GET DATA ID FILES

```
import json
import pandas as pd
import requests
```

```
API_KEY = 'ae8f44a0db772e7e8f2d45b9fad24017'
```

```
#Genre Mapping Dictionary ; to be used later
GENRE_LIST_URL = f"https://api.themoviedb.org/3/genre/movie/list?api_key={API_KEY}&

def get_genre_mapping():
    response = requests.get(GENRE_LIST_URL)
    if response.status_code == 200:
        data = response.json()
        # Build a dictionary: key = genre id, value = genre name
        genre_mapping = { genre['id']: genre['name'] for genre in data.get('genres'
        return genre_mapping
    else:
        print(f"Error fetching genres: Status Code {response.status_code}")
        return {}

# Fetch the mapping
genre_dict = get_genre_mapping()
```

```
genre_dict
```

```
{28: 'Action',
 12: 'Adventure',
 16: 'Animation',
 35: 'Comedy',
 80: 'Crime',
 99: 'Documentary',
 18: 'Drama',
 10751: 'Family',
 14: 'Fantasy',
 36: 'History',
 27: 'Horror',
 10402: 'Music',
 9648: 'Mystery',
 10749: 'Romance',
 878: 'Science Fiction',
 10770: 'TV Movie',
 53: 'Thriller',
 10752: 'War',
 37: 'Western'}
```

```
# Extract all movies for Marvel Studios
START_YEAR = '2008-01-01'
END_YEAR = '2019-12-31'
company_id = 420

# Discover Movies helper function to get data for all MCU productions between 200
def discover_movies(company_id):
    page = 1
    movies = []
    while True:
        url = (
            f'https://api.themoviedb.org/3/discover/movie?api_key={API_KEY}'
            f'&with_companies={company_id}'
            f'&primary_release_date.gte={START_YEAR}'
            f'&primary_release_date.lte={END_YEAR}'
            f'&sort_by=release_date.desc'
            f'&page={page}'
        )
        response = requests.get(url)
        data = response.json()
        movies.extend(data['results'])

        if page >= data['total_pages']:
            break
        page += 1
    return movies
```

```
if company_id:
    movies = discover_movies(company_id)
else:
    print("Production company not found.")
```

```
all_movie_df = pd.DataFrame(movies)
all_movie_df = all_movie_df[['poster_path','genre_ids','id','title','original_lan
```

```
#Code to filter out only MCU movies
import itertools

def get_frequency_of_values_in_lists(data):
    """
    Counts the frequency of values in a list of lists.

    Args:
        data: A list of lists.

    Returns:
        A dictionary where keys are values from the lists and values are their co
    """

    flattened_list = list(itertools.chain.from_iterable(data))
    frequency_counts = {}

    for item in flattened_list:
        frequency_counts[item] = frequency_counts.get(item, 0) + 1

    return frequency_counts
```

```
pd.DataFrame({genre_dict[k]:{"id":k,"count":v} for k,v in get_frequency_of_values_
```

	Documentary	Science Fiction	Action	Adventure	Comedy	Fantasy	Drama	TV Movie	H
id	99	878	28	12	35	14	18	10770	
count	3	31	33	29	5	13	1	1	

```
# Define genres to exclude : Discard all documentaries (99),Animation(16), Crime(
excluded_genres = {99, 16, 80, 36}

#And also filter out movies which are one-shot or tie-ins
exclusion_string = "marvel one|peter's|holiday special|team"

movie_df = all_movie_df[all_movie_df['genre_ids'].apply(lambda x: not any(genre i
```

```
movie_df
```

	poster_path	genre_ids	id	title	original_l

2	/4q2NNj4S5dG2RLF9CpXsej7yXl.jpg	[28, 12, 878]	429617	Spider-Man: Far From Home
3	/ulzhLuWrPK07P1YkdWQLZnQh1JL.jpg	[12, 878, 28]	299534	Avengers: Endgame
4	/AtsgWhDnHTq68L0lLsUrCnM7TjG.jpg	[28, 12, 878]	299537	Captain Marvel
5	/cFQEO687n1K6umXbInzocxcnAQz.jpg	[28, 12, 878]	363088	Ant-Man and the Wasp
6	/7WsyChQLEftFiDOVTGkv3hFpyyt.jpg	[12, 28, 878]	299536	Avengers: Infinity War
8	/uxzzxijgPIY7slzFvMotPv8wjKA.jpg	[28, 12, 878]	284054	Black Panther
9	/rzRwTcFvttcN1ZpX2xv4j3tSdJu.jpg	[28, 12, 878]	284053	Thor: Ragnarok
10	/c24sv2weTHPsmDa7jEMN0m2P3RT.jpg	[28, 12, 878, 18]	315635	Spider-Man: Homecoming

11	/y4MBh0EjBlMuOzv9axM4qJlmhzz.jpg	[878, 12, 28]	283995	Guardians of the Galaxy Vol. 2
13	/uGBVj3bEbCoZbDjjl9wTxcygko1.jpg	[28, 12, 14]	284052	Doctor Strange
15	/rAGiXaUfPzY7CDEyNKUofk3Kw2e.jpg	[12, 28, 878]	271110	Captain America: Civil War
17	/rQRnQfUl3kfp78nCWq8Ks04vnq1.jpg	[878, 28, 12]	102899	Ant-Man
18	/4ssDuvEDkSArWEdyBl2X5EHvYKU.jpg	[28, 12, 878]	99861	Avengers: Age of Ultron
19	/jPrJPZKJVhvyJ4DmUTrDgmFN0yG.jpg	[28, 878, 12]	118340	Guardians of the Galaxy

```
movie_df['id'].unique()
```

```
array([429617, 299534, 299537, 363088, 299536, 284054, 284053, 315635,
       283995, 284052, 271110, 102899,  99861, 118340, 100402,  76338,
        68721,  24428,   1771,  10195,  10138,   1724,   1726])
```

```
import time
from tqdm import tqdm
```

```
#Movie Cast API
def get_movie_cast(movie_ids):
    base_url = 'https://api.themoviedb.org/3/movie/'
    cast_data = {}

    for movie_id in tqdm(movie_ids):
        url = f"{base_url}{movie_id}/credits?api_key={API_KEY}"
        response = requests.get(url)

        if response.status_code == 200:
            data = response.json()
            cast_list = data.get('cast', [])
            actor_info = [(actor['name'], actor['id'],actor['popularity'],actor['
            cast_data[movie_id] = actor_info
        else:
            print(f"❌ Failed to fetch cast for movie ID {movie_id} (Status {resp
            cast_data[movie_id] = []
        time.sleep(1)

    return cast_data, response
```

```
cast_data,response = get_movie_cast(movie_df['id'])
```

```
100%|██████████| 23/23 [00:26<00:00,  1.15s/it]
```

Get Movie cast list for each and every movie and keep only unique actors

```
movie_cast_list = pd.DataFrame.from_dict(cast_data,orient='index').values.tolist(
```

```
unique_actors = set([j for i in movie_cast_list for j in i])
```

```
actors = pd.DataFrame(unique_actors,columns=['name','id','popularity','character'
actors.dropna(axis=0,inplace=True)
actors['id'] = actors['id'].astype(int)
```

```
actors['movie_id'] = actors['movie_id'].astype("category")
actors['movie_id'] = actors['movie_id'].cat.set_categories(movie_df['id'])
actors.sort_values(["movie_id"],ascending=False,inplace=True)
```

```
actors.drop_duplicates(['name'],keep='first',inplace=True)
```

```
lead_filter = actors['popularity'].mean()
```

```
lead_filter
```

```
np.float64(2.910201393728223)
```

Get lead actors vs non lead actors dataframe : Used for scalability in further applications

```
#Number of lead actors
lead_actors_df = actors[actors['popularity']>=lead_filter]
```

```
#Number of not so lead actors
actors[actors['popularity']<lead_filter]
```

	name	id	popularity	character	movie_id
650	Shaun Toub	17857	1.9773	Yinsen	1726
655	Joshua Harto	34544	1.4475	CAOC Analyst	1726
699	Kevin Foster	95698	1.0430	Jimmy	1726
506	Bill Smitrovich	17200	2.1970	General Gabriel	1726
706	Micah A. Hauptman	150669	1.0197	CAOC Analyst	1726
...	...	...	...	...	...
118	JB Smoove	65920	2.4921	Mr. Dell	429617
127	Anjana Vasan	1503076	1.1538	Queens Reporter	429617
520	Meagan Holder	1378683	1.3179	Pretty Tourist (uncredited)	429617
221	Massi Furlan	1010873	1.1199	Flight Attendant (uncredited)	429617
12	Claire Rushbrook	68384	1.5047	Janice	429617

400 rows × 5 columns

```
lead_actors_df
```

	name	id	popularity	character	movie_id
93	Gwyneth Paltrow	12052	4.1431	Pepper Potts	1726
525	Robert Downey Jr.	3223	11.2703	Tony Stark	1726
513	Terrence Howard	18288	4.6701	Rhodey	1726
82	Jeff Bridges	1229	4.6122	Obadiah Stane	1726
485	Samuel L. Jackson	2231	10.9680	Nick Fury (uncredited)	1726
...	...	...	...	...	...
793	Ken Jeong	83586	4.8592	Security Guard	299534
775	James D'Arcy	19655	3.1687	Jarvis	299534
63	Hiroyuki Sanada	9195	4.9882	Akihiko	299534
621	Jake Gyllenhaal	131	8.2200	Quentin Beck / Mysterio	429617
568	J.K. Simmons	18999	6.9058	J. Jonah Jameson	429617

174 rows × 5 columns

```
import time
from tqdm import tqdm
```

Get revenue, review data for each actor

```
def get_actor_movies_with_money_info(actor_name, actor_id, start_year=2003, end_y
    print(f"For {actor_name}")
    base_url = "https://api.themoviedb.org/3"
    credits_url = f"{base_url}/person/{actor_id}/movie_credits?api_key={API_KEY}"

    response = requests.get(credits_url)
    if response.status_code != 200:
        print(f"Failed to fetch credits for actor {actor_id}")
        return []

    movies = response.json().get("cast", [])
```

```
    selected_movies = []
    num_req = 0

    for movie in tqdm(movies):
        movie_id = movie.get("id")
        movie_name = movie.get("title")
        release_date = movie.get("release_date", "")

        # Skip if no valid release date
        try:
            year = int(release_date[:4])
        except:
            continue

        if not (start_year <= year <= end_year):
            continue

        # Fetch movie details
        movie_details_url = f"{base_url}/movie/{movie_id}?api_key={API_KEY}"
        details_response = requests.get(movie_details_url)
        if details_response.status_code != 200:
            continue

        details = details_response.json()
        budget = details.get("budget", 0)
        revenue = details.get("revenue", 0)
        runtime = details.get("runtime", 0)
        release_date = details.get("release_date", "")
        genres = details.get("genres", [])

        # Fetch cast order
        credits_url = f"{base_url}/movie/{movie_id}/credits?api_key={API_KEY}"
        credits_response = requests.get(credits_url)
        if credits_response.status_code != 200:
            cast_order = None
        else:
            cast_data = credits_response.json().get("cast", [])
            cast_order = next((c['order'] for c in cast_data if c['id'] == actor_

        selected_movies.append(
            (movie_id, movie_name, budget, revenue, runtime, release_date, genres
        )

        num_req += 2  # 2 requests per movie (details + credits)
        if num_req >= 48:
```

```
            time.sleep(1)
            num_req = 0

    return selected_movies
```

```
#Treatment group specified : for this causal analysis
main_actors_str = '''Robert Downey Jr.
Chris Evans
Scarlett Johansson
Mark Ruffalo
Jeremy Renner
Tom Holland
Benedict Cumberbatch
Chadwick Boseman
Paul Rudd
Tom Hiddleston
Chris Pratt
Chris Hemsworth'''
```

```
main_actors_str.split("\n")
```

```
['Robert Downey Jr.',
 'Chris Evans',
 'Scarlett Johansson',
 'Mark Ruffalo',
 'Jeremy Renner',
 'Tom Holland',
 'Benedict Cumberbatch',
 'Chadwick Boseman',
 'Paul Rudd',
 'Tom Hiddleston',
 'Chris Pratt',
 'Chris Hemsworth']
```

```
avengers = actors[actors['name'].isin(main_actors_str.split("\n"))]
```

```
import numpy as np
```

```
#For each actor get their revenue, ratings, reviews, genres for all movies
for actor_name,actor_id in zip(avengers['name'],avengers['id']):
  acting_list = get_actor_movies_with_money_info(actor_name,actor_id)

  acted_df = pd.DataFrame(acting_list,columns=['movie_id','title','budget','reven
  acted_df['genres'] = acted_df['genres'].apply(lambda x: [i['name'] for i in x i

  acted_df['genres'] = acted_df['genres'].apply(lambda x: np.nan if x == [] else
  acted_df.dropna(how='any', axis=0,inplace=True)
  acted_df['actor_name'] = actor_name
  acted_df[acted_df['revenue']>0].to_json(f"avenger_{actor_id}.json")
```

```
For Robert Downey Jr.
100%|██████████| 127/127 [00:11<00:00, 10.67it/s]
For Scarlett Johansson
100%|██████████| 101/101 [00:22<00:00,  4.57it/s]
For Jeremy Renner
100%|██████████| 56/56 [00:12<00:00,  4.47it/s]
For Tom Hiddleston
100%|██████████| 60/60 [00:15<00:00,  3.86it/s]
For Chris Hemsworth
100%|██████████| 59/59 [00:12<00:00,  4.58it/s]
For Chris Evans
100%|██████████| 66/66 [00:15<00:00,  4.35it/s]
For Mark Ruffalo
100%|██████████| 89/89 [00:17<00:00,  5.17it/s]
For Chris Pratt
100%|██████████| 65/65 [00:16<00:00,  3.84it/s]
For Paul Rudd
100%|██████████| 112/112 [00:26<00:00,  4.23it/s]
For Chadwick Boseman
100%|██████████| 31/31 [00:08<00:00,  3.70it/s]
For Tom Holland
100%|██████████| 36/36 [00:07<00:00,  4.89it/s]
For Benedict Cumberbatch
100%|██████████| 109/109 [00:27<00:00,  4.02it/s]
```

```
import numpy as np
import glob
```

Export data for treatment and control group

```
all_data = []
avenger_df = pd.DataFrame()
for file_path in glob.glob('avenger_*.json'):
  try:
    with open(file_path, 'r') as f:
      data = json.load(f)
      avenger_df=pd.concat([avenger_df,pd.DataFrame(data)],ignore_index=True)
  except FileNotFoundError:
    print(f"Error: File not found: {file_path}")
  except json.JSONDecodeError:
    print(f"Error: Invalid JSON format in file: {file_path}")
```

```
avenger_df
```

	movie_id	title	budget	revenue	runtime	release_date	genres
0	4964	Knocked Up	30000000	219900000	129	2007-06-01	[Comedy, Romance, Drama]
1	22958	The Shape of Things	0	735992	96	2003-07-24	[Comedy, Drama, Romance]
2	6575	Walk Hard: The Dewey Cox Story	35000000	18317151	96	2007-12-21	[Comedy, Music]
3	6957	The 40 Year Old Virgin	26000000	177400000	116	2005-08-11	[Comedy, Romance]
4	8699	Anchorman: The Legend of Ron Burgundy	26000000	90574188	95	2004-06-28	[Comedy]
...	...	...	...	...	...	...	...

```
OMDB_API_KEY = "4280db16"
```

```
def get_rotten_tomatoes_rating(movie_title):
    url = f"http://www.omdbapi.com/?t={movie_title}&apikey={OMDB_API_KEY}"
    response = requests.get(url)
    if response.status_code != 200:
        return None
    ratings = response.json().get("Ratings", [])
    imdb_id = response.json().get("imdbID", [])
    imdb_votes = response.json().get("imdbVotes", [])
    try:
      return imdb_votes,imdb_id, [{i['Source']:i["Value"]} for i in ratings]
    # for rating in ratings:
    #     if rating["Source"] == "Rotten Tomatoes":
    #         return rating["Value"]
    except Exception as e:
      print(e)
      return None,None, None
```

```
get_rotten_tomatoes_rating("Anchorman: The Legend of Ron Burgundy")
```

```
('388,982',
 'tt0357413',
 [{'Internet Movie Database': '7.1/10'},
  {'Rotten Tomatoes': '66%'},
  {'Metacritic': '63/100'}])
```

```
tqdm.pandas()
```

```
avenger_df['imdb_votes'],avenger_df['imdb_id'],avenger_df['movie_ratings'] = zip(
```

```
100%|██████████| 356/356 [00:41<00:00,  8.49it/s]
```

```
avenger_df['movie_ratings'][0]
```

```
[{'Internet Movie Database': '6.9/10'},
 {'Rotten Tomatoes': '90%'},
 {'Metacritic': '85/100'}]
```

```
# Function to flatten ratings
def extract_ratings(rating_list):
    rating_dict = {}
    for item in rating_list:
        rating_dict.update(item)
    return rating_dict

# Convert ratings to proper numeric formats
def convert_imdb(val):
    try:
        return float(val.split('/')[0])
    except:
        return None

def convert_rt(val):
    try:
        return int(val.strip('%'))
    except:
        return None

def convert_mc(val):
    try:
        return int(val.split('/')[0])
    except:
        return None
```

```
# Apply and expand to new columns
ratings_expanded = avenger_df['movie_ratings'].apply(extract_ratings).apply(pd.Se

ratings_expanded['Internet Movie Database'] = ratings_expanded['Internet Movie Da
ratings_expanded['Rotten Tomatoes'] = ratings_expanded['Rotten Tomatoes'].apply(c
ratings_expanded['Metacritic'] = ratings_expanded['Metacritic'].apply(convert_mc)

# Merge into original dataframe
avenger_df_processed = avenger_df.drop(columns=['movie_ratings']).join(ratings_ex
```

```
avenger_df_processed.head()
```

	movie_id	title	budget	revenue	runtime	release_date	genres	cas
0	4964	Knocked Up	30000000	219900000	129	2007-06-01	[Comedy, Romance, Drama]	
1	22958	The Shape of Things	0	735992	96	2003-07-24	[Comedy, Drama, Romance]	
2	6575	Walk Hard: The Dewey Cox Story	35000000	18317151	96	2007-12-21	[Comedy, Music]	
3	6957	The 40 Year Old Virgin	26000000	177400000	116	2005-08-11	[Comedy, Romance]	
4	8699	Anchorman: The Legend	26000000	90574188	95	2004-06-28	[Comedy]	

```
avenger_df_processed.groupby('actor_name').agg({"budget":"mean","revenue":"mean",
```

actor_name	budget	revenue	runtime	Internet Movie Database	Rotten Tomatoes	Metacritic
Benedict Cumberbatch	9.930938e+07	4.776762e+08	123.562500	7.253125	72.387097	65.200000
Chadwick Boseman	1.196667e+08	6.445723e+08	130.416667	7.166667	70.583333	61.333333
Chris Evans	1.113568e+08	4.548289e+08	116.324324	6.889189	62.200000	56.513514
Chris Hemsworth	1.309259e+08	4.978783e+08	122.444444	6.911111	67.115385	59.769231
Chris Pratt	1.119344e+08	4.984149e+08	119.062500	6.693750	61.218750	56.548387
Jeremy Renner	9.038793e+07	3.981202e+08	121.185185	7.011111	70.888889	63.555556
Mark Ruffalo	7.555641e+07	3.525558e+08	119.205128	6.984615	69.810811	62.615385
Paul Rudd	7.059526e+07	2.344231e+08	108.263158	6.505263	64.027778	59.694444
Robert Downey Jr	1.042879e+08	4.733456e+08	121.515152	6.966667	66.424242	61.696970

```
from bs4 import BeautifulSoup
```

```
HEADERS = {
    "User-Agent": "Mozilla/5.0 (compatible; AcademicBot/1.0; +http://youruniversi
}

def get_bom_opening_weekend(imdb_id):
    url = f"https://www.boxofficemojo.com/title/{imdb_id}/?ref_=bo_hm_rd"
    response = requests.get(url, headers=HEADERS)
    html_test= response.text

    soup = BeautifulSoup(html_test, 'html.parser')

    # Find all divs in case structure varies
    sections = soup.find_all('div', class_='a-section a-spacing-none')

    opening_weekend = None

    for section in sections:
        labels = section.find_all('span')
        if labels and 'Domestic Opening' in labels[0].text:
            money_tag = section.find('span', class_='money')
            if money_tag:
                opening_weekend = money_tag.text.strip()
                break

    return opening_weekend

def get_opening_weekend_bom(imdb_id, title, year=None):
    if not imdb_id:
        print(f"❌ IMDb ID not found for {title}")
        return "$0"

    opening = get_bom_opening_weekend(imdb_id)
    return opening
```

```
avenger_df_processed['opening_weekend'] = avenger_df_processed.progress_apply(lam
```

```
 90%|▌▌▌▌▌▌▌▌▌ | 322/356 [03:22<00:25,  1.36it/s]❌ IMDb ID not found for Tin
100%|▌▌▌▌▌▌▌▌▌▌| 356/356 [03:43<00:00,  1.59it/s]
```

```
def parse_money(money_str):
    if not money_str:
        return None
    return int(money_str.replace('$', '').replace(',', ''))

avenger_df_processed['opening_weekend'] = avenger_df_processed['opening_weekend']
```

```
avenger_df_processed.head()
```

	movie_id	title	budget	revenue	runtime	release_date	genres	cas
0	4964	Knocked Up	30000000	219900000	129	2007-06-01	[Comedy, Romance, Drama]	
1	22958	The Shape of Things	0	735992	96	2003-07-24	[Comedy, Drama, Romance]	
2	6575	Walk Hard: The Dewey Cox Story	35000000	18317151	96	2007-12-21	[Comedy, Music]	
3	6957	The 40 Year Old Virgin	26000000	177400000	116	2005-08-11	[Comedy, Romance]	
4	8699	Anchorman: The Legend of Ron Burgundy	26000000	90574188	95	2004-06-28	[Comedy]	

```
avenger_df_processed1 = avenger_df_processed.copy()
```

```
from sklearn.preprocessing import MultiLabelBinarizer

mlb = MultiLabelBinarizer()
one_hot = pd.DataFrame(mlb.fit_transform(avenger_df_processed1['genres']), column

# Merge with original DataFrame
avenger_df_processed1 = pd.concat([avenger_df_processed1.drop(columns=['genres'])
```

```
avenger_df_sorted = avenger_df_processed1.sort_values(['release_date'],ascending=
```

```
avenger_df_sorted['MCU'] = avenger_df_sorted.apply(lambda x: 1 if x['movie_id'] i
```

```
def keep_first_one_only(df, group_col='actor_name', flag_col='MCU'):
    # Create a column tracking cumulative sum of 1s per group
    df['_cumsum'] = df.groupby(group_col)[flag_col].cumsum()

    # Set flag to 0 if it's a 1 and it's not the first one
    df["MCU Entry"] = df.apply(lambda row: 1 if row[flag_col] == 1 and row['_cums

    # Drop helper column
    df.drop(columns=['_cumsum'], inplace=True)
    return df

avenger_df_sorted = keep_first_one_only(avenger_df_sorted)
```

```
avenger_df_sorted.to_json("treatment_group_data_raw.json")
```

```
#We need to export data for the control group as well

non_avengers = {
    "Ben Foster":11107,
    "Channing Tatum":38673,
    "Charlize Theron":6885,
    "Edward Norton":819,
    "Jim Carrey":206,
    "John David Washington":1117313,
    "Matthew Goode":1247,
    "Ryan Reynolds":10859,
    "Sam Worthington":65731,
    "Steve Carell":4495,
    "Taron Egerton":1303037,
```

```
    "Tye Sheridan":1034681
}


for actor_name,actor_id in non_avengers.items():
  acting_list = get_actor_movies_with_money_info(actor_name,actor_id)

  acted_df = pd.DataFrame(acting_list,columns=['movie_id','title','budget','reven
  acted_df['genres'] = acted_df['genres'].apply(lambda x: [i['name'] for i in x i

  acted_df['genres'] = acted_df['genres'].apply(lambda x: np.nan if x == [] else
  acted_df.dropna(how='any', axis=0,inplace=True)
  acted_df['actor_name'] = actor_name
  acted_df[acted_df['revenue']>0].to_json(f"non_avenger_{actor_id}.json")
```

```
For Ben Foster
100%|██████████| 58/58 [00:13<00:00,  4.34it/s]
For Channing Tatum
100%|██████████| 65/65 [00:16<00:00,  4.03it/s]
For Charlize Theron
100%|██████████| 72/72 [00:13<00:00,  5.38it/s]
For Edward Norton
100%|██████████| 72/72 [00:12<00:00,  5.61it/s]
For Jim Carrey
100%|██████████| 87/87 [00:12<00:00,  6.73it/s]
For John David Washington
100%|██████████| 17/17 [00:04<00:00,  3.90it/s]
For Matthew Goode
100%|██████████| 37/37 [00:10<00:00,  3.54it/s]
For Ryan Reynolds
100%|██████████| 95/95 [00:21<00:00,  4.52it/s]
For Sam Worthington
100%|██████████| 60/60 [00:13<00:00,  4.40it/s]
For Steve Carell
100%|██████████| 72/72 [00:16<00:00,  4.36it/s]
For Taron Egerton
100%|██████████| 23/23 [00:04<00:00,  4.62it/s]
For Tye Sheridan
100%|██████████| 30/30 [00:08<00:00,  3.44it/s]
```

```
all_data = []
non_avenger_df = pd.DataFrame()
for file_path in glob.glob('non_avenger_*.json'):
  try:
    with open(file_path, 'r') as f:
      data = json.load(f)
      non_avenger_df=pd.concat([non_avenger_df,pd.DataFrame(data)],ignore_index=T
  except FileNotFoundError:
    print(f"Error: File not found: {file_path}")
  except json.JSONDecodeError:
    print(f"Error: Invalid JSON format in file: {file_path}")
```

```
non_avenger_df['imdb_votes'],non_avenger_df['imdb_id'],non_avenger_df['movie_rati
```

```
100%|██████████| 314/314 [00:45<00:00,  6.91it/s]
```

```
# Apply and expand to new columns
ratings_expanded = non_avenger_df['movie_ratings'].apply(extract_ratings).apply(p

ratings_expanded['Internet Movie Database'] = ratings_expanded['Internet Movie Da
ratings_expanded['Rotten Tomatoes'] = ratings_expanded['Rotten Tomatoes'].apply(c
ratings_expanded['Metacritic'] = ratings_expanded['Metacritic'].apply(convert_mc)

# Merge into original dataframe
non_avenger_df_processed = non_avenger_df.drop(columns=['movie_ratings']).join(ra
```

```
non_avenger_df_processed.groupby('actor_name').agg({"budget":"mean","revenue":"me
```

actor_name	budget	revenue	runtime	Internet Movie Database	Rotten Tomatoes	Metacritic
Ben Foster	3.613237e+07	7.365474e+07	109.444444	6.651852	57.458333	55.360000
Channing Tatum	5.460000e+07	1.681020e+08	111.687500	6.441667	59.812500	58.446809
Charlize Theron	7.610345e+07	2.309435e+08	111.724138	6.558621	58.068966	57.655172
Edward Norton	5.004681e+07	1.070255e+08	113.200000	6.916667	65.040000	63.916667
Jim Carrey	6.676213e+07	1.970498e+08	103.000000	6.372222	50.500000	52.352941
John David Washington	7.916000e+07	1.211021e+08	129.600000	7.020000	66.250000	66.000000
Matthew Goode	2.466250e+07	4.925628e+07	112.833333	6.781818	60.650000	57.789474
Ryan Reynolds	6.335327e+07	2.017278e+08	107.681818	6.609091	50.604651	48.238095

```
non_avenger_df_processed['opening_weekend'] = non_avenger_df_processed.progress_a
```

```
  2%|          | 5/314 [00:02<02:59,  1.72it/s]❌ IMDb ID not found for Lemon
 82%|▉▉▉▉▉▉▉▉  | 258/314 [02:46<00:29,  1.88it/s]❌ IMDb ID not found for Sou
100%|▉▉▉▉▉▉▉▉▉▉| 314/314 [03:15<00:00,  1.60it/s]
```

```
non_avenger_df_processed['opening_weekend'] = non_avenger_df_processed['opening_w
```

```
non_avenger_df_processed1 = non_avenger_df_processed.copy()
```

```
mlb1 = MultiLabelBinarizer()
one_hot1 = pd.DataFrame(mlb1.fit_transform(non_avenger_df_processed1['genres']),

# Merge with original DataFrame
non_avenger_df_processed1 = pd.concat([non_avenger_df_processed1.drop(columns=['g
```

```
non_avenger_df_sorted = non_avenger_df_processed1.sort_values(['release_date'],as
```

```
non_avenger_df_sorted['MCU'] = non_avenger_df_sorted.apply(lambda x: 0 if x['movi
```

```
non_avenger_df_sorted = keep_first_one_only(non_avenger_df_sorted)
```

```
non_avenger_df_sorted.to_json("control_group_data_raw.json")
```