Movie Analysis

Our goal is to identify:

- top genre combinations
- top producers
- directors
- actors
- actresses

We will focus on two metrics:

- average Return on Investment
- average profit

```
import sqlite3
import numpy as np
import pandas as pd
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
import recordlinkage
from recordlinkage.preprocessing import clean
```

IMDb database

Lets connect to the database and explore the data

```
path = "zippedData/im.db"
conn = sqlite3.connect(path)
cursor = conn.cursor()
```

```
# to see all the tables in the database.

imdb_df = pd.read_sql(
    """
    SELECT *
    FROM sqlite_master
    """
, conn
)

imdb_df[imdb_df['type'] == 'table']
```

	type	name	tbl_name	rootpage	sql
0	table	movie_basics	movie_basics	2	CREATE TABLE "movie_basics" (\n"movie_id" TEXT...
1	table	directors	directors	3	CREATE TABLE "directors" (\n"movie_id" TEXT,\n...
2	table	known_for	known_for	4	CREATE TABLE "known_for" (\n"person_id" TEXT,\...
3	table	movie_akas	movie_akas	5	CREATE TABLE "movie_akas" (\n"movie_id" TEXT,\...
4	table	movie_ratings	movie_ratings	6	CREATE TABLE "movie_ratings" (\n"movie_id" TEX...
5	table	persons	persons	7	CREATE TABLE "persons" (\n"person_id" TEXT,\n ...
6	table	principals	principals	8	CREATE TABLE "principals" (\n"movie_id" TEXT,\...
7	table	writers	writers	9	CREATE TABLE "writers" (\n"movie_id" TEXT,\n ...

```
query1 = """ SELECT * FROM movie_basics ORDER BY -start_year LIMIT 10"""
pd.read_sql(query1, conn)
```

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
0	tt5174640	100 Years	100 Years	2115	None	Drama
1	tt5637536	Avatar 5	Avatar 5	2027	None	Action,Adventure,Fantasy
2	tt10300398	Untitled Star Wars Film	Untitled Star Wars Film	2026	None	Fantasy
3	tt3095356	Avatar 4	Avatar 4	2025	None	Action,Adventure,Fantasy
4	tt10300396	Untitled Star Wars Film	Untitled Star Wars Film	2024	None	None
5	tt6149054	Fantastic Beasts and Where to Find Them 5	Fantastic Beasts and Where to Find Them 5	2024	None	Adventure,Family,Fantasy
6	tt10255736	Untitled Marvel Project	Untitled Marvel Project	2023	None	Action
7	tt10298848	Untitled Disney Live-Action Project	Untitled Disney Live-Action Project	2023	None	None
8	tt1757678	Avatar 3	Avatar 3	2023	None	Action,Adventure,Drama
9	tt6258542	Wraith of the Umbra and Eidolon II	Wraith of the Umbra and Eidolon II	2023	None	Adventure,Drama,Fantasy

```
# get column info
cursor.execute("PRAGMA table_info(movie_basics)")
cursor.fetchall()
```

```
[(0, 'movie_id', 'TEXT', 0, None, 0),
 (1, 'primary_title', 'TEXT', 0, None, 0),
 (2, 'original_title', 'TEXT', 0, None, 0),
 (3, 'start_year', 'INTEGER', 0, None, 0),
 (4, 'runtime_minutes', 'REAL', 0, None, 0),
 (5, 'genres', 'TEXT', 0, None, 0)]
```

```
query4 = """ SELECT COUNT(*) FROM movie_basics"""
pd.read_sql(query4, conn)
```

	COUNT(*)
0	146144

```
query2 = """ SELECT * FROM movie_ratings LIMIT 5"""
pd.read_sql(query2, conn)
```

	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

```
query11 = """ SELECT * FROM directors LIMIT 5"""
pd.read_sql(query11, conn)
```

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

Let's combine the tables to include movie data, ratings, as well as the producers, directors, and actors who worked on the movie

```
query201 = """
    WITH ranked_directors AS (
    SELECT
        d.movie_id,
        p.primary_name AS director_name,
        ROW_NUMBER() OVER (PARTITION BY d.movie_id ORDER BY d.person_id) AS director_rank
```

```
        FROM directors d
        JOIN persons p ON d.person_id = p.person_id
        GROUP BY d.movie_id, p.primary_name, d.person_id
    ),

    ranked_principals AS (
        SELECT
            p.movie_id,
            per.primary_name AS person_name,
            p.category,
            ROW_NUMBER() OVER (PARTITION BY p.movie_id, p.category ORDER BY p.person_id) AS person_rank
        FROM principals p
        JOIN persons per ON p.person_id = per.person_id
        WHERE p.category IN ('actor', 'actress', 'producer')
    )

    SELECT
        mr.movie_id AS imdb_movie_id,
        mb.primary_title,
        mr.averagerating AS average_rating,
        mr.numvotes,
        mb.start_year,
        mb.runtime_minutes,
        mb.genres,
        MAX(CASE WHEN rd.director_rank = 1 THEN rd.director_name END) AS director1,
        MAX(CASE WHEN rd.director_rank = 2 THEN rd.director_name END) AS director2,
        MAX(CASE WHEN rp.category = 'actress' AND rp.person_rank = 1 THEN rp.person_name END) AS actress1,
        MAX(CASE WHEN rp.category = 'actress' AND rp.person_rank = 2 THEN rp.person_name END) AS actress2,
        MAX(CASE WHEN rp.category = 'actress' AND rp.person_rank = 3 THEN rp.person_name END) AS actress3,
        MAX(CASE WHEN rp.category = 'actor' AND rp.person_rank = 1 THEN rp.person_name END) AS actor1,
        MAX(CASE WHEN rp.category = 'actor' AND rp.person_rank = 2 THEN rp.person_name END) AS actor2,
        MAX(CASE WHEN rp.category = 'actor' AND rp.person_rank = 3 THEN rp.person_name END) AS actor3,
        MAX(CASE WHEN rp.category = 'actor' AND rp.person_rank = 4 THEN rp.person_name END) AS actor4,
        MAX(CASE WHEN rp.category = 'producer' AND rp.person_rank = 1 THEN rp.person_name END) AS producer1,
        MAX(CASE WHEN rp.category = 'producer' AND rp.person_rank = 2 THEN rp.person_name END) AS producer2
    FROM movie_ratings mr
    JOIN movie_basics mb ON mr.movie_id = mb.movie_id
    LEFT JOIN ranked_directors rd ON mr.movie_id = rd.movie_id
    LEFT JOIN ranked_principals rp ON mr.movie_id = rp.movie_id
    WHERE mr.numvotes >= 30
    GROUP BY
        mr.movie_id,
        mb.primary_title,
        mr.averagerating,
        mr.numvotes,
        mb.start_year,
        mb.runtime_minutes,
        mb.genres
    ORDER BY -mr.numvotes

    """
    pd.read_sql(query201, conn)
```

	imdb_movie_id	primary_title	average_rating	numvotes	start_year	runtime_minutes	genres	director1	dir
0	tt1375666	Inception	8.8	1841066	2010	148.0	Action,Adventure,Sci-Fi	Christopher Nolan	
1	tt1345836	The Dark Knight Rises	8.4	1387769	2012	164.0	Action,Thriller	Christopher Nolan	
2	tt0816692	Interstellar	8.6	1299334	2014	169.0	Adventure,Drama,Sci-Fi	Christopher Nolan	
3	tt1853728	Django Unchained	8.4	1211405	2012	165.0	Drama,Western	Quentin Tarantino	
4	tt0848228	The Avengers	8.1	1183655	2012	143.0	Action,Adventure,Sci-Fi	Joss Whedon	
...	...	...	...	...	...	...	...	...	
43735	tt9378760	Sarah Millican: Control Enthusiast Live	6.9	30	2018	82.0	Comedy	Brian Klein	
43736	tt9442146	Hüddam 2	5.3	30	2019	92.0	Drama,Horror,Thriller	Utku Uçar	
43737	tt9598566	Ave Maria	7.3	30	2018	74.0	Drama	Vipin Radhakrishnan	
43738	tt9613316	Frances Ferguson	6.8	30	2019	74.0	Comedy	Bob Byington	
43739	tt9647980	Patria	7.5	30	2019	89.0	Documentary	Matías Gueilburt	

43740 rows × 18 columns

```
imdb_df = pd.read_sql(query201, conn)
imdb_df.head()
```

	imdb_movie_id	primary_title	average_rating	numvotes	start_year	runtime_minutes	genres	director1	director2
0	tt1375666	Inception	8.8	1841066	2010	148.0	Action,Adventure,Sci-Fi	Christopher Nolan	None
1	tt1345836	The Dark Knight Rises	8.4	1387769	2012	164.0	Action,Thriller	Christopher Nolan	None
2	tt0816692	Interstellar	8.6	1299334	2014	169.0	Adventure,Drama,Sci-Fi	Christopher Nolan	None
3	tt1853728	Django Unchained	8.4	1211405	2012	165.0	Drama,Western	Quentin Tarantino	None
4	tt0848228	The Avengers	8.1	1183655	2012	143.0	Action,Adventure,Sci-Fi	Joss Whedon	None

```
imdb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43740 entries, 0 to 43739
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   imdb_movie_id    43740 non-null  object
 1   primary_title    43740 non-null  object
 2   average_rating   43740 non-null  float64
 3   numvotes         43740 non-null  int64
 4   start_year       43740 non-null  int64
 5   runtime_minutes  41466 non-null  float64
 6   genres           43619 non-null  object
 7   director1        43528 non-null  object
 8   director2        4352 non-null   object
 9   actress1         33042 non-null  object
 10  actress2         19261 non-null  object
 11  actress3         6159 non-null   object
 12  actor1           37258 non-null  object
 13  actor2           32216 non-null  object
 14  actor3           19403 non-null  object
```

```
 15  actor4           6332 non-null   object
 16  producer1        31705 non-null  object
 17  producer2        16412 non-null  object
dtypes: float64(2), int64(2), object(14)
memory usage: 6.0+ MB
```

Movie Budgets dataset

Let's explore the dataset containing movie budgets and revenue data

```
# Read the CSV file
budgets_df = pd.read_csv('zippedData/tn.movie_budgets.csv.gz')

# Display the first few rows of the DataFrame
print(budgets_df.tail(20))
```

```
      id  release_date                          movie production_budget  \
5762  63  Apr 11, 1997                 Pink Flamingos           $12,000
5763  64  Apr 28, 2006        Grip: A Criminal's Story           $12,000
5764  65  Dec 31, 2007                    Tin Can Man           $12,000
5765  66   Mar 9, 2001                        Dayereh           $10,000
5766  67  Apr 28, 2006                          Clean           $10,000
5767  68   Jul 6, 2001                           Cure           $10,000
5768  69  May 28, 2004                On the Downlow           $10,000
5769  70   Apr 1, 1996                           Bang           $10,000
5770  71  Aug 14, 2008  The Rise and Fall of Miss Thang           $10,000
5771  72  May 19, 2015               Family Motocross           $10,000
5772  73  Jan 13, 2012                      Newlyweds            $9,000
5773  74  Feb 26, 1993                     El Mariachi            $7,000
5774  75   Oct 8, 2004                         Primer            $7,000
5775  76  May 26, 2006                         Cavite            $7,000
5776  77  Dec 31, 2004                The Mongol King            $7,000
5777  78  Dec 31, 2018                         Red 11            $7,000
5778  79   Apr 2, 1999                      Following            $6,000
5779  80  Jul 13, 2005    Return to the Land of Wonders            $5,000
5780  81  Sep 29, 2015             A Plague So Pleasant            $1,400
5781  82   Aug 5, 2005                My Date With Drew            $1,100

     domestic_gross worldwide_gross
5762       $413,802        $413,802
5763         $1,336          $1,336
5764             $0              $0
5765       $673,780        $673,780
5766       $138,711        $138,711
5767        $94,596         $94,596
5768         $1,987          $1,987
5769           $527            $527
5770           $401            $401
5771             $0              $0
5772         $4,584          $4,584
5773     $2,040,920      $2,041,928
5774       $424,760        $841,926
5775        $70,071         $71,644
5776           $900            $900
5777             $0              $0
5778        $48,482        $240,495
5779         $1,338          $1,338
5780             $0              $0
5781       $181,041        $181,041
```

```
budgets_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5782 non-null   int64
 1   release_date       5782 non-null   object
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   object
 4   domestic_gross     5782 non-null   object
 5   worldwide_gross    5782 non-null   object
dtypes: int64(1), object(5)
memory usage: 271.2+ KB
```

```
# convert release_dat column to datetime
budgets_df['release_date'] = pd.to_datetime(budgets_df['release_date'], format='%b %d, %Y')


# add a new column with just the year
budgets_df['release_year'] = budgets_df['release_date'].dt.year
budgets_df.head()
```

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	release_year
0	1	2009-12-18	Avatar	$425,000,000	$760,507,625	$2,776,345,279	2009
1	2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875	2011
2	3	2019-06-07	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350	2019
3	4	2015-05-01	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963	2015
4	5	2017-12-15	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747	2017

```
budgets_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5782 non-null   int64
 1   release_date       5782 non-null   datetime64[ns]
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   object
 4   domestic_gross     5782 non-null   object
 5   worldwide_gross    5782 non-null   object
 6   release_year       5782 non-null   int32
dtypes: datetime64[ns](1), int32(1), int64(1), object(4)
memory usage: 293.7+ KB
```

```
imdb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43740 entries, 0 to 43739
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   imdb_movie_id    43740 non-null  object
 1   primary_title    43740 non-null  object
 2   average_rating   43740 non-null  float64
 3   numvotes         43740 non-null  int64
 4   start_year       43740 non-null  int64
 5   runtime_minutes  41466 non-null  float64
 6   genres           43619 non-null  object
 7   director1        43528 non-null  object
 8   director2        4352 non-null   object
 9   actress1         33042 non-null  object
 10  actress2         19261 non-null  object
 11  actress3         6159 non-null   object
 12  actor1           37258 non-null  object
 13  actor2           32216 non-null  object
 14  actor3           19403 non-null  object
 15  actor4           6332 non-null   object
 16  producer1        31705 non-null  object
 17  producer2        16412 non-null  object
dtypes: float64(2), int64(2), object(14)
memory usage: 6.0+ MB
```

```
budgets_df['release_year'] = budgets_df['release_year'].astype('int64')
```

```
# Check for missing values in each column
budgets_df.isnull().sum()
```

```
id                   0
release_date         0
movie                0
production_budget    0
domestic_gross       0
worldwide_gross      0
release_year         0
dtype: int64
```

```
imdb_df.isnull().sum()
```

```
imdb_movie_id          0
primary_title          0
average_rating         0
numvotes               0
start_year             0
runtime_minutes     2274
genres               121
director1            212
director2          39388
actress1           10698
actress2           24479
actress3           37581
actor1              6482
actor2             11524
actor3             24337
actor4             37408
producer1          12035
producer2          27328
dtype: int64
```

Start coding or generate with AI.

Combining IMDb and Movie Budgets datasets

Normalize titles by converting to lowercase and stripping spaces

```
budgets_df['movie'] = budgets_df['movie'].str.lower().str.strip()
imdb_df['primary_title'] = imdb_df['primary_title'].str.lower().str.strip()
```

Years have to match exactly but not movie titles

```
# Create an indexer and define the comparison criteria
indexer = recordlinkage.Index()
indexer.block(left_on='release_year', right_on='start_year')
candidate_links = indexer.index(budgets_df, imdb_df)

compare = recordlinkage.Compare()
compare.exact('release_year', 'start_year', label='year')
compare.string('movie', 'primary_title', method='jarowinkler', threshold=0.95, label='title')
features = compare.compute(candidate_links, budgets_df, imdb_df)
```

```
# Filter matches based on a threshold
matches = features[features.sum(axis=1) > 1.5]
```

Now Let's merge the data frames based on the matches

```
budgets_df['index'] = budgets_df.index
imdb_df['index'] = imdb_df.index

matches.reset_index(inplace=True)
merged_df = pd.merge(budgets_df, matches, left_on='index', right_on='level_0')
merged_df = pd.merge(merged_df, imdb_df, left_on='level_1', right_on='index')

# Drop unnecessary columns and duplicates
merged_df.drop(columns=['index_x', 'index_y', 'level_0', 'level_1'], inplace=True)
merged_df.drop_duplicates(inplace=True)
merged_df
```

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	release_year	year	title	imdb_movie_ic
0	2	2011-05-20	pirates of the caribbean: on stranger tides	$410,600,000	$241,063,875	$1,045,663,875	2011	1	1.0	tt129865(
1	3	2019-06-07	dark phoenix	$350,000,000	$42,762,350	$149,762,350	2019	1	1.0	tt656570‹
2	4	2015-05-01	avengers: age of ultron	$330,600,000	$459,005,868	$1,403,013,963	2015	1	1.0	tt239542‹
3	7	2018-04-27	avengers: infinity war	$300,000,000	$678,815,482	$2,048,134,200	2018	1	1.0	tt415475(
4	9	2017-11-17	justice league	$300,000,000	$229,024,295	$655,945,209	2017	1	1.0	tt097401‹
...	...	...	...	...	...	...	...	...	...	..
1595	35	2013-10-25	her cry: la llorona investigation	$35,000	$0	$0	2013	1	1.0	tt246921(
1596	49	2015-09-01	exeter	$25,000	$0	$489,792	2015	1	1.0	tt194504‹
1597	52	2015-12-01	dutch kills	$25,000	$0	$0	2015	1	1.0	tt275906(
1598	59	2011-11-25	the ridges	$17,300	$0	$0	2011	1	1.0	tt178193‹
1599	62	2014-12-31	stories of our lives	$15,000	$0	$0	2014	1	1.0	tt397361‹

1600 rows × 27 columns

```
# rearrange columns
columns = list(merged_df.columns)

# Move 'primary_title' next to 'movie'
columns.insert(columns.index('movie') + 1, columns.pop(columns.index('primary_title')))
merged_df = merged_df[columns]
merged_df
```

	id	release_date	movie	primary_title	production_budget	domestic_gross	worldwide_gross	release_year	year	title
0	2	2011-05-20	pirates of the caribbean: on stranger tides	pirates of the caribbean: on stranger tides	$410,600,000	$241,063,875	$1,045,663,875	2011	1	1.(
1	3	2019-06-07	dark phoenix	dark phoenix	$350,000,000	$42,762,350	$149,762,350	2019	1	1.(
2	4	2015-05-01	avengers: age of ultron	avengers: age of ultron	$330,600,000	$459,005,868	$1,403,013,963	2015	1	1.(
3	7	2018-04-27	avengers: infinity war	avengers: infinity war	$300,000,000	$678,815,482	$2,048,134,200	2018	1	1.(
4	9	2017-11-17	justice league	justice league	$300,000,000	$229,024,295	$655,945,209	2017	1	1.(
...	...	...	...	...	...	...	...	...	...	..
1595	35	2013-10-25	her cry: la llorona investigation	her cry: la llorona investigation	$35,000	$0	$0	2013	1	1.(
1596	49	2015-09-01	exeter	exeter	$25,000	$0	$489,792	2015	1	1.(
1597	52	2015-12-01	dutch kills	dutch kills	$25,000	$0	$0	2015	1	1.(
1598	59	2011-11-25	the ridges	the ridges	$17,300	$0	$0	2011	1	1.(
1599	62	2014-12-31	stories of our lives	stories of our lives	$15,000	$0	$0	2014	1	1.(

1600 rows × 27 columns

Filter rows where 'movie' does not equal 'primary_title' to manually explore the data

```
non_exact_matches = merged_df[merged_df['movie'] != merged_df['primary_title']]

# Set display options to show all rows
pd.set_option('display.max_rows', None)

non_exact_matches
```

	id	release_date	movie	primary_title	production_budget	domestic_gross	worldwide_gross	release_year	yea
33	49	2017-05-05	guardians of the galaxy vol 2	guardians of the galaxy vol. 2	$200,000,000	$389,813,101	$862,316,233	2017	
65	92	2018-07-27	mission: impossibleâ€ fallout	mission: impossible - fallout	$178,000,000	$220,159,104	$787,456,552	2018	
97	32	2014-11-05	interstellar	interstelar	$165,000,000	$188,017,894	$666,379,375	2014	
103	40	2013-05-24	fast and furious 6	fast & furious 6	$160,000,000	$238,679,850	$789,300,444	2013	
110	49	2015-07-01	terminator: genisys	terminator genisys	$155,000,000	$89,760,956	$432,150,894	2015	
122	76	2019-05-10	pokã©mon: detective pikachu	pokémon detective pikachu	$150,000,000	$139,507,806	$411,258,433	2019	
142	25	2012-11-16	the twilight saga: breaking dawn, part 2	the twilight saga: breaking dawn - part 2	$136,200,000	$292,324,737	$829,724,737	2012	
145	27	2015-12-25	the revenant	the event	$135,000,000	$183,637,894	$532,938,302	2015	
166	57	2011-11-18	the twilight saga: breaking dawn, part 1	the twilight saga: breaking dawn - part 1	$127,500,000	$281,287,133	$689,420,051	2011	
169	61	2011-07-15	harry potter and the deathly hallows: part ii	harry potter and the deathly hallows: part 2	$125,000,000	$381,193,157	$1,341,693,157	2011	
171	64	2010-11-19	harry potter and the deathly hallows: part i	harry potter and the deathly hallows: part 1	$125,000,000	$296,131,568	$960,431,568	2010	
182	80	2016-12-21	assassinâ€ s creed	assassin's creed	$125,000,000	$54,647,948	$240,759,682	2016	
191	1	2010-12-17	how do you know?	how do you know	$120,000,000	$30,212,620	$49,628,177	2010	
193	2	2010-06-23	knight and day	night and day	$117,000,000	$76,423,035	$258,751,370	2010	
196	10	2016-04-22	the huntsman: winterâ€ s war	the huntsman: winter's war	$115,000,000	$48,003,015	$165,149,302	2016	
206	30	2016-09-30	miss peregrineâ€ s home for peculiar children	miss peregrine's home for peculiar children	$110,000,000	$87,242,834	$295,986,876	2016	
231	96	2014-08-15	the expendables 3	the extendables	$100,000,000	$39,322,544	$209,461,378	2014	
253	44	2018-12-14	spider-man: into the spider-verse 3d	spider-man: into the spider-verse	$90,000,000	$190,173,195	$375,381,768	2018	
296	93	2013-12-20	walking with dinosaurs	walking with dinosaurs 3d	$80,000,000	$36,076,121	$123,368,842	2013	
311	45	2011-11-11	immortals	immortalitas	$75,000,000	$83,504,017	$211,562,435	2011	
320	10	2011-03-11	battle: los angeles	battle los angeles	$70,000,000	$83,552,429	$213,463,976	2011	
324	19	2012-01-20	underworld: awakening	underworld awakening	$70,000,000	$62,321,039	$160,379,930	2012	
364	41	2019-01-11	a dogâ€ s way home	a dog's way home	$61,000,000	$41,952,715	$81,149,689	2019	
432	100	2018-12-14	the mule	the mute	$50,000,000	$103,804,407	$170,857,676	2018	
435	8	2011-07-29	crazy, stupid, love	crazy, stupid, love.	$50,000,000	$84,351,197	$147,142,328	2011	
439	24	2014-07-18	planes: fire and rescue	planes: fire & rescue	$50,000,000	$59,157,732	$156,399,644	2014	

447	41	2014-10-10	the judge	the judgment	$50,000,000	$47,119,388	$76,119,388	2014
464	4	2013-03-15	upside down	upsidedown	$50,000,000	$102,118	$26,387,039	2013
473	42	2016-12-23	silence	silenced	$46,500,000	$7,100,177	$23,726,626	2016
495	35	2013-06-28	the heat	the east	$43,000,000	$159,581,587	$229,727,774	2013
496	40	2018-11-09	the girl in the spiderâ€ s web	the girl in the spider's web	$43,000,000	$14,828,555	$34,983,342	2018
511	81	2019-05-17	john wick: chapter 3 â€ parabellum	john wick: chapter 3 - parabellum	$40,000,000	$141,744,320	$256,498,033	2019
515	94	2017-02-10	john wick: chapter two	john wick: chapter 2	$40,000,000	$92,029,184	$171,350,009	2017
544	70	2011-12-25	extremely loud and incredibly close	extremely loud & incredibly close	$40,000,000	$31,847,881	$55,247,881	2011
560	30	2016-11-11	billy lynnâ€ s long halftime walk	billy lynn's long halftime walk	$40,000,000	$1,738,477	$30,230,402	2016
562	41	2014-12-31	dragon nest warriors' dawn	dragon nest: warriors' dawn	$40,000,000	$0	$734,423	2014
563	42	2018-12-31	the crow	the row	$40,000,000	$0	$0	2018
588	6	2011-02-11	gnomeo and juliet	gnomeo & juliet	$36,000,000	$99,967,670	$193,737,977	2011
619	3	2016-09-16	bridget jonesâ€ s baby	bridget jones's baby	$35,000,000	$24,139,805	$205,822,688	2016
668	65	2017-08-18	the hitmanâ€ s bodyguard	the hitman's bodyguard	$30,000,000	$75,468,583	$172,778,667	2017
708	79	2011-04-29	hoodwinked too: hood vs. evil	hoodwinked too! hood vs. evil	$30,000,000	$10,143,779	$23,353,111	2011
710	84	2017-02-03	the space between us	the space between	$30,000,000	$7,885,294	$16,481,405	2017
742	19	2016-11-23	rules donâ€ t apply	rules don't apply	$26,700,000	$3,652,206	$3,871,448	2016
758	86	2017-01-27	a dogâ€ s purpose	a dog's purpose	$25,000,000	$64,321,890	$203,671,625	2017
853	39	2017-11-17	the star	the stray	$20,000,000	$40,847,995	$62,758,010	2017
861	61	2013-04-12	scary movie v	scary movie 5	$20,000,000	$32,015,787	$78,613,981	2013
865	80	2010-09-17	alpha and omega 3d	alpha and omega	$20,000,000	$25,107,267	$48,958,353	2010
871	5	2012-10-26	silent hill: revelation 3d	silent hill: revelation	$20,000,000	$17,530,219	$55,975,672	2012
872	7	2017-03-31	the zookeeperâ€ s wife	the zookeeper's wife	$20,000,000	$17,445,186	$26,308,749	2017
902	11	2015-05-29	survivor	survivors	$20,000,000	$0	$1,703,281	2015
921	64	2014-05-09	neighbors	neighbours	$18,000,000	$150,086,800	$270,944,428	2014
940	40	2017-03-17	t2: trainspotting	t2 trainspotting	$18,000,000	$2,402,004	$42,091,262	2017
946	53	2014-08-22	the prince	the principle	$18,000,000	$0	$0	2014
974	97	2010-11-26	the kingâ€ s speech	the king's speech	$15,000,000	$138,797,449	$430,821,168	2010
988	39	2018-11-23	the favourite	the favorite	$15,000,000	$34,366,783	$94,113,929	2018

1001	13	2013-01-04	promised land	promise land	$15,000,000	$7,597,898	$12,394,562	2013
1065	77	2016-08-12	hell or high water	hell or high waters	$12,000,000	$27,007,844	$37,584,304	2016
1085	55	2015-10-30	dancin' it's on	dancin': it's on!	$12,000,000	$0	$0	2015
1105	51	2015-06-05	insidious chapter 3	insidious: chapter 3	$10,000,000	$52,218,558	$120,453,155	2015
1144	69	2016-03-30	everybody wants some	everybody wants some!!	$10,000,000	$3,400,278	$5,437,126	2016
1160	33	2010-12-31	the reef	the tree	$10,000,000	$0	$15,037,867	2010
1240	99	2013-05-31	the east	the heat	$6,500,000	$2,274,649	$3,027,956	2013
1242	99	2013-05-31	the east	the past	$6,500,000	$2,274,649	$3,027,956	2013
1266	53	2013-09-13	insidious chapter 2	insidious: chapter 2	$5,000,000	$83,586,447	$161,921,515	2013
1270	56	2015-09-11	the visit	the visitor	$5,000,000	$65,206,105	$98,677,816	2015
1273	59	2012-10-19	paranormal activity 4	paranormal captivity	$5,000,000	$53,900,335	$142,817,992	2012
1291	8	2016-07-15	hillaryâ€ s america: the secret history of the...	hillary's america: the secret history of the d...	$5,000,000	$13,099,931	$13,099,931	2016
1294	20	2014-05-09	momsâ€ night out	moms' night out	$5,000,000	$10,429,707	$10,537,341	2014
1319	18	2012-09-14	barfi	barfi!	$4,600,000	$2,804,874	$36,751,984	2012
1325	56	2014-11-07	fugly	fugly!	$4,500,000	$0	$0	2014
1349	94	2016-02-19	the witch	the witching	$3,500,000	$25,138,705	$40,454,520	2016
1385	72	2015-08-14	amnesiac	amnesia	$3,000,000	$0	$0	2015
1392	13	2012-07-13	2016: obamaâ€ s america	2016: obama's america	$2,500,000	$33,349,941	$33,349,941	2012
1401	44	2013-06-21	alien uprising	alien rising	$2,500,000	$0	$0	2013
1432	79	2013-04-01	stitches	stitch	$2,000,000	$0	$63,555	2013
1476	77	2018-02-06	blood feast	blood fest	$1,200,000	$8,708	$8,708	2018
1479	87	2015-02-03	bleeding hearts	bleeding heart	$1,200,000	$0	$0	2015
1516	48	2012-08-31	for a good time, call	for a good time, call...	$850,000	$1,251,749	$1,386,088	2012
1518	54	2012-08-03	celeste and jesse forever	celeste & jesse forever	$840,000	$3,103,407	$3,787,689	2012
1565	33	2014-03-14	the word	the m word	$200,000	$3,648	$3,648	2014
1568	43	2011-09-23	weekend	weekender	$190,000	$484,592	$1,577,585	2011
1572	60	2015-04-17	antarctic edge: 70âº south	antarctic edge: 70° south	$150,000	$7,193	$7,193	2015
1581	93	2014-12-31	dude, where's my dog	dude, where's my dog?!	$100,000	$0	$0	2014
1585	6	2011-12-31	absentia	absent	$70,000	$0	$8,555	2011

84 rows × 27 columns

```
# Reset display options to default
pd.reset_option('display.max_rows')
```

Let's remove movies with poorly matched titles by index

```
poor_match = [97, 145, 447, 495, 563, 853, 946, 1160, 1240, 1242]
merged_df = merged_df.drop(poor_match)
```

```
merged_df.columns
```

```
Index(['id', 'release_date', 'movie', 'primary_title', 'production_budget',
       'domestic_gross', 'worldwide_gross', 'release_year', 'year', 'title',
       'imdb_movie_id', 'average_rating', 'numvotes', 'start_year',
       'runtime_minutes', 'genres', 'director1', 'director2', 'actress1',
       'actress2', 'actress3', 'actor1', 'actor2', 'actor3', 'actor4',
       'producer1', 'producer2'],
      dtype='object')
```

We can remove redundunt columns

```
col_to_remove = ['id', 'movie', 'release_year', 'year', 'title', 'start_year', 'runtime_minutes']
merged_df = merged_df.drop(col_to_remove, axis=1)
```

```
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1590 entries, 0 to 1599
Data columns (total 20 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   release_date       1590 non-null   datetime64[ns]
 1   primary_title      1590 non-null   object
 2   production_budget  1590 non-null   object
 3   domestic_gross     1590 non-null   object
 4   worldwide_gross    1590 non-null   object
 5   imdb_movie_id      1590 non-null   object
 6   average_rating     1590 non-null   float64
 7   numvotes           1590 non-null   int64
 8   genres             1590 non-null   object
 9   director1          1589 non-null   object
 10  director2          139 non-null    object
 11  actress1           1383 non-null   object
 12  actress2           730 non-null    object
 13  actress3           177 non-null    object
 14  actor1             1534 non-null   object
 15  actor2             1379 non-null   object
 16  actor3             825 non-null    object
 17  actor4             179 non-null    object
 18  producer1          1380 non-null   object
 19  producer2          997 non-null    object
dtypes: datetime64[ns](1), float64(1), int64(1), object(17)
memory usage: 260.9+ KB
```

```
# convert string columns to numeric
for col in ['production_budget', 'domestic_gross', 'worldwide_gross']:
    merged_df[col] = merged_df[col].str.replace('$', '', regex=False)
    merged_df[col] = merged_df[col].str.replace(',', '', regex=False)
    merged_df[col] = pd.to_numeric(merged_df[col])
```

```
# Calculate profit
merged_df['profit'] = merged_df['worldwide_gross'] - merged_df['production_budget']
```

```
# Calculate ROI
merged_df['roi'] = ((merged_df['profit']-merged_df['production_budget']) / merged_df['production_budget']) * 100
```

```
merged_df.head()
```

	release_date	primary_title	production_budget	domestic_gross	worldwide_gross	imdb_movie_id	average_rating	numvotes	
0	2011-05-20	pirates of the caribbean: on stranger tides	410600000	241063875	1045663875	tt1298650	6.6	447624	A
1	2019-06-07	dark phoenix	350000000	42762350	149762350	tt6565702	6.0	24451	
2	2015-05-01	avengers: age of ultron	330600000	459005868	1403013963	tt2395427	7.3	665594	
3	2018-04-27	avengers: infinity war	300000000	678815482	2048134200	tt4154756	8.5	670926	
4	2017-11-17	justice league	300000000	229024295	655945209	tt0974015	6.5	329135	A

5 rows × 22 columns

Split the genres column into multiple columns

```
merged_df[['genre1', 'genre2', 'genre3']] = merged_df['genres'].str.split(',', expand=True)# expand=True: Ensures that the resul
```

```
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1590 entries, 0 to 1599
Data columns (total 25 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   release_date       1590 non-null   datetime64[ns]
 1   primary_title      1590 non-null   object
 2   production_budget  1590 non-null   int64
 3   domestic_gross     1590 non-null   int64
 4   worldwide_gross    1590 non-null   int64
 5   imdb_movie_id      1590 non-null   object
 6   average_rating     1590 non-null   float64
 7   numvotes           1590 non-null   int64
 8   genres             1590 non-null   object
 9   director1          1589 non-null   object
 10  director2          139 non-null    object
 11  actress1           1383 non-null   object
 12  actress2           730 non-null    object
 13  actress3           177 non-null    object
 14  actor1             1534 non-null   object
 15  actor2             1379 non-null   object
 16  actor3             825 non-null    object
 17  actor4             179 non-null    object
 18  producer1          1380 non-null   object
 19  producer2          997 non-null    object
 20  profit             1590 non-null   int64
 21  roi                1590 non-null   float64
 22  genre1             1590 non-null   object
 23  genre2             1408 non-null   object
 24  genre3             1046 non-null   object
dtypes: datetime64[ns](1), float64(2), int64(5), object(17)
memory usage: 323.0+ KB
```

Genre Analisys

We can set the float format to display up to a specific number of decimal places

```
pd.set_option('display.float_format', lambda x: '%.2f' % x if x != 0 else 0)
```

Let's melting the genre columns and group by genre and calculate metrics

```
# Melting the genre columns
melted_genres_df = merged_df.melt(id_vars=['profit', 'roi', 'average_rating', 'numvotes', 'production_budget'],
                                  value_vars=['genre1', 'genre2', 'genre3'],
                                  value_name='genre').dropna(subset=['genre'])
```

```
# Group by genre and calculate metrics
genre_analysis = melted_genres_df.groupby('genre').agg(
    avg_budget=('production_budget', 'mean'),
    avg_profit=('profit', 'mean'),
    avg_roi=('roi', 'mean'),
    avg_rating=('average_rating', 'mean'),
    total_votes=('numvotes', 'sum')
).reset_index()

genre_analysis = genre_analysis.sort_values(by=['avg_roi', 'avg_rating'], ascending=[False, False])

genre_analysis
```

	genre	avg_budget	avg_profit	avg_roi	avg_rating	total_votes
14	Mystery	25125550.38	71788590.80	782.92	6.10	16594459
11	Horror	18517602.59	53785183.95	701.17	5.45	12523815
18	Thriller	33116149.45	85528899.22	432.83	6.00	32491608
12	Music	17200754.72	61393441.79	182.88	6.33	3572520
3	Biography	25500143.88	56630786.06	158.11	7.01	15425091
2	Animation	93888392.86	262007659.49	157.61	6.49	13066701
16	Sci-Fi	92910000.00	243146307.59	156.71	6.41	37240239
15	Romance	22858128.08	46024403.09	148.43	6.30	16168782
7	Drama	27205330.64	49728696.69	119.25	6.53	75170946
9	Fantasy	85167537.31	168263427.75	118.22	6.11	21040550
1	Adventure	105360923.48	245259640.20	109.90	6.44	74095851
10	History	33772727.27	58648039.41	108.04	6.86	4561316
4	Comedy	40577748.64	92092923.88	106.71	6.19	50991232
6	Documentary	5360756.76	11935614.24	82.96	6.62	341119
8	Family	65006000.00	120873862.53	67.79	6.05	7224888
0	Action	79215253.16	160527513.80	62.33	6.25	83036951
5	Crime	33050000.00	51401423.40	17.89	6.31	27679957
13	Musical	40035000.00	144864392.90	12.86	5.48	615712
17	Sport	24084558.82	27435814.06	9.99	6.89	2395967
20	Western	55983333.33	47182244.17	-41.26	6.30	2158691
19	War	27416666.67	27183357.72	-66.51	6.45	1290577

Now let's group by genre combinations and sort by the highest ROI

```
# Grouping by genre combinations and calculating metrics
genre_combinations_analysis = merged_df.groupby('genres').agg(
    count=('genres', 'size'),  # Count the number of occurrences of each genre combination
    avg_budget=('production_budget', 'mean'),
    avg_profit=('profit', 'mean'),
    avg_roi=('roi', 'mean'),
    avg_rating=('average_rating', 'mean'),
    total_votes=('numvotes', 'sum')
).reset_index()

# Filter for genre combinations with at least 30 occurrences
genre_combinations_analysis_filtered = genre_combinations_analysis[genre_combinations_analysis['count'] >= 30]

# BY ROI
# Sort by average ROI to get the top ROI combinations
genre_combinations_analysis_filtered = genre_combinations_analysis_filtered.sort_values(by='avg_roi', ascending=False)

# Display the top 20 genre combinations based on average ROI
genre_combinations_analysis_filtered.head(20)
```

	genres	count	avg_budget	avg_profit	avg_roi	avg_rating	total_votes
225	Horror,Mystery,Thriller	33	9721969.70	97892046.21	2614.19	5.53	2662668
203	Drama,Romance	35	22846857.14	54608913.46	270.25	6.82	3008082
63	Adventure,Animation,Comedy	72	100090277.78	292000004.51	192.66	6.41	8178889
12	Action,Adventure,Sci-Fi	55	169716363.64	467118871.35	166.97	6.73	22181507
132	Comedy,Drama,Romance	58	18023965.52	29467109.05	122.75	6.38	4758146
152	Comedy,Romance	44	25565909.09	56749376.34	119.30	5.85	3215413
172	Drama	61	11451672.13	14978908.18	113.12	6.70	2879926
124	Comedy,Drama	52	16168461.54	27646498.40	94.45	6.44	2985850
8	Action,Adventure,Fantasy	33	149109090.91	247778337.39	48.95	6.12	7794508
21	Action,Comedy,Crime	33	39930000.00	58148700.58	44.70	6.03	3794183
117	Comedy	63	29061190.48	41244930.44	24.59	5.54	3906958
32	Action,Crime,Drama	39	27548717.95	25689928.59	2.77	6.35	3922818
6	Action,Adventure,Drama	30	88153333.33	84898672.47	-17.70	5.96	3590583

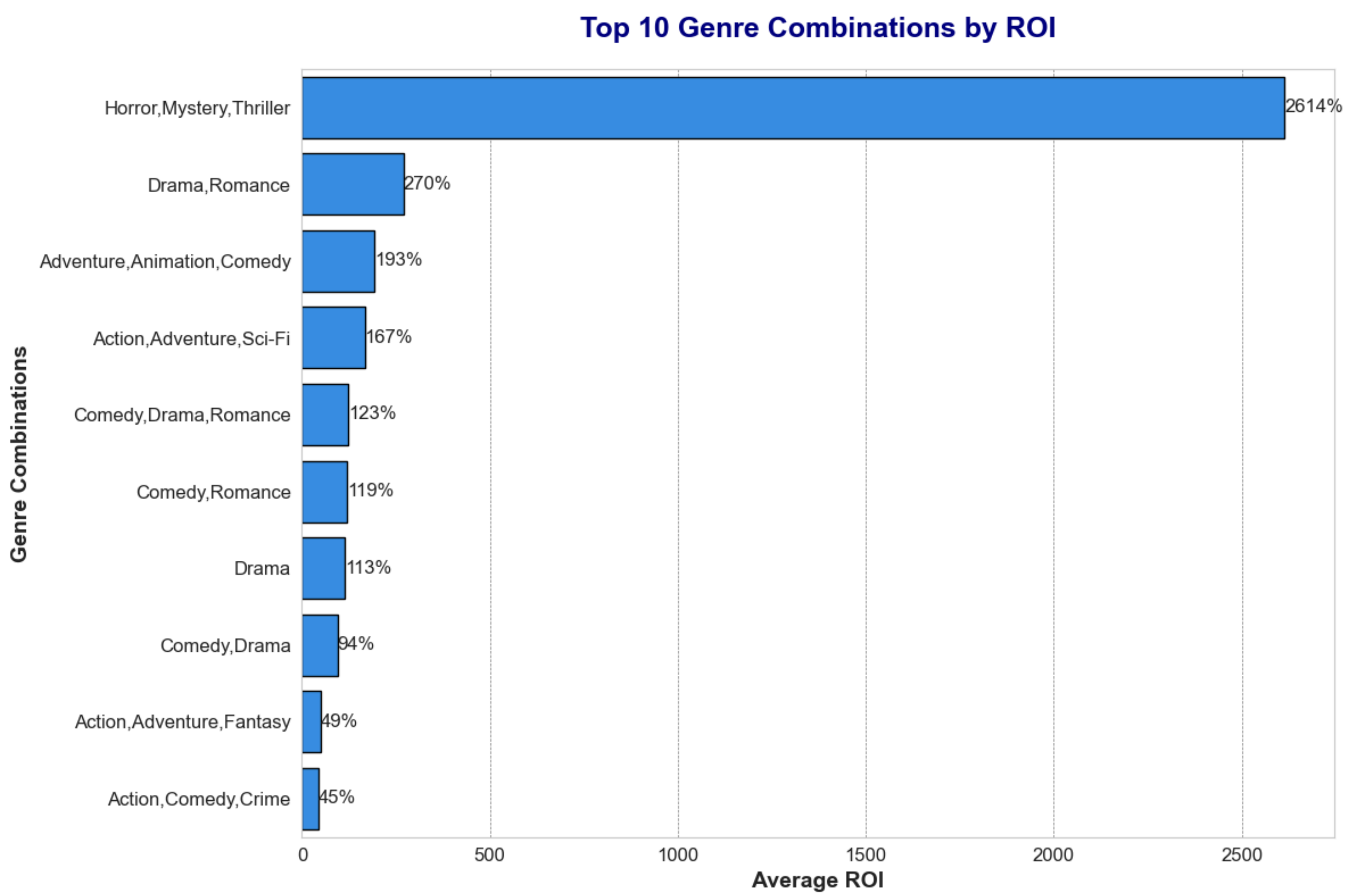
```
# Grouping by genre combinations and calculating metrics
genre_combinations_analysis = merged_df.groupby('genres').agg(
    count=('genres', 'size'),  # Count the number of occurrences of each genre combination
    avg_budget=('production_budget', 'mean'),
    avg_profit=('profit', 'mean'),
    avg_roi=('roi', 'mean'),
    avg_rating=('average_rating', 'mean'),
    total_votes=('numvotes', 'sum')
).reset_index()

# Filter for genre combinations with at least 30 occurrences
genre_combinations_analysis_filtered = genre_combinations_analysis[genre_combinations_analysis['count'] >= 30]

# BY ROI
# Sort by average ROI to get the top ROI combinations
genre_combinations_analysis_filtered = genre_combinations_analysis_filtered.sort_values(by='avg_roi', ascending=False)

# Display the top 20 genre combinations based on average ROI
genre_combinations_analysis_filtered.head(20)
```


	genres	count	avg_budget	avg_profit	avg_roi	avg_rating	total_votes
225	Horror,Mystery,Thriller	33	9721969.70	97892046.21	2614.19	5.53	2662668
203	Drama,Romance	35	22846857.14	54608913.46	270.25	6.82	3008082
63	Adventure,Animation,Comedy	72	100090277.78	292000004.51	192.66	6.41	8178889
12	Action,Adventure,Sci-Fi	55	169716363.64	467118871.35	166.97	6.73	22181507
132	Comedy,Drama,Romance	58	18023965.52	29467109.05	122.75	6.38	4758146
152	Comedy,Romance	44	25565909.09	56749376.34	119.30	5.85	3215413
172	Drama	61	11451672.13	14978908.18	113.12	6.70	2879926
124	Comedy,Drama	52	16168461.54	27646498.40	94.45	6.44	2985850
8	Action,Adventure,Fantasy	33	149109090.91	247778337.39	48.95	6.12	7794508
21	Action,Comedy,Crime	33	39930000.00	58148700.58	44.70	6.03	3794183
117	Comedy	63	29061190.48	41244930.44	24.59	5.54	3906958
32	Action,Crime,Drama	39	27548717.95	25689928.59	2.77	6.35	3922818
6	Action,Adventure,Drama	30	88153333.33	84898672.47	-17.70	5.96	3590583

Let's include some visualizations

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set modern and professional style
```

```
sns.set_style("whitegrid")
plt.figure(figsize=(12, 8))

# Sort by 'avg_roi' to get the top 10 genre combinations
top_10_genres_by_roi = genre_combinations_analysis_filtered.nlargest(10, 'avg_roi')

# Create the bar plot with dodgerblue color
barplot = sns.barplot(x='avg_roi', y='genres', data=top_10_genres_by_roi, color='dodgerblue', edgecolor='black')

# Customize title and labels for a professional look
barplot.set_title('Top 10 Genre Combinations by ROI', fontsize=18, weight='bold', color='navy', pad=20)
barplot.set_xlabel('Average ROI', fontsize=14, weight='bold')
barplot.set_ylabel('Genre Combinations', fontsize=14, weight='bold')

# Add grid for better readability
barplot.xaxis.grid(True, color='gray', linestyle='--', linewidth=0.5)

# Customize ticks
barplot.tick_params(labelsize=12)

# Add value labels for each bar, rounded to integer
for i in barplot.containers:
    barplot.bar_label(i, fmt='%.0f%%', fontsize=12)  # Rounded to integer

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

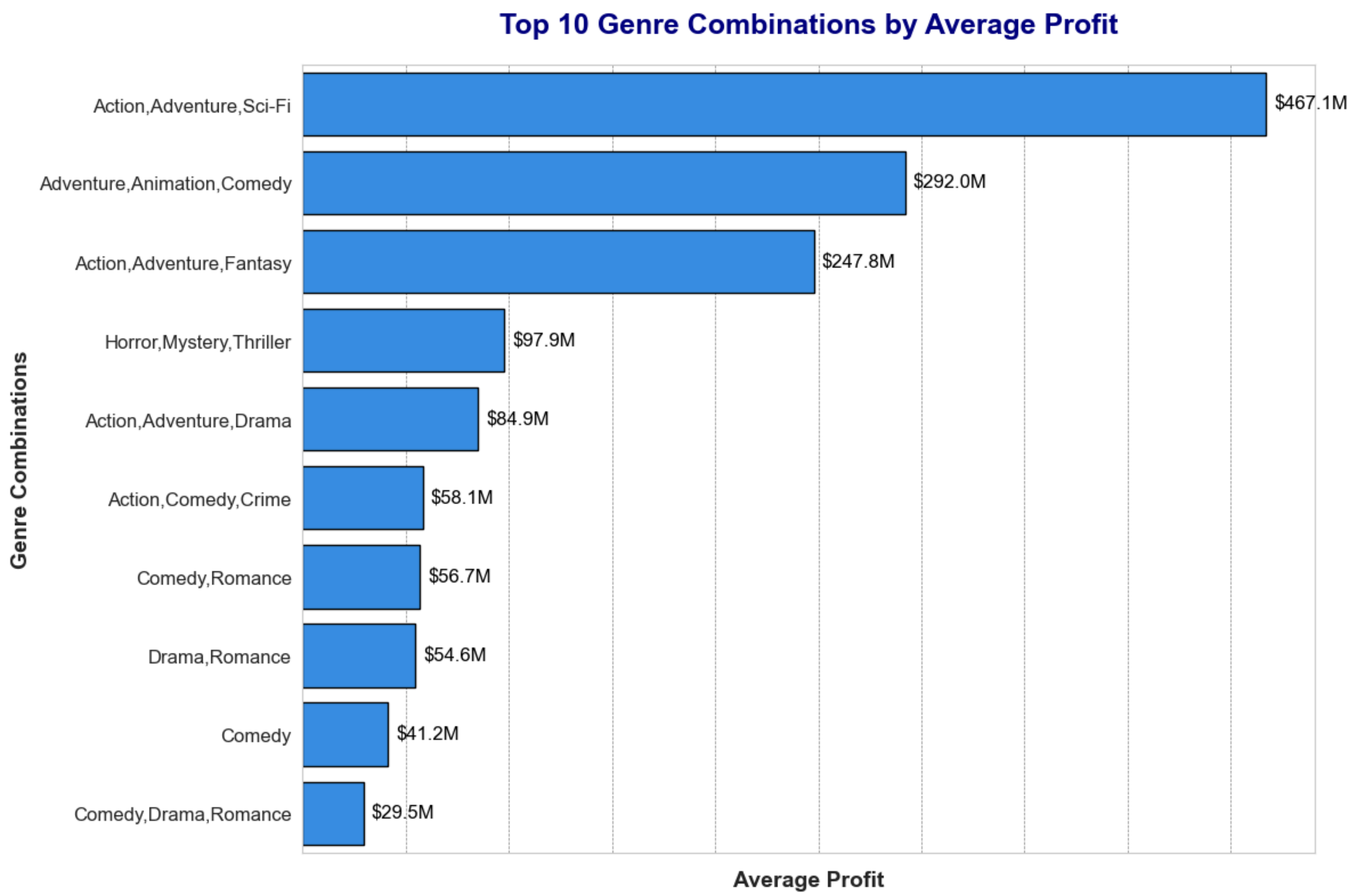


Now we can sort by average profit

```
# Sort by average profit to get the top profitable combinations
genre_combinations_analysis_filtered = genre_combinations_analysis_filtered.sort_values(by='avg_profit', ascending=False)

# Display the top 20 genre combinations based on average profit
genre_combinations_analysis_filtered.head(20)
```

	genres	count	avg_budget	avg_profit	avg_roi	avg_rating	total_votes
12	Action,Adventure,Sci-Fi	55	169716363.64	467118871.35	166.97	6.73	22181507
63	Adventure,Animation,Comedy	72	100090277.78	292000004.51	192.66	6.41	8178889
8	Action,Adventure,Fantasy	33	149109090.91	247778337.39	48.95	6.12	7794508
225	Horror,Mystery,Thriller	33	9721969.70	97892046.21	2614.19	5.53	2662668
6	Action,Adventure,Drama	30	88153333.33	84898672.47	-17.70	5.96	3590583
21	Action,Comedy,Crime	33	39930000.00	58148700.58	44.70	6.03	3794183
152	Comedy,Romance	44	25565909.09	56749376.34	119.30	5.85	3215413
203	Drama,Romance	35	22846857.14	54608913.46	270.25	6.82	3008082
117	Comedy	63	29061190.48	41244930.44	24.59	5.54	3906958
132	Comedy,Drama,Romance	58	18023965.52	29467109.05	122.75	6.38	4758146
124	Comedy,Drama	52	16168461.54	27646498.40	94.45	6.44	2985850
32	Action,Crime,Drama	39	27548717.95	25689928.59	2.77	6.35	3922818
172	Drama	61	11451672.13	14978908.18	113.12	6.70	2879926

```
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as ticker

# Function to format numbers as currency with millions
def format_currency_millions(value):
    return f"${value / 1_000_000:.1f}M"  # Format as millions with one decimal place

# Set modern and professional style
sns.set_style("whitegrid")
plt.figure(figsize=(12, 8))

# Sort by 'avg_profit' to get the top 10 genre combinations
top_10_genres_by_profit = genre_combinations_analysis_filtered.nlargest(10, 'avg_profit')

# Create the bar plot with dodgerblue color
barplot = sns.barplot(x='avg_profit', y='genres', data=top_10_genres_by_profit, color='dodgerblue', edgecolor='black')

# Customize title and labels for a professional look
barplot.set_title('Top 10 Genre Combinations by Average Profit', fontsize=18, weight='bold', color='navy', pad=20)
barplot.set_xlabel('Average Profit', fontsize=14, weight='bold')  # Proper x-axis title
barplot.set_ylabel('Genre Combinations', fontsize=14, weight='bold')

# Add grid for better readability
barplot.xaxis.grid(True, color='gray', linestyle='--', linewidth=0.5)

# Remove x-axis tick values but keep the grid
barplot.xaxis.set_major_locator(ticker.MaxNLocator(integer=True))  # Ensures grid lines are integer values
barplot.set_xticklabels([])  # Remove x-tick labels

# Customize tick parameters
barplot.tick_params(labelsize=12)

# Add value labels for each bar formatted as currency in millions, with adjusted positioning
for p in barplot.patches:
    barplot.annotate(format_currency_millions(p.get_x() + p.get_width()),  # Format the label
                     (p.get_x() + p.get_width(), p.get_y() + p.get_height() / 2),  # Position the label
                     ha='left', va='center', fontsize=12, color='black',
                     xytext=(5, 0), textcoords='offset points')  # Offset from the bar

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



Horror, Mystery, Thriller

Find the most successful producers

Find best producers for Horror, Mystery, or Thriller by the weighted score of (avg_roi * movie_count). We can include all movies tagged as either Horror, Mystery, or Thriller:

```
# Create a mask for the specified genres
mask_genres_hmt = merged_df['genres'] == 'Horror,Mystery,Thriller'
mask_separate_genre_hmt = merged_df['genre1'].isin(['Horror', 'Mystery', 'Thriller']) | \
                 merged_df['genre2'].isin(['Horror', 'Mystery', 'Thriller']) | \
                 merged_df['genre3'].isin(['Horror', 'Mystery', 'Thriller'])

# Combine the masks
hmt_all_df = merged_df[mask_genres_hmt | mask_separate_genre_hmt]
```

If we want to include only specific genre category 'Horror,Mystery,Thriller'

```
# Filter the DataFrame for the specified genres
hmt_df = merged_df[merged_df['genres'] == 'Horror,Mystery,Thriller']
```

```
# Include separate genres
# Melt the producers into a single column
producers = pd.melt(hmt_all_df, id_vars=['profit', 'roi'],
                    value_vars=['producer1', 'producer2'],
                    var_name='producer_type', value_name='producer_name')

# Remove rows with NaN producers
```

```
producers = producers.dropna(subset=['producer_name'])

# Group by producer name and calculate average profit and movie count
producer_analysis = producers.groupby('producer_name').agg(
    avg_profit=('profit', 'mean'),
    avg_roi=('roi', 'mean'),
    movie_count=('roi', 'count')
).reset_index()

# Calculate a weighted score
producer_analysis['total_profit'] = producer_analysis['avg_profit'] * producer_analysis['movie_count']

# Sort by score to find the best producers
best_producers = producer_analysis.sort_values(by='total_profit', ascending=False)

# Display the top producers
best_producers.head(10)
```

	producer_name	avg_profit	avg_roi	movie_count	total_profit
368	Neal H. Moritz	862008239.75	372.68	4	3448032959.00
211	Jason Blum	85844562.28	2538.74	36	3090404242.00
340	Michael Fottrell	939577505.00	399.13	3	2818732515.00
125	David Heyman	709565120.50	499.49	2	1419130241.00
109	Dana Brunetti	255373029.25	457.40	4	1021492117.00
334	Michael Bay	112803038.22	1141.24	9	1015227344.00
338	Michael De Luca	202912027.20	344.20	5	1014560136.00
208	James Wan	190578607.60	2169.43	5	952893038.00
44	Barbara Broccoli	910526981.00	355.26	1	910526981.00
119	David Barron	453404490.00	293.65	2	906808980.00

```
# Include only specific genre category 'Horror,Mystery,Thriller'

# Melt the producers into a single column
producers = pd.melt(hmt_df, id_vars=['profit', 'roi'],
                    value_vars=['producer1', 'producer2'],
                    var_name='producer_type', value_name='producer_name')

# Remove rows with NaN producers
producers = producers.dropna(subset=['producer_name'])

# Group by producer name and calculate average profit and movie count
producer_analysis = producers.groupby('producer_name').agg(
    avg_profit=('profit', 'mean'),
    avg_roi=('roi', 'mean'),
    movie_count=('roi', 'count')
).reset_index()

# Calculate a weighted score
producer_analysis['total_profit'] = producer_analysis['avg_profit'] * producer_analysis['movie_count']

# Sort by score to find the best producers
best_producers = producer_analysis.sort_values(by='total_profit', ascending=False)

# Display the top producers
best_producers.head(10)
```

	producer_name	avg_profit	avg_roi	movie_count	total_profit
8	Jason Blum	129855601.64	5012.65	14	1817978423.00
7	James Wan	202271632.00	2017.76	4	809086528.00
19	Peter Safran	270373892.50	2793.82	2	540747785.00
26	Sean McKittrick	242289130.50	2989.21	2	484578261.00
30	Tony DeRosa-Grund	298000141.00	1390.00	1	298000141.00
20	Rob Cowan	298000141.00	1390.00	1	298000141.00
28	Steven Schneider	202039844.00	3940.80	1	202039844.00
18	Oren Peli	156921515.00	3038.43	1	156921515.00
16	Michael Bay	98300632.00	1866.01	1	98300632.00
14	Marc Bienstock	93677816.00	1773.56	1	93677816.00

Find the most successful directors

Find best directors for Horror, Mystery, or Thriller by the weighted score of (avg_roi * movie_count).

Include separate genres

```
# Melt the directors into a single column
directors = pd.melt(hmt_all_df, id_vars=['profit', 'roi'],
                    value_vars=['director1', 'director2'],
                    var_name='director_type', value_name='director_name')

# Remove rows with NaN producers
directors = directors.dropna(subset=['director_name'])

# Group by director name and calculate average roi and movie count
director_analysis = directors.groupby('director_name').agg(
    avg_profit=('profit', 'mean'),
    avg_roi=('roi', 'mean'),
    movie_count=('roi', 'count')
).reset_index()

# Calculate a weighted score
director_analysis['total_profit'] = director_analysis['avg_profit'] * director_analysis['movie_count']

# Sort by score to find the best director
best_directors = director_analysis.sort_values(by='total_profit', ascending=False)

# Display the top directors
best_directors.head(10)
```

	director_name	avg_profit	avg_roi	movie_count	total_profit
150	James Wan	594548150.00	1675.92	3	1783644450.00
310	Sam Mendes	745073952.00	224.24	2	1490147904.00
188	Justin Lin	567231949.00	298.72	2	1134463898.00
111	F. Gary Gray	984846267.00	293.94	1	984846267.00
87	David Yates	835431568.00	568.35	1	835431568.00
53	Christopher Nolan	809439099.00	194.34	1	809439099.00
18	Andy Muschietti	397776767.50	1290.02	2	795553535.00
52	Christopher McQuarrie	383413644.00	202.34	2	766827288.00
304	Ruben Fleischer	737628605.00	535.89	1	737628605.00
253	Neil Boultby	623008101.00	256.00	1	623008101.00

Include only specific genre category 'Horror,Mystery,Thriller'

```
# Melt the producers into a single column
directors = pd.melt(hmt_df, id_vars=['profit', 'roi'],
```

```
                    value_vars=['director1', 'director2'],
                    var_name='director_type', value_name='director_name')

# Remove rows with NaN producers
directors = directors.dropna(subset=['director_name'])

# Group by director name and calculate average roi and movie count
director_analysis = directors.groupby('director_name').agg(
    avg_profit=('profit', 'mean'),
    avg_roi=('roi', 'mean'),
    movie_count=('roi', 'count')
).reset_index()

# Calculate a weighted score
director_analysis['total_profit'] = director_analysis['avg_profit'] * director_analysis['movie_count']

# Sort by score to find the best director
best_directors = director_analysis.sort_values(by='total_profit', ascending=False)

# Display the top directors
best_directors.head(10)
```

	director_name	avg_profit	avg_roi	movie_count	total_profit
18	Jordan Peele	242289130.50	2989.21	2	484578261.00
14	James Wan	227460828.00	2214.22	2	454921656.00
2	Ariel Schulman	169928918.00	3298.58	2	339857836.00
13	Henry Joost	169928918.00	3298.58	2	339857836.00
9	David F. Sandberg	290384865.00	1835.90	1	290384865.00
17	John R. Leonetti	250362920.00	3751.74	1	250362920.00
6	Christopher Landon	102957557.00	1959.15	2	205915114.00
0	Adam Robitel	157885588.00	1478.86	1	157885588.00
25	Scott Derrickson	71342212.50	1408.72	2	142684425.00
20	Leigh Whannell	110453155.00	1004.53	1	110453155.00

Find the most successful actresses

Find top 10 actresses based on score(total profit = avg_profit * movie_count).

Include separate genres

```
# Combine all actress columns into a single column for analysis
actresses_df = hmt_all_df.melt(id_vars=['profit', 'roi'], value_vars=['actress1', 'actress2', 'actress3'],
                                var_name='actress_rank', value_name='actress')

# Remove rows with missing actress values
actresses_df = actresses_df[actresses_df['actress'].notna()]

# Group by actress and calculate average profit, ROI, and total movies
best_actresses = actresses_df.groupby('actress').agg(
    avg_profit=('profit', 'mean'),
    avg_roi=('roi', 'mean'),
    movie_count=('actress', 'size')  # Count how many movies each actress appeared in
).reset_index()

# Calculate a weighted score
best_actresses['total_profit'] = best_actresses['avg_profit'] * best_actresses['movie_count']

# Sort by avg_profit to find the best actresses
best_actresses = best_actresses.sort_values(by='total_profit', ascending=False)

best_actresses.head(10)
```

	actress	avg_profit	avg_roi	movie_count	total_profit
341	Michelle Rodriguez	371506279.60	182.62	5	1857531398.00
239	Judi Dench	600724855.50	392.11	2	1201449711.00
363	Naomie Harris	910526981.00	355.26	1	910526981.00
161	Emma Watson	284635935.33	157.77	3	853907806.00
154	Eloise Mumford	423674360.00	851.34	2	847348720.00
124	Dakota Johnson	281517699.67	531.31	3	844553099.00
49	Anne Hathaway	397903064.00	19.90	2	795806128.00
344	Michelle Williams	737628605.00	535.89	1	737628605.00
453	Sophia Lillis	662457969.00	1792.74	1	662457969.00
364	Natalia Kaverznikova	623008101.00	256.00	1	623008101.00

```
# Find top 10 actresses based on score(total profit = avg_profit * movie_count)
# Don't Include separate genres

# Combine all actress columns into a single column for analysis
actresses_df = hmt_df.melt(id_vars=['profit', 'roi'], value_vars=['actress1', 'actress2', 'actress3'],
                                var_name='actress_rank', value_name='actress')

# Remove rows with missing actress values
actresses_df = actresses_df[actresses_df['actress'].notna()]

# Group by actress and calculate average profit, ROI, and total movies
best_actresses = actresses_df.groupby('actress').agg(
    avg_profit=('profit', 'mean'),
    avg_roi=('roi', 'mean'),
    movie_count=('actress', 'size')  # Count how many movies each actress appeared in
).reset_index()

# Calculate a weighted score
best_actresses['total_profit'] = best_actresses['avg_profit'] * best_actresses['movie_count']

# Sort by avg_profit to find the best actresses
best_actresses = best_actresses.sort_values(by='total_profit', ascending=False)

best_actresses.head(10)
```

	actress	avg_profit	avg_roi	movie_count	total_profit
35	Lin Shaye	157403551.50	2258.64	2	314807103.00
55	Vera Farmiga	298000141.00	1390.00	1	298000141.00
34	Lili Taylor	298000141.00	1390.00	1	298000141.00
50	Samara Lee	290384865.00	1835.90	1	290384865.00
38	Miranda Otto	290384865.00	1835.90	1	290384865.00
2	Allison Williams	250367951.00	4907.36	1	250367951.00
14	Catherine Keener	250367951.00	4907.36	1	250367951.00
0	Alfre Woodard	250362920.00	3751.74	1	250362920.00
5	Annabelle Wallis	250362920.00	3751.74	1	250362920.00
36	Lupita Nyong'o	234210310.00	1071.05	1	234210310.00

Find the most successful actors

Find top 10 actors based on score(total profit = avg_profit * movie_count).

Include separate genres

```
# Combine all actor columns into a single column for analysis
actors_df = hmt_all_df.melt(id_vars=['profit', 'roi'], value_vars=['actor1', 'actor2', 'actor3', 'actor4'],
                                var_name='actor_rank', value_name='actor')
```

```
# Remove rows with missing actor values
actors_df = actors_df[actors_df['actor'].notna()]

# Group by actress and calculate average profit, ROI, and total movies
best_actors = actors_df.groupby('actor').agg(
    avg_profit=('profit', 'mean'),
    avg_roi=('roi', 'mean'),
    movie_count=('actor', 'size')  # Count how many movies each actor appeared in
).reset_index()

# Calculate a weighted score
best_actors['total_profit'] = best_actors['avg_profit'] * best_actors['movie_count']

# Sort by avg_profit to find the best actor
best_actors = best_actors.sort_values(by='total_profit', ascending=False)

best_actors.head(10)
```

	actor	avg_profit	avg_roi	movie_count	total_profit
774	Vin Diesel	741613263.60	339.33	5	3708066318.00
214	Dwayne Johnson	611676037.83	286.68	6	3670056227.00
336	Jason Statham	282868960.91	88.57	11	3111558570.00
592	Paul Walker	821062230.67	398.92	3	2463186692.00
156	Daniel Craig	407791010.00	99.43	4	1631164040.00
748	Tom Hardy	535840171.67	306.03	3	1607520515.00
256	Gary Oldman	207737957.20	123.45	5	1038689786.00
339	Javier Bardem	323022463.67	127.02	3	969067391.00
161	Daniel Radcliffe	313503944.33	368.62	3	940511833.00
745	Tom Cruise	288924592.33	157.08	3	866773777.00

```
# Find top 10 actors based on score(total profit = avg_profit * movie_count)
# Don't Include separate genres

# Combine all actor columns into a single column for analysis
actors_df = hmt_df.melt(id_vars=['profit', 'roi'], value_vars=['actor1', 'actor2', 'actor3', 'actor4'],
                                var_name='actor_rank', value_name='actor')

# Remove rows with missing actor values
actors_df = actors_df[actors_df['actor'].notna()]

# Group by actress and calculate average profit, ROI, and total movies
best_actors = actors_df.groupby('actor').agg(
    avg_profit=('profit', 'mean'),
    avg_roi=('roi', 'mean'),
    movie_count=('actor', 'size')  # Count how many movies each actor appeared in
).reset_index()

# Calculate a weighted score
best_actors['total_profit'] = best_actors['avg_profit'] * best_actors['movie_count']

# Sort by avg_profit to find the best actor
best_actors = best_actors.sort_values(by='total_profit', ascending=False)

best_actors.head(10)
```

	actor	avg_profit	avg_roi	movie_count	total_profit
50	Patrick Wilson	227460828.00	2214.22	2	454921656.00
55	Ron Livingston	298000141.00	1390.00	1	298000141.00
2	Anthony LaPaglia	290384865.00	1835.90	1	290384865.00
5	Brad Greenquist	290384865.00	1835.90	1	290384865.00
1	Angus Sampson	134169371.50	1241.69	2	268338743.00
43	Leigh Whannell	134169371.50	1241.69	2	268338743.00
15	Daniel Kaluuya	250367951.00	4907.36	1	250367951.00
6	Bradley Whitford	250367951.00	4907.36	1	250367951.00
65	Ward Horton	250362920.00	3751.74	1	250362920.00
63	Tony Amendola	250362920.00	3751.74	1	250362920.00

Start coding or generate with AI.

Action, Adventure, Sci-Fi (maximize profit)

Find the most successful producers

Find best producers for Action, Adventure, or Sci-Fi by the weighted score of (avg_profit * movie_count).

Include either genre

```
aasf_all_df = merged_df[
    (merged_df['genre1'].isin(['Action', 'Adventure', 'Sci-Fi'])) |
    (merged_df['genre2'].isin(['Action', 'Adventure', 'Sci-Fi'])) |
    (merged_df['genre3'].isin(['Action', 'Adventure', 'Sci-Fi']))
]


# To include only specific genre category 'Action,Adventure,Sci-Fi'
# Filter the DataFrame for the specified genres
aasf_df = merged_df[merged_df['genres'] == 'Action,Adventure,Sci-Fi']


# Include separate genres

# Melt the producers into a single column
producers = pd.melt(aasf_all_df, id_vars=['profit'],
                    value_vars=['producer1', 'producer2'],
                    var_name='producer_type', value_name='producer_name')

# Remove rows with NaN producers
producers = producers.dropna(subset=['producer_name'])

# Group by producer name and calculate average profit and movie count
producer_analysis = producers.groupby('producer_name').agg(
    avg_profit=('profit', 'mean'),
    movie_count=('profit', 'count')
).reset_index()

# Calculate a weighted score
producer_analysis['total_profit'] = producer_analysis['avg_profit'] * producer_analysis['movie_count']

# Sort by score to find the best producers
best_producers = producer_analysis.sort_values(by='total_profit', ascending=False)

# Display the top producers
best_producers.head(10)
```