELECT name FROM sqlite_master WHERE type='table';
↪", conn)
table_names = tables['name'].tolist()
im_df = {table: pd.read_sql_query(f"SELECT * FROM {table};", conn) for table in␣
↪table_names}

# close connection
conn.close()

# print the head of each table
for table, df in im_df.items():
    print(f'{table}:\n{df.head()}\n\n')
```

```
movie_basics:
    movie_id                    primary_title              original_title
start_year  runtime_minutes                genres
0  tt0063540                        Sunghursh                   Sunghursh
2013            175.0    Action,Crime,Drama
1  tt0066787  One Day Before the Rainy Season             Ashad Ka Ek Din
2019            114.0       Biography,Drama
2  tt0069049       The Other Side of the Wind  The Other Side of the Wind
2018            122.0                 Drama
3  tt0069204                  Sabse Bada Sukh             Sabse Bada Sukh
2018              NaN          Comedy,Drama
4  tt0100275         The Wandering Soap Opera       La Telenovela Errante
2017             80.0  Comedy,Drama,Fantasy
```

```
directors:
    movie_id  person_id
0  tt0285252  nm0899854
1  tt0462036  nm1940585
2  tt0835418  nm0151540
3  tt0835418  nm0151540
4  tt0878654  nm0089502


known_for:
   person_id   movie_id
0  nm0061671  tt0837562
1  nm0061671  tt2398241
2  nm0061671  tt0844471
3  nm0061671  tt0118553
4  nm0061865  tt0896534


movie_akas:
    movie_id  ordering                                    title region language
types   attributes  is_original_title
0  tt0369610       10                                        BG       bg
None         None                0.0
1  tt0369610       11                      Jurashikku warudo     JP     None
imdbDisplay         None                0.0
2  tt0369610       12  Jurassic World: O Mundo dos Dinossauros     BR     None
imdbDisplay         None                0.0
3  tt0369610       13                  O Mundo dos Dinossauros     BR     None
None  short title                0.0
4  tt0369610       14                         Jurassic World     FR     None
imdbDisplay         None                0.0


movie_ratings:
     movie_id  averagerating  numvotes
0  tt10356526            8.3        31
1  tt10384606            8.9       559
2   tt1042974            6.4        20
3   tt1043726            4.2     50352
4   tt1060240            6.5        21


persons:
   person_id       primary_name  birth_year  death_year
primary_profession
0  nm0061671  Mary Ellen Bauder         NaN         NaN
```

```
miscellaneous,production_manager,producer
1  nm0061865       Joseph Bauer         NaN         NaN
composer,music_department,sound_department
2  nm0062070         Bruce Baum         NaN         NaN
miscellaneous,actor,writer
3  nm0062195       Axel Baumann         NaN         NaN
camera_department,cinematographer,art_department
4  nm0062798        Pete Baxter         NaN         NaN
production_designer,art_department,set_decorator


principals:
    movie_id  ordering  person_id  category       job        characters
0  tt0111414         1  nm0246005     actor      None       ["The Man"]
1  tt0111414         2  nm0398271  director      None              None
2  tt0111414         3  nm3739909  producer  producer              None
3  tt0323808        10  nm0059247    editor      None              None
4  tt0323808         1  nm3579312   actress      None  ["Beth Boothby"]


writers:
    movie_id  person_id
0  tt0285252  nm0899854
1  tt0438973  nm0175726
2  tt0438973  nm1802864
3  tt0462036  nm1940585
4  tt0835418  nm0310087
```

[7]:

```python
# flatten the imdb dataframe
movie_df = im_df['movie_basics']

# merge movie_ratings
movie_df = pd.merge(movie_df, im_df['movie_ratings'], on='movie_id', how='left')

# mark missing values as unknown
movie_df['genres'] = movie_df['genres'].fillna('Unknown')

# split genres into a list
movie_df['genres'] = movie_df['genres'].str.split(',')

# normalize genres
movie_df['genres'] = movie_df['genres'].apply(lambda x: [genre.strip().lower()␣
↪for genre in x])
```

```python
# merge directors
df = pd.merge(df, im_df['directors'], on='movie_id', how='left')

# merge writers
# df = pd.merge(df, im_df['writers'], on='movie_id', how='left')

# # merge with principals
# df = pd.merge(df, im_df['principals'], on='movie_id', how='left')

# # merge with writers
# df = pd.merge(df, im_df['writers'], on='movie_id', how='left')

# # merge with persons
# df = pd.merge(df, im_df['persons'], on='person_id', how='left')

movie_df.head(10)
```

[7]:

```
    movie_id                      primary_title              original_title
start_year  runtime_minutes                           genres  averagerating
numvotes
0  tt0063540                          Sunghursh                   Sunghursh
2013            175.0          [action, crime, drama]            7.0      77.0
1  tt0066787  One Day Before the Rainy Season             Ashad Ka Ek Din
2019            114.0              [biography, drama]            7.2      43.0
2  tt0069049       The Other Side of the Wind  The Other Side of the Wind
2018            122.0                         [drama]            6.9    4517.0
3  tt0069204                    Sabse Bada Sukh             Sabse Bada Sukh
2018              NaN                 [comedy, drama]            6.1      13.0
4  tt0100275         The Wandering Soap Opera       La Telenovela Errante
2017             80.0        [comedy, drama, fantasy]            6.5     119.0
5  tt0111414                        A Thin Life                 A Thin Life
2018             75.0                        [comedy]            NaN       NaN
6  tt0112502                            Bigfoot                     Bigfoot
2017              NaN             [horror, thriller]            4.1      32.0
7  tt0137204                    Joe Finds Grace             Joe Finds Grace
2017             83.0  [adventure, animation, comedy]            8.1     263.0
8  tt0139613                         O Silêncio                  O Silêncio
2012              NaN           [documentary, history]            NaN       NaN
9  tt0144449            Nema aviona za Zagreb       Nema aviona za Zagreb
2012             82.0                     [biography]            NaN       NaN
```

1.2.5 Combine the data

- combine the data from the movie database, movie gross, movie reviews, and IMDB data into a single data frame
- save the data to derrived_data.csv

[8]:
```
movie_df.head()
```

[8]:
```
    movie_id                    primary_title              original_title  \
start_year  runtime_minutes                       genres  averagerating  numvotes
0  tt0063540                        Sunghursh                   Sunghursh
2013             175.0     [action, crime, drama]            7.0       77.0
1  tt0066787  One Day Before the Rainy Season             Ashad Ka Ek Din
2019             114.0         [biography, drama]            7.2       43.0
2  tt0069049       The Other Side of the Wind  The Other Side of the Wind
2018             122.0                    [drama]            6.9     4517.0
3  tt0069204                  Sabse Bada Sukh             Sabse Bada Sukh
2018               NaN            [comedy, drama]            6.1       13.0
4  tt0100275         The Wandering Soap Opera       La Telenovela Errante
2017              80.0   [comedy, drama, fantasy]            6.5      119.0
```

[9]:
```
rotten_tomato_df.head()
# print the number of rows that contain 'No Title Found' in the 'title' column␣
 ↪as a ratio of the total number of rows
print(rotten_tomato_df[rotten_tomato_df['title'] == 'No Title Found'].shape[0] /
 ↪ rotten_tomato_df.shape[0])
```

```
0.1656387665198238
```

[10]:
```
money_df.head()
```

[10]:
```
   id release_date                                     title
production_budget  domestic_gross_tn  worldwide_gross studio  domestic_gross_bom
foreign_gross  year                            original_title
movie_score_tmdb release_month  release_year    profit   roi
0   2   2011-05-20  Pirates of the Caribbean: On Stranger Tides
4.11e+08           2.41e+08          1.05e+09     BV           2.41e+08
804600000  2011  Pirates of the Caribbean: On Stranger Tides              6.4
May           2011  6.35e+08  1.55
1   4   2015-05-01                          Avengers: Age of Ultron
3.31e+08           4.59e+08          1.40e+09     BV           4.59e+08
946400000  2015                          Avengers: Age of Ultron              7.3
May           2015  1.07e+09  3.24
2   7   2018-04-27                           Avengers: Infinity War
3.00e+08           6.79e+08          2.05e+09     BV           6.79e+08
1,369.5  2018                          Avengers: Infinity War              8.3
Apr           2018  1.75e+09  5.83
3   9   2017-11-17                                   Justice League
3.00e+08           2.29e+08          6.56e+08     WB           2.29e+08
428900000  2017                                  Justice League              6.2
Nov           2017  3.56e+08  1.19
5  10   2015-11-06                                          Spectre
3.00e+08           2.00e+08          8.80e+08    Sony          2.00e+08
680600000  2015                                         Spectre              6.4
```

```
Nov         2015  5.80e+08  1.93
```

[11]:

```
# merge movie_df with money_df on 'title' and either 'primary_title' or␣
↪'original_title'
merged_df_primary = pd.merge(money_df, movie_df, left_on='title',␣
↪right_on='primary_title', how='left')

# for rows where primary_title didn't match, try merging with original_title
no_match_primary = merged_df_primary[merged_df_primary['primary_title'].isna()]
matched_original = pd.merge(no_match_primary.drop(columns=movie_df.columns.
↪difference(['movie_id', 'original_title'])),
                            movie_df, left_on='title',␣
↪right_on='original_title', how='left')

# combine the results
final_movie_money_df = pd.
↪concat([merged_df_primary[~merged_df_primary['primary_title'].isna()],␣
↪matched_original])

# merge the combined DataFrame with rotten_tomato_df on 'title'
final_combined_df = pd.merge(final_movie_money_df, rotten_tomato_df,␣
↪on='title', how='left')

# rename the 'averagerating' column to 'rating_imdb'
final_combined_df.rename(columns={'averagerating': 'rating_imdb'}, inplace=True)

# rename the 'review_score' column to 'rating_rotten_tomato'
final_combined_df.rename(columns={'review_score': 'critic_score'}, inplace=True)

# rename the 'movie_score_tmdb' column to 'rating_tmdb'
final_combined_df.rename(columns={'movie_score_tmdb': 'rating_tmdb'},␣
↪inplace=True)

# create a new column 'rating' that is the average of the 'rating_imdb',␣
↪'rating_rotten_tomato', and 'rating_tmdb' columns
final_combined_df['rating'] = final_combined_df[['rating_imdb', 'rating_tmdb',␣
↪'critic_score']].mean(axis=1)

# automatically combine columns with suffixes '_x' and '_y'
for col in final_combined_df.columns:
    if '_x' in col:
        base_col_name = col.rstrip('_x')
        if base_col_name in final_combined_df.columns:
            final_combined_df[base_col_name] = final_combined_df[base_col_name].
↪combine_first(final_combined_df[col])
        else:
```

```
            final_combined_df[base_col_name] = final_combined_df[col]
        final_combined_df.drop(columns=[col], inplace=True)
    elif '_y' in col:
        base_col_name = col.rstrip('_y')
        if base_col_name in final_combined_df.columns:
            final_combined_df[base_col_name] = final_combined_df[base_col_name].
 ↪combine_first(final_combined_df[col])
        else:
            final_combined_df[base_col_name] = final_combined_df[col]
        final_combined_df.drop(columns=[col], inplace=True)

# convert list-type columns to a hashable type (like a string or tuple)
for col in final_combined_df.columns:
    if final_combined_df[col].apply(lambda x: isinstance(x, list)).any():
        final_combined_df[col] = final_combined_df[col].apply(lambda x: ', '.
 ↪join(x) if isinstance(x, list) else x)


# drop duplicates
final_combined_df.drop_duplicates(inplace=True)

# drop empty/unused columns
final_combined_df.drop(columns=['theater_date', 'currency', 'fresh',
                                'genre', 'runtime', 'director',
                                'writer', 'runtime', 'month'
                                ], inplace=True)

# save the DataFrame to a CSV file
final_combined_df.to_csv('./data/derrived_data.csv', index=False)
```

```
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_67205/2232669968.py:3
9: FutureWarning: The behavior of array concatenation with empty entries is
deprecated. In a future version, this will no longer exclude empty items when
determining the result dtype. To retain the old behavior, exclude the empty
entries before the concat operation.
  final_combined_df[base_col_name] =
final_combined_df[base_col_name].combine_first(final_combined_df[col])
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_67205/2232669968.py:3
9: FutureWarning: The behavior of array concatenation with empty entries is
deprecated. In a future version, this will no longer exclude empty items when
determining the result dtype. To retain the old behavior, exclude the empty
entries before the concat operation.
  final_combined_df[base_col_name] =
final_combined_df[base_col_name].combine_first(final_combined_df[col])
```

[12]:
```
df = pd.read_csv('./data/derrived_data.csv')
```

```
df.drop(columns=[ 'original_title', 'primary_title', 'domestic_gross_tn',␣
↪'worldwide_gross',
                  'domestic_gross_bom', 'foreign_gross', 'numvotes', 'id',
                  'rating_tmdb', 'rating_imdb', 'critic_score', 'year',
                  'profit', 'production_budget', 'release_date', 'movie_id'
                  ], inplace=True)

# explode the genres column
# df['genres'] = df['genres'].str.split(', ')
# df = df.explode('genres')

# Encode 'release_month' to numerical values for the month
month_mapping = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6,
                  'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}
df['release_month_encoded'] = df['release_month'].map(month_mapping)


df['studio_encoded'] = LabelEncoder().fit_transform(df['studio'])

# adjust the rating column to be between 1 and 10
#df['rating_decile'] = pd.qcut(df['rating'], q=10, labels=False) + 1

# drop rows with missing genres
df = df[df['genres'].notna()]

df.head()
```

[12]:

```
                                         title release_month   roi  \
runtime_minutes                          genres  rating studio
release_month_encoded  studio_encoded
0  Pirates of the Caribbean: On Stranger Tides           May  1.55
136.0   action, adventure, fantasy    6.50     BV                     5
14
1                      Avengers: Age of Ultron           May  3.24
141.0    action, adventure, sci-fi    7.30     BV                     5
14
2                       Avengers: Infinity War           Apr  5.83
149.0    action, adventure, sci-fi    8.40     BV                     4
14
3                               Justice League           Nov  1.19
120.0   action, adventure, fantasy    6.35     WB                    11
86
4                                      Spectre           Nov  1.93
148.0  action, adventure, thriller    6.60   Sony                    11
74
```

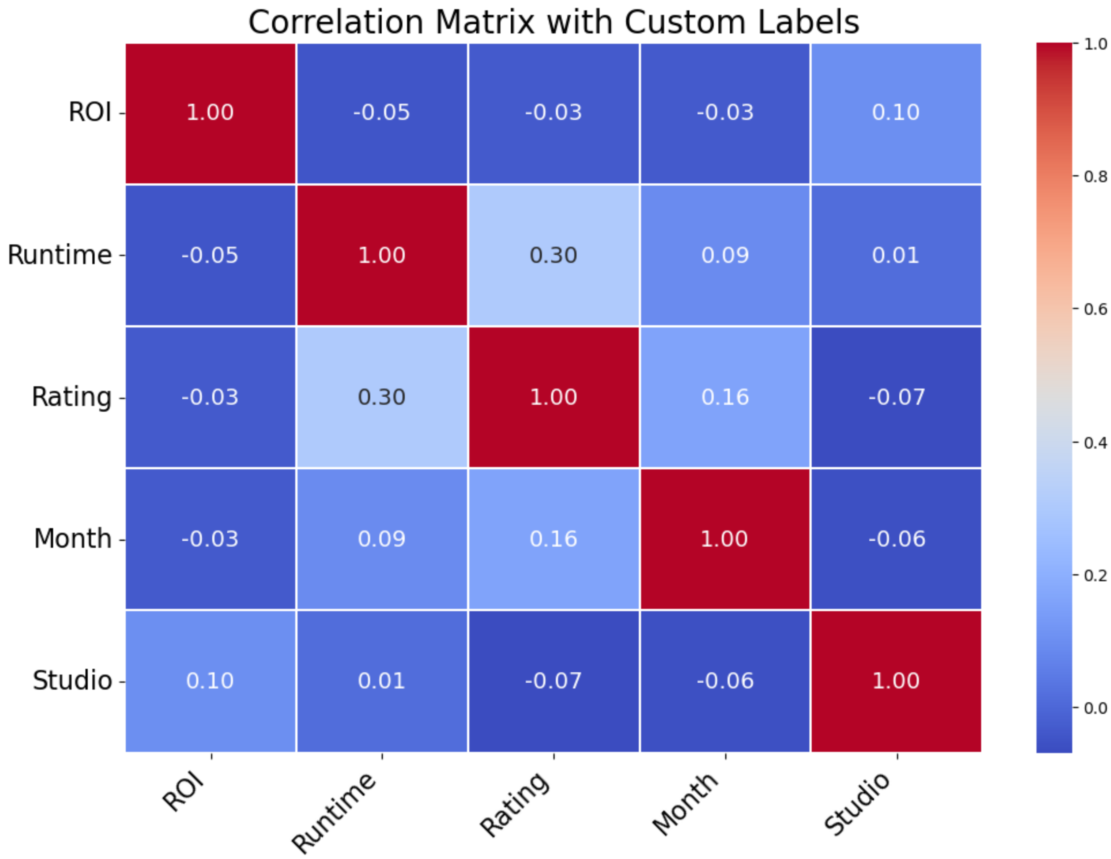
1.3 Exploratory Data Analysis

1.4 Correlation Matrix

- create a correlation matrix to determine the relationships between the various numerical attributes

[13]:
```
# Step 1: Select relevant numeric columns
numeric_cols = df.select_dtypes(include=[np.number])

# Step 2: Compute the correlation matrix
correlation_matrix = numeric_cols.corr()

# Define a dictionary to map original column names to custom labels
custom_labels = {
    'roi': 'ROI',
    'runtime_minutes': 'Runtime',
    'rating': 'Rating',
    'release_month_encoded': 'Month',  # Replace with 'MONTH' or any desired␣
 ↪label
    'studio_encoded': 'Studio'
}

# Rename the columns and index of the correlation matrix
correlation_matrix = correlation_matrix.rename(columns=custom_labels,␣
 ↪index=custom_labels)

# Plot the heatmap with customized labels and increased font sizes
plt.figure(figsize=(12, 8))
heatmap = sns.heatmap(
    correlation_matrix,
    annot=True,
    fmt=".2f",
    cmap='coolwarm',
    linewidths=0.2,
    annot_kws={"size": 14}  # Increase font size of annotations inside the␣
 ↪heatmap
)

heatmap.set_xticklabels(heatmap.get_xticklabels(), fontsize=16, rotation=45,␣
 ↪ha='right')  # Rotate x-axis labels
heatmap.set_yticklabels(heatmap.get_yticklabels(), fontsize=16, rotation=0)  #␣
 ↪Make y-axis labels horizontal
plt.title('Correlation Matrix with Custom Labels', fontsize=20)  # Increase␣
 ↪title font size


# Save and show the plot
```

```
plt.savefig('./images/correlation_matrix_custom_labels_large_font.png',␣
↪bbox_inches='tight')
plt.show()
```



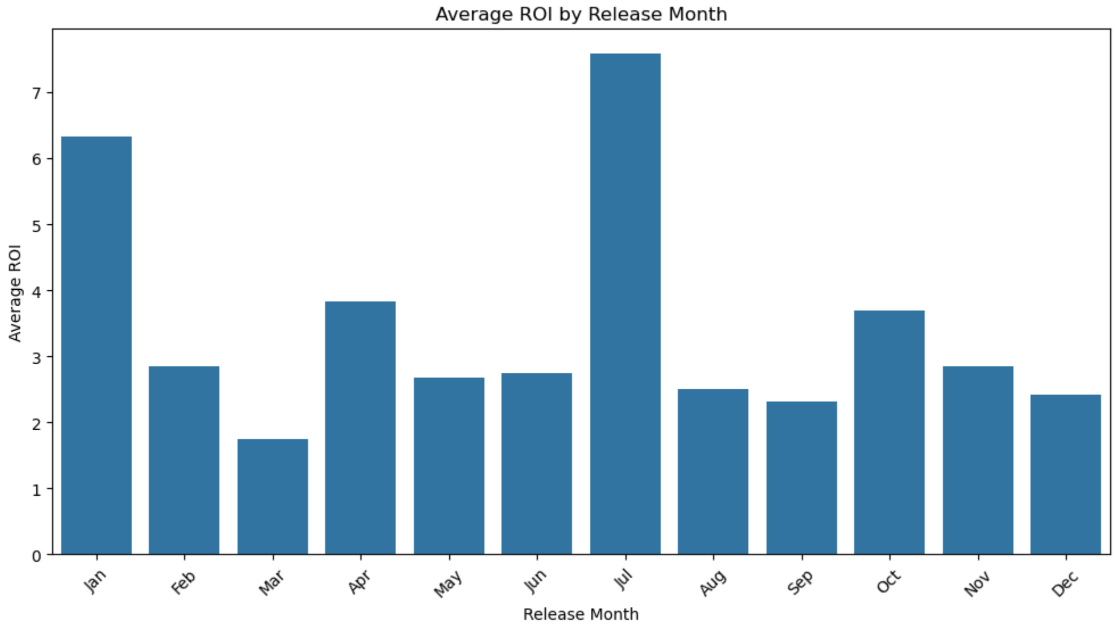
Note

- Genres are difficult to encode given how many unique values exist in the attribute. One hot encoding would create many columns.

1.4.1 ROI by Month

[14]:
```
# group by release month and calculate the mean ROI for each month
monthly_roi = df.groupby('release_month')['roi'].mean().reset_index()

# sort the DataFrame by the month order
monthly_roi['release_month'] = pd.Categorical(monthly_roi['release_month'],␣
↪categories=[
    'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct',␣
↪'Nov', 'Dec'], ordered=True)
monthly_roi = monthly_roi.sort_values('release_month')
```

```
# create a bar chart for ROI
plt.figure(figsize=(12, 6))
sns.barplot(x='release_month', y='roi', data=monthly_roi)
plt.title('Average ROI by Release Month')
plt.xlabel('Release Month')
plt.ylabel('Average ROI')
plt.xticks(rotation=45)
# save the image
plt.savefig('./images/average_roi_by_release_month.png', bbox_inches='tight')
plt.show()
```



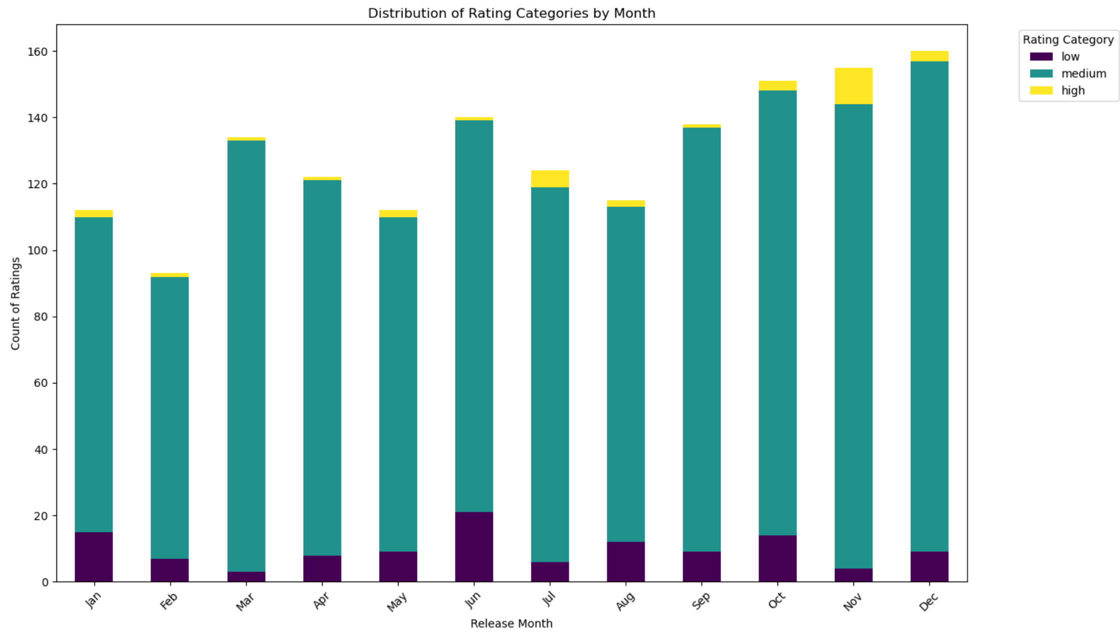
Findings

- The highest ROI months are January and July

[15]:
```
bins = [0, 5, 8, 10]
labels = ['low', 'medium', 'high']
df['rating_category'] = pd.cut(df['rating'], bins=bins, labels=labels)

# Group by release month and rating category, and count the occurrences
rating_by_month = df.groupby(['release_month', 'rating_category']).size().
↪unstack().fillna(0)

# Sort the DataFrame by the month order
rating_by_month.index = pd.Categorical(rating_by_month.index, categories=[
```

```
    'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct',␣
 ↪'Nov', 'Dec'], ordered=True)
rating_by_month = rating_by_month.sort_index()

# Check the result
rating_by_month.head()

# Create a stacked bar chart
rating_by_month.plot(kind='bar', stacked=True, figsize=(14, 8),␣
 ↪colormap='viridis')

plt.title('Distribution of Rating Categories by Month')
plt.xlabel('Release Month')
plt.ylabel('Count of Ratings')
plt.xticks(rotation=45)
plt.legend(title='Rating Category', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

# Save the figure
plt.savefig('./images/rating_distribution_by_month.png', bbox_inches='tight',␣
 ↪dpi=300)

# Show the plot
plt.show()
```

```
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_67205/1537225582.py:6
: FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  rating_by_month = df.groupby(['release_month',
'rating_category']).size().unstack().fillna(0)
```



Findings

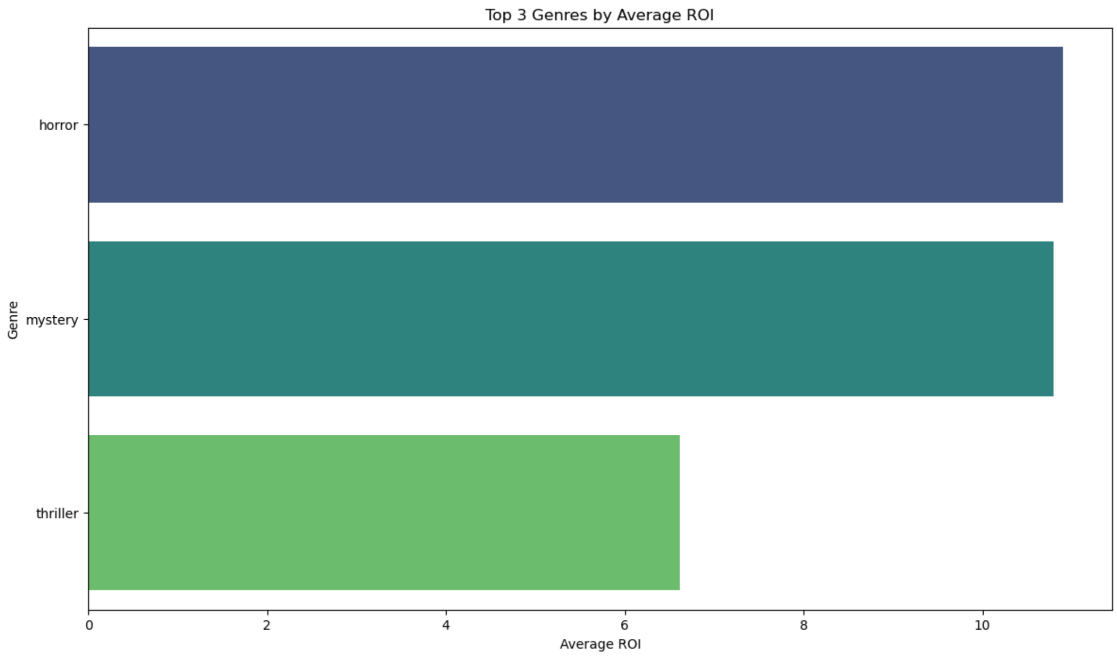
- The most low reviews occur in movies released in January, July and October
- The highest reviews occur in movies released in November

1.4.2 Comparison of ROI by Genre

[16]:
```python
# explode the genres so each genre gets its own row
df['genres'] = df['genres'].str.split(', ')
movie_df_exploded = df.explode('genres')

# calculate the average ROI for each genre
roi_by_genre = movie_df_exploded.groupby('genres')['roi'].mean().reset_index()

# sort the data by ROI
roi_by_genre_sorted = roi_by_genre.sort_values(by='roi', ascending=False)

# select the top 5 and bottom 3 genres
top_3_genres = roi_by_genre_sorted.head(3)

# create the bar chart using seaborn
plt.figure(figsize=(14, 8))
sns.barplot(x='roi', y='genres', data=top_3_genres, palette='viridis')
plt.title('Top 3 Genres by Average ROI')
plt.xlabel('Average ROI')
plt.ylabel('Genre')
```

```
plt.savefig('./images/top_3_genres_by_roi.png', bbox_inches='tight')
plt.show()

plt.show()
```

```
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_67205/3879649618.py:1
6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='roi', y='genres', data=top_3_genres, palette='viridis')
```

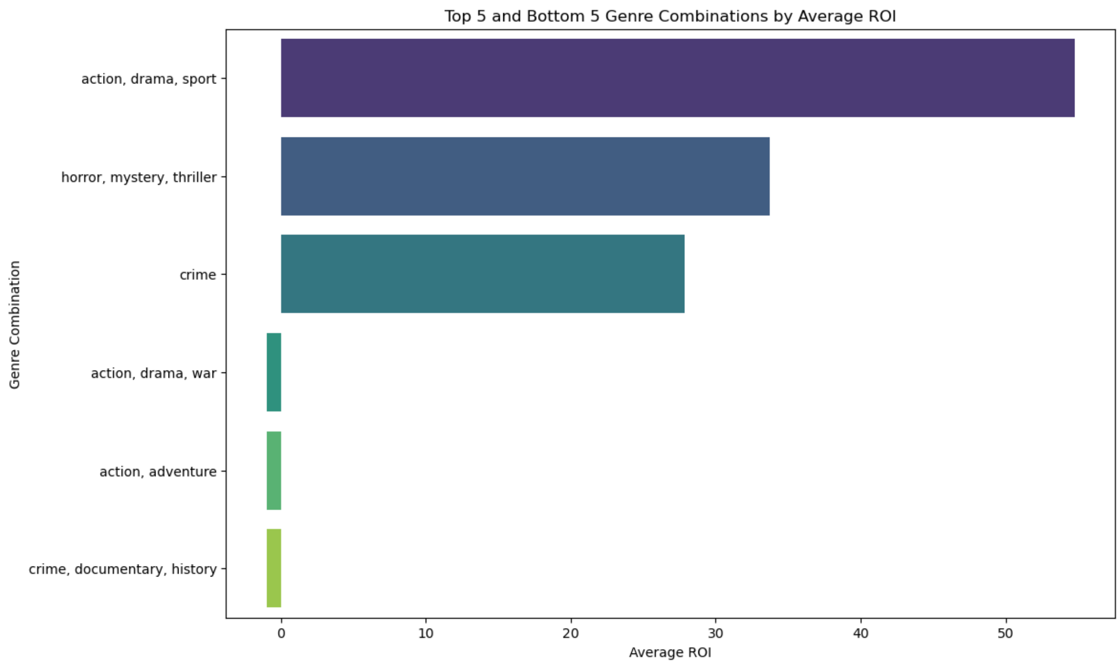


Findings

- The most profitable single genres are:
  - Horror
  - Mystery
  - Thriller
- The least profitable single genres are:
  - Western
  - War
  - News

[17]:
```python
# convert the genres column from lists to strings
df['genres'] = df['genres'].apply(lambda x: ', '.join(x) if isinstance(x, list)␣
 ↪else x)

# group by Genre Combinations and Calculate Average ROI
roi_by_genre_combination = df.groupby('genres')['roi'].mean().reset_index()

# sort the Combinations by ROI
roi_by_genre_combination = roi_by_genre_combination.sort_values(by='roi',␣
 ↪ascending=False)

# select the Top 3 and Bottom 3 Genre Combinations
top_3_genres = roi_by_genre_combination.head(3)

bottom_3_genres = roi_by_genre_combination.tail(3)

# combine the top and bottom 3
selected_genres = pd.concat([top_3_genres, bottom_3_genres])

# visualize the results
plt.figure(figsize=(12, 8))
sns.barplot(x='roi', y='genres', data=selected_genres, palette='viridis')
plt.title('Top 5 and Bottom 5 Genre Combinations by Average ROI')
plt.xlabel('Average ROI')
plt.ylabel('Genre Combination')
plt.show()
```

```
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_67205/1019516597.py:2
0: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='roi', y='genres', data=selected_genres, palette='viridis')
```



Findings

- The genres with the highest ROI are:
  - Action, Drama, Sport
  - Horror, Mystery, Thriller
  - Crime
- The genres with the lowest ROI are:
  - Action, Drama, War
  - Action, Adventure
  - Crime, Documentary, History

Note that these findings are also supported by the previous analysis of Individual Genres

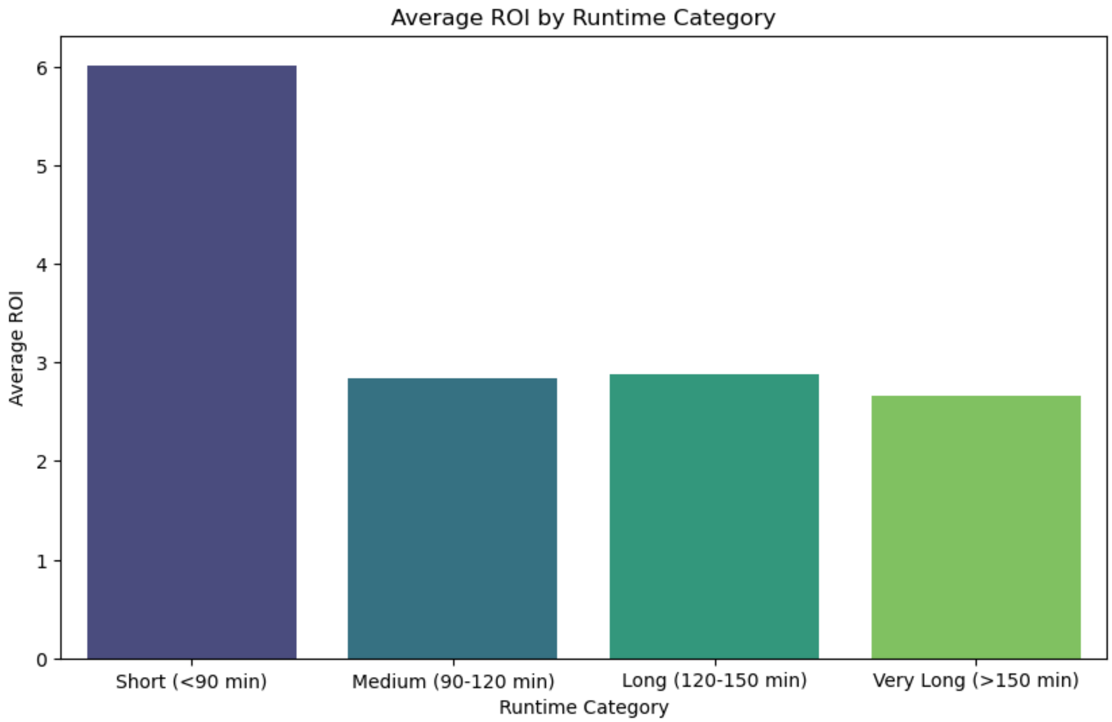
1.4.3 Analysis of Movie Runtime

- Consider the relationship between runtime and ROI

[18]:
```
# bin runtimes into categories
bins = [0, 90, 120, 150, float('inf')]
labels = ['Short (<90 min)', 'Medium (90-120 min)', 'Long (120-150 min)', 'Very␣
↪Long (>150 min)']
df['runtime_category'] = pd.cut(df['runtime_minutes'], bins=bins,␣
↪labels=labels, right=False)

# calculate the ROI for each runtime category
```

```
roi_by_runtime_category = df.groupby('runtime_category')['roi'].mean().
↪reset_index()

# plot the results
plt.figure(figsize=(10, 6))
sns.barplot(x='runtime_category', y='roi', data=roi_by_runtime_category,␣
↪palette='viridis')
plt.title('Average ROI by Runtime Category')
plt.xlabel('Runtime Category')
plt.ylabel('Average ROI')


plt.savefig('./images/avg_roi_by_runtime.png', bbox_inches='tight')

plt.show()
```

```
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_67205/4034857503.py:7
: FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  roi_by_runtime_category =
df.groupby('runtime_category')['roi'].mean().reset_index()
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_67205/4034857503.py:1
1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='runtime_category', y='roi', data=roi_by_runtime_category,
palette='viridis')
```



Findings

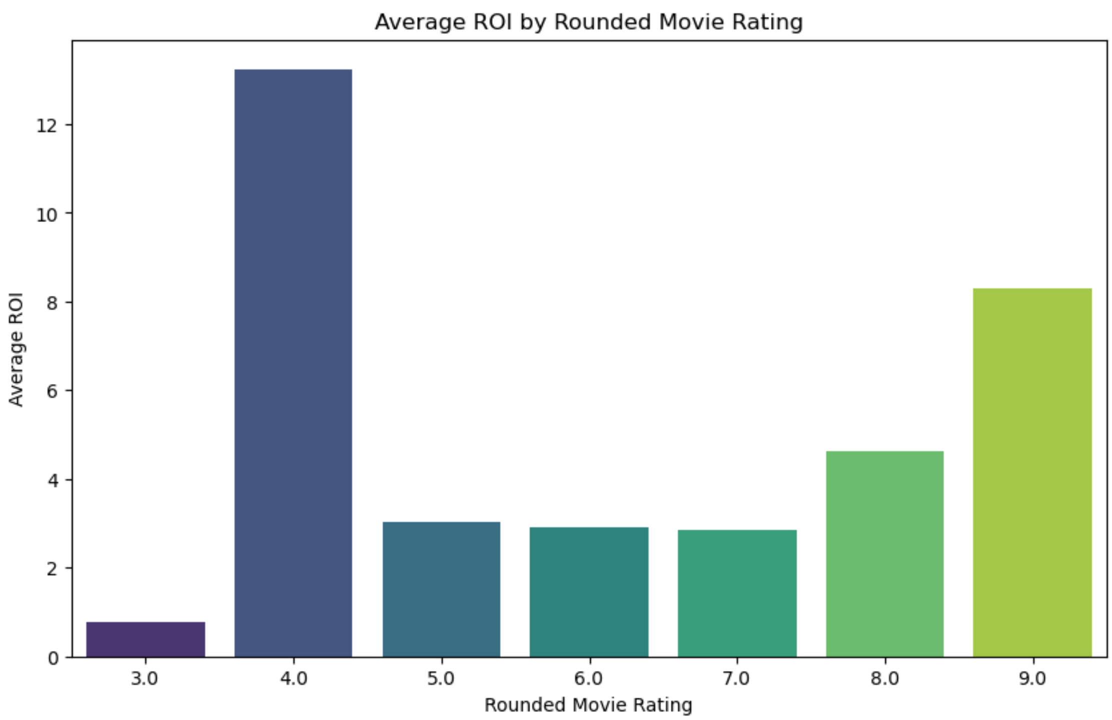
- Shorter runtime movies tend to have higher ROI on average

1.4.4 Analysis of Reviews

- Consider the relationship between reviews and ROI

[19]:
```
# Step 1: Round the Ratings to the Nearest Integer
df['rounded_rating'] = df['rating'].round()

# Step 2: Calculate the Average ROI for Each Rounded Rating
roi_by_rounded_rating = df.groupby('rounded_rating')['roi'].mean().reset_index()

# Step 3: Create the Bar Chart
plt.figure(figsize=(10, 6))
sns.barplot(x='rounded_rating', y='roi', data=roi_by_rounded_rating,␣
↪palette='viridis')
plt.title('Average ROI by Rounded Movie Rating')
plt.xlabel('Rounded Movie Rating')
plt.ylabel('Average ROI')

plt.savefig('./images/avg_roi_by_rounded_rating.png', bbox_inches='tight')
plt.show()
```

```
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_67205/517332610.py:9:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='rounded_rating', y='roi', data=roi_by_rounded_rating,
palette='viridis')
```



Findings

- Movies around 4 have the highest ROI
- Movies with a rating of 9 have the second-highest ROI

1.4.5 Analysis of Genres and Reviews

- Consider the relationship between genres and reviews

[20]:
```
# convert the genres column from lists to strings
#df['genres'] = df['genres'].apply(lambda x: ', '.join(x) if isinstance(x,␣
↪list) else x)

# convert the genres column from strings to lists
```

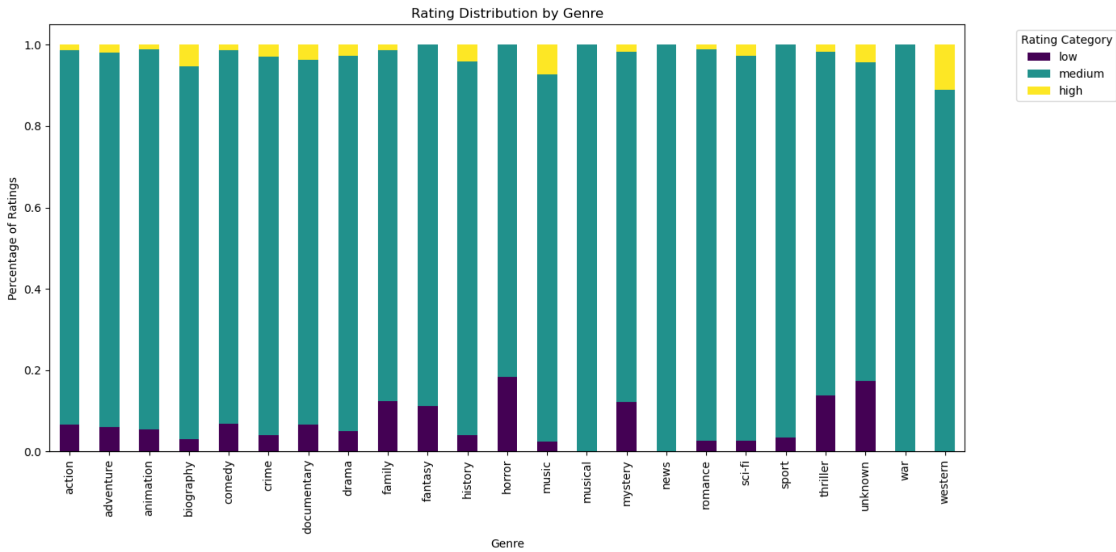
```python
df['genres'] = df['genres'].apply(lambda x: x.split(', ') if isinstance(x, str) else x)

# Example: Classify ratings into low, medium, and high
bins = [0, 5, 8, 10]
labels = ['low', 'medium', 'high']
df['rating_category'] = pd.cut(df['rating'], bins=bins, labels=labels)

# remove 'unkonw' as a value in genres
df = df[df['genres'] != 'unknown']
# If genres are still in a list format, explode them
df_exploded = df.explode('genres')

# One-hot encode the genres
genre_dummies = pd.get_dummies(df_exploded['genres'])

# Combine the one-hot encoded genres with the original DataFrame
df_with_genres = pd.concat([df_exploded, genre_dummies], axis=1)

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Define the features (genres) and the target (rating category)
X = df_with_genres[genre_dummies.columns]
y = df_with_genres['rating_category']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

# Get feature importances
importances = model.feature_importances_
indices = np.argsort(importances)[::-1]

# Print the feature ranking
```

```python
print("Feature ranking:")

for f in range(X_train.shape[1]):
    print(f"{f + 1}. feature {X.columns[indices[f]]}␣
↪({importances[indices[f]]})")

# Group by genre and rating category
genre_rating_distribution = df_with_genres.groupby(['genres',␣
↪'rating_category']).size().unstack().fillna(0)

# Normalize to get the percentage distribution
genre_rating_distribution = genre_rating_distribution.
↪div(genre_rating_distribution.sum(axis=1), axis=0)

# Plot the distribution
genre_rating_distribution.plot(kind='bar', stacked=True, figsize=(14, 7),␣
↪colormap='viridis')
plt.title('Rating Distribution by Genre')
plt.xlabel('Genre')
plt.ylabel('Percentage of Ratings')
plt.legend(title='Rating Category', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

```
              precision    recall  f1-score   support

        high       0.00      0.00      0.00        16
         low       0.00      0.00      0.00        51
      medium       0.91      1.00      0.95       660

    accuracy                           0.91       727
   macro avg       0.30      0.33      0.32       727
weighted avg       0.82      0.91      0.86       727

[[  0   0  16]
 [  0   0  51]
 [  0   0 660]]
Feature ranking:
1. feature horror (0.2199685038100825)
2. feature thriller (0.20477943412722652)
3. feature mystery (0.07496879630571542)
4. feature romance (0.0610146335948188)
5. feature family (0.052683363417426604)
6. feature music (0.035833437176026287)
7. feature biography (0.03136510280505253)
8. feature drama (0.03106077527373496)
9. feature fantasy (0.029929704035040482)
```

```
10. feature unknown (0.02973022027774637)
11. feature western (0.02786821701758435)
12. feature crime (0.027010705158783038)
13. feature action (0.025434384338256573)
14. feature sci-fi (0.023412510205188024)
15. feature comedy (0.022274769774166994)
16. feature history (0.02198402071782992)
17. feature adventure (0.017921815801474134)
18. feature animation (0.016488156348710224)
19. feature documentary (0.016107129925586786)
20. feature sport (0.012974825811870094)
21. feature war (0.011961547603012698)
22. feature musical (0.004355950385734626)
23. feature news (0.0008719960889321283)
```

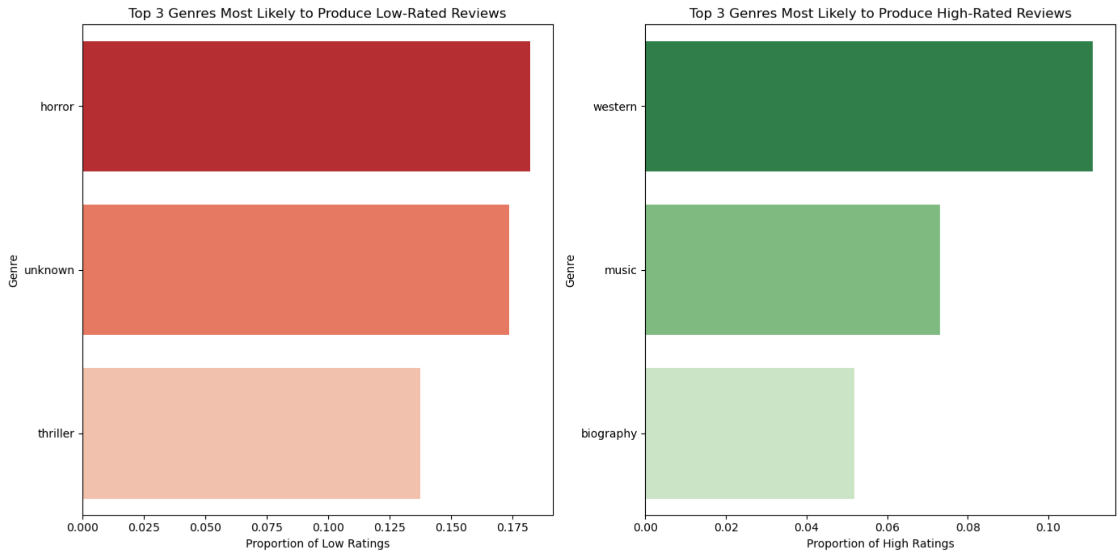
```
/Users/rob/micromamba/envs/learn-env/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1497: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/rob/micromamba/envs/learn-env/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1497: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/rob/micromamba/envs/learn-env/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1497: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_67205/2001348197.py:5
6: FutureWarning: The default of observed=False is deprecated and will be
changed to True in a future version of pandas. Pass observed=False to retain
current behavior or observed=True to adopt the future default and silence this
warning.
  genre_rating_distribution = df_with_genres.groupby(['genres',
'rating_category']).size().unstack().fillna(0)
```



[21]:
```python
# Explode genres if they are stored as lists
df_exploded = df.explode('genres')

# Step 2: Aggregate by Genre
genre_rating_distribution = df_exploded.groupby(['genres', 'rating_category']).
    ↪size().unstack().fillna(0)

# Normalize to get the percentage distribution
genre_rating_distribution = genre_rating_distribution.
    ↪div(genre_rating_distribution.sum(axis=1), axis=0)

# Step 3: Identify Top Genres for Low and High Ratings
top_low_genres = genre_rating_distribution['low'].nlargest(3)
top_high_genres = genre_rating_distribution['high'].nlargest(3)

# Step 4: Visualize the Results
plt.figure(figsize=(14, 7))

# Plot for Low-Rated Reviews
plt.subplot(1, 2, 1)
sns.barplot(x=top_low_genres.values, y=top_low_genres.index, palette='Reds_r')
plt.title('Top 3 Genres Most Likely to Produce Low-Rated Reviews')
plt.xlabel('Proportion of Low Ratings')
plt.ylabel('Genre')

# Plot for High-Rated Reviews
plt.subplot(1, 2, 2)
```

```
sns.barplot(x=top_high_genres.values, y=top_high_genres.index,␣
 ↪palette='Greens_r')
plt.title('Top 3 Genres Most Likely to Produce High-Rated Reviews')
plt.xlabel('Proportion of High Ratings')
plt.ylabel('Genre')

plt.tight_layout()

plt.savefig('./images/top_bottom_3_genres_rating_trend.png',␣
 ↪bbox_inches='tight')
plt.show()
```

```
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_67205/2406264323.py:5
: FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  genre_rating_distribution = df_exploded.groupby(['genres',
'rating_category']).size().unstack().fillna(0)
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_67205/2406264323.py:1
9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_low_genres.values, y=top_low_genres.index, palette='Reds_r')
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_67205/2406264323.py:2
6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_high_genres.values, y=top_high_genres.index,
palette='Greens_r')
```



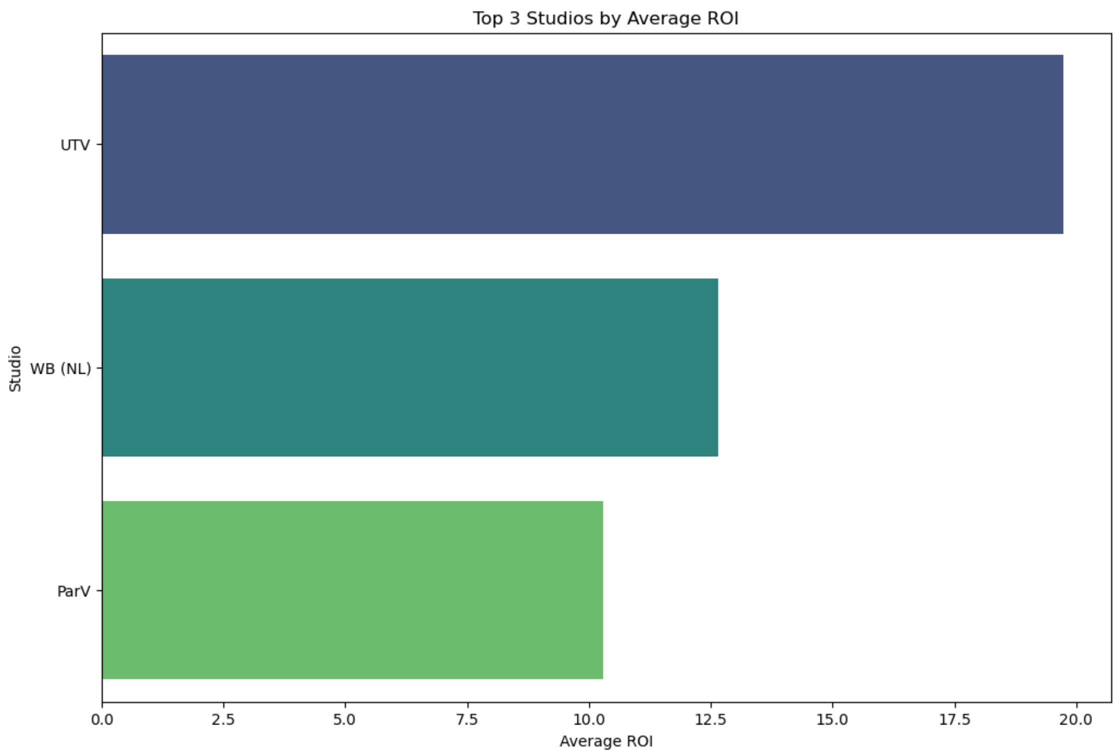
1.4.6 Analysis of Studios

- Consider the relationship between studios and ROI

[22]:
```
# group by Studio and Calculate Average ROI
roi_by_studio = df.groupby('studio')['roi'].mean().reset_index()

# sort the Data by ROI
roi_by_studio = roi_by_studio.sort_values(by='roi', ascending=False)

# select Top 5 and Bottom 5 Studios
top_3_studios = roi_by_studio.head(3)
#bottom_3_studios = roi_by_studio.tail(3)

# combine the top and bottom 5 into one DataFrame
top_bottom_studios = pd.concat([top_3_studios
                                #, bottom_5_studios
                                ])

# create the Bar Chart
plt.figure(figsize=(12, 8))
sns.barplot(x='roi', y='studio', data=top_bottom_studios, palette='viridis')
plt.title('Top 3 Studios by Average ROI')
plt.xlabel('Average ROI')
plt.ylabel('Studio')


# save the image
plt.savefig('./images/top_3_studios_by_roi.png', bbox_inches='tight')
```

```
plt.show()
```

```
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_67205/2505824710.py:1
8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='roi', y='studio', data=top_bottom_studios, palette='viridis')
```



Findings

- UTV has the highest ROI

2 Conclusion & Recommendations

Invest in films that are: - Short - 90 minute runtime or less - Low rated movies (particularly those released early to mid year) - Movies in the Horror/Thriller genre - Produced by a top studio (UTV, WB or ParV) - Released in July