Import Libraries and Load data

```
import pandas as pd
import numpy as np
```

```
treatment = pd.read_json("/content/treatment_group_data_raw.json")
control = pd.read_json("/content/control_group_data_raw.json")
```

Preprocessing and Feature Engineering

```
treatment['release_date'] = pd.to_datetime(treatment['release_date'])
treatment['release_year'] = treatment['release_date'].dt.year
treatment['release_quarter'] = treatment['release_date'].dt.quarter

control['release_date'] = pd.to_datetime(control['release_date'])
control['release_year'] = control['release_date'].dt.year
control['release_quarter'] = control['release_date'].dt.quarter
```

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

```
treatment['MCU'] = treatment.apply(lambda x: 1 if x['movie_id'] in mcu_movie_ids el
```

```
np.int64(76)
```

```
treatment.drop(['movie_id', 'imdb_id'], axis=1,inplace=True)
control.drop(['movie_id', 'imdb_id'], axis=1,inplace=True)
```

Constructing Treatment and Control Groups

```
treatment['Treatment'] = 1
control['Treatment'] = 0
```

```
treatment_control = pd.concat([treatment,control],axis=0)
```

```
from tqdm import tqdm
tqdm.pandas()
```

```
treatment_control['Lead Role'] = treatment_control.progress_apply(lambda x: 1 if
treatment_control['Supporting Role'] = treatment_control.progress_apply(lambda x:
```

```
100%|██████████| 670/670 [00:00<00:00, 66293.55it/s]
100%|██████████| 670/670 [00:00<00:00, 91426.74it/s]
```

```
treatment_control
```

	title	budget	revenue	runtime	release_date	cast_order	actor_nam
272	My Life Without Me	2500000	12300000	106	2003-03-07	3	Mark Ruffa
274	View from the Top	30000000	19526014	87	2003-03-21	2	Mark Ruffa
1	The Shape of Things	0	735992	96	2003-07-24	0	Paul Ruc
335	S.W.A.T.	80000000	207700000	117	2003-08-08	5	Jeren Renn
169	Lost in Translation	4000000	119723856	102	2003-09-18	1	Scarle Johanss
...	...	...	...	...	...	...	
137	Blink Twice	20000000	46393906	102	2024-08-21	1	Channi Tatu
308	The Killer	30000000	318618	126	2024-08-22	2	Sa Worthingto
270	The Order	20000000	1970445	116	2024-12-05	2	Tye Sherida
44	A Complete Unknown	65000000	138003641	140	2024-12-18	1	Edwa Norto
12	Sonic the Hedgehog 3	122000000	486018457	110	2024-12-19	0	Jim Carr

670 rows × 33 columns

Identifying MCU Entry and Actor Treatment Timing

```
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

treatment_control = keep_first_one_only(treatment_control)
```

```
treatment_control['MCU Entry Year'] = treatment_control.apply(lambda x: 0 if not
```

```
treatment_control['actor_name'].unique()
```

```
array(['Mark Ruffalo', 'Paul Rudd', 'Jeremy Renner', 'Scarlett Johansson',
       'Robert Downey Jr.', 'Chris Evans', 'Chris Pratt',
       'Benedict Cumberbatch', 'Chadwick Boseman', 'Chris Hemsworth',
       'Tom Hiddleston', 'Tom Holland', 'Matthew Goode', 'Ben Foster',
       'Jim Carrey', 'Ryan Reynolds', 'Steve Carell', 'Charlize Theron',
       'Edward Norton', 'Sam Worthington', 'Channing Tatum',
       'Tye Sheridan', 'Taron Egerton', 'John David Washington'],
      dtype=object)
```

```
treatment_control[treatment_control['MCU Entry']==1]
```

	title	budget	revenue	runtime	release_date	cast_order	actor_na
215	Iron Man	140000000	585174222	126	2008-04-30	0	Rob Downey
172	Iron Man 2	200000000	623933331	124	2010-04-28	3	Scarl Johanss
311	Thor	150000000	449326618	115	2011-04-21	2	To Hiddlest
284	Thor	150000000	449326618	115	2011-04-21	0	Ch Hemswo
330	Thor	150000000	449326618	115	2011-04-21	50	Jerer Renr
91	Captain America: The First Avenger	140000000	370569774	124	2011-07-22	0	Chris Eva
279	The Avengers	220000000	1518815515	143	2012-04-25	2	Mark Ruffa
68	Guardians of the Galaxy	170000000	772776600	121	2014-07-30	0	Chris Pr
30	Ant-Man	130000000	519311965	117	2015-07-14	0	Paul Ru
75	Captain America: Civil War	250000000	1155046416	147	2016-04-27	12	Tom Holla
128	Captain America: Civil War	250000000	1155046416	147	2016-04-27	7	Chadw Bosem
166	Doctor Strange	180000000	676343174	115	2016-10-25	0	Benec Cumberbat

12 rows × 34 columns

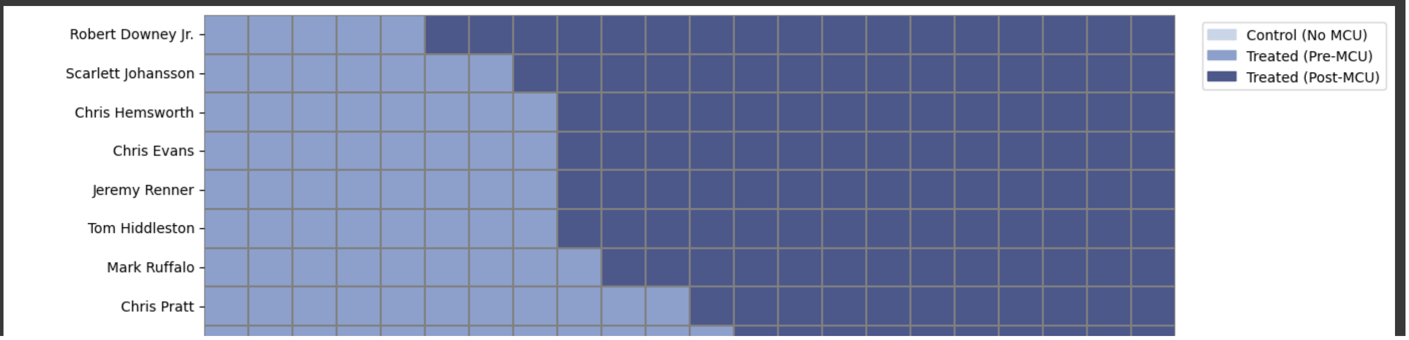
Visualizing Actor Treatment Timeline

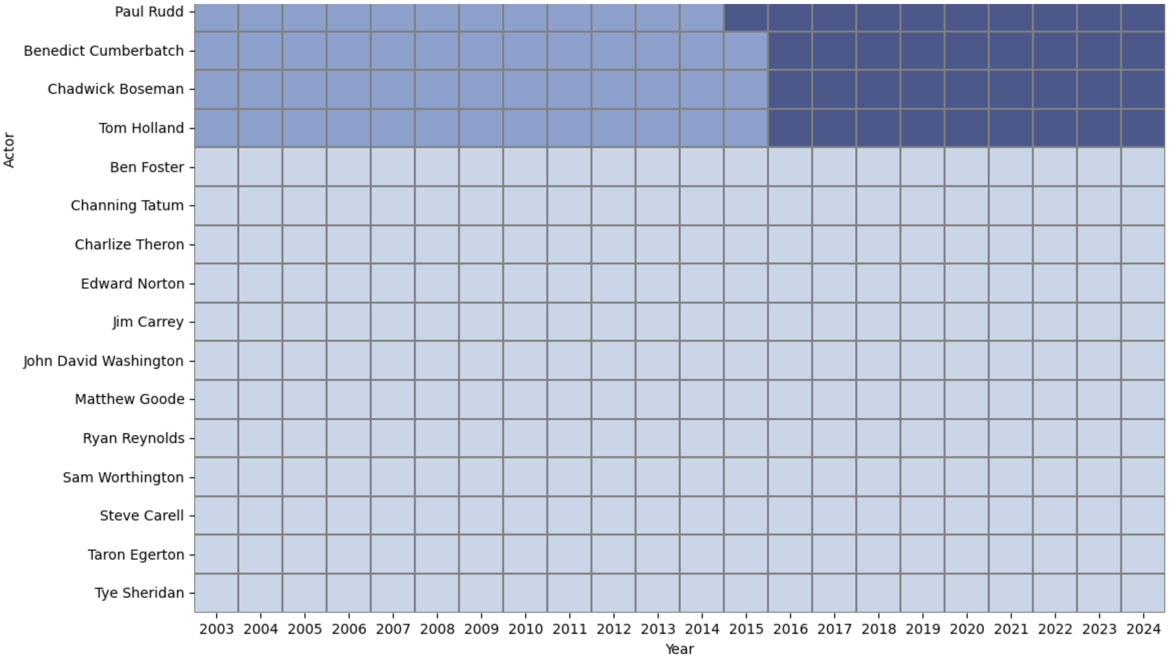
```
heatmap_df = treatment_control.groupby('actor_name').agg({'MCU Entry Year':"sum",
heatmap_df.head()
```

	MCU Entry Year	Treatment
actor_name		
Ben Foster	0	0
Benedict Cumberbatch	2016	1
Chadwick Boseman	2016	1
Channing Tatum	0	0
Charlize Theron	0	0

Staggered DiD Diagram

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

# --- Step 1: Setup base matrix ---
years = list(range(2003, 2025))
actors = heatmap_df.index.tolist()

# Create full grid: rows = actors, columns = years
status_matrix = pd.DataFrame(index=actors, columns=years)

# --- Step 2: Fill in treatment status ---
for actor in actors:
    entry_year = heatmap_df.loc[actor, 'MCU Entry Year']
    treated = heatmap_df.loc[actor, 'Treatment']

    for year in years:
        if treated == 0:
            status_matrix.loc[actor, year] = 0  # control
        elif year < entry_year:
            status_matrix.loc[actor, year] = 1  # treated (pre)
        else:
            status_matrix.loc[actor, year] = 2  # treated (post)

status_matrix = status_matrix.astype(float)  # convert to numeric for heatmap

# --- Step 3: Sort actors by entry year ---
entry_order = heatmap_df[heatmap_df['Treatment'] == 1].sort_values('MCU Entry Yea
control_order = heatmap_df[heatmap_df['Treatment'] == 0].index.tolist()
status_matrix = status_matrix.loc[entry_order + control_order]

# --- Step 4: Plot heatmap ---
plt.figure(figsize=(14, 10))
cmap = sns.color_palette(["#cbd5e8", "#8da0cb", "#4c578a"])  # light -> dark

sns.heatmap(
    status_matrix,
    cmap=cmap,
    linewidths=0.2,
    linecolor='gray',
    cbar=False
)

plt.xlabel("Year")
plt.ylabel("Actor")

# Legend
legend_patches = [
    mpatches.Patch(color="#cbd5e8", label='Control (No MCU)'),
    mpatches.Patch(color="#8da0cb", label='Treated (Pre-MCU)'),
    mpatches.Patch(color="#4c578a", label='Treated (Post-MCU)')
]
plt.legend(handles=legend_patches, bbox_to_anchor=(1.02, 1), loc='upper left')

plt.tight_layout()
plt.show()
```



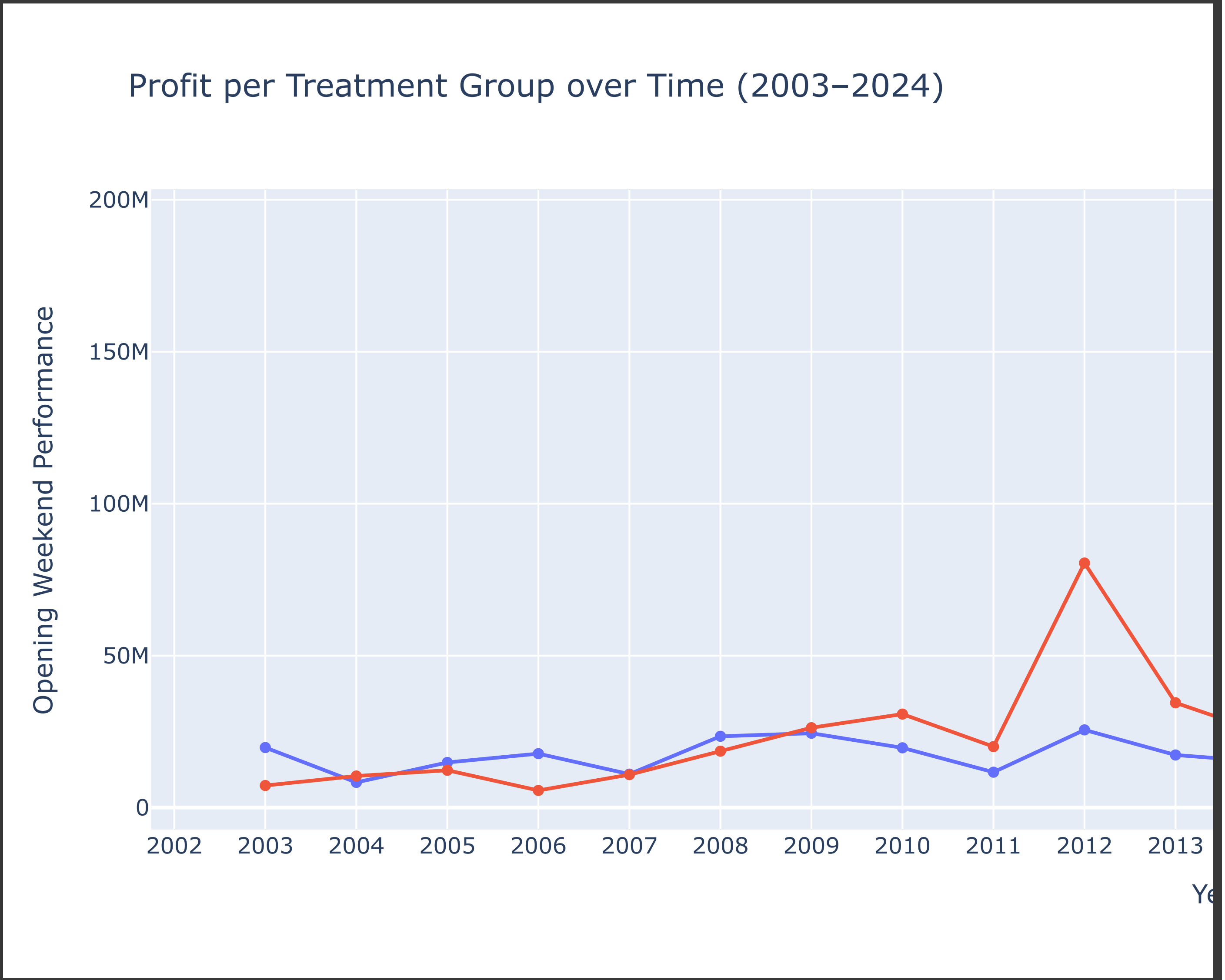

Paul Rudd
Benedict Cumberbatch
Chadwick Boseman
Tom Holland
Ben Foster
Channing Tatum
Charlize Theron
Edward Norton
Jim Carrey
John David Washington
Matthew Goode
Ryan Reynolds
Sam Worthington
Steve Carell
Taron Egerton
Tye Sheridan
Actor
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Year

```
treatment_control.columns
```

```
Index(['title', 'budget', 'revenue', 'runtime', 'release_date', 'cast_order',
       'actor_name', 'imdb_votes', 'Internet Movie Database',
       'Rotten Tomatoes', 'Metacritic', 'opening_weekend', 'Action',
       'Adventure', 'Comedy', 'Drama', 'Family', 'Fantasy', 'Horror',
'Music',
       'Mystery', 'Romance', 'Science Fiction', 'Thriller', 'War', 'Western',
       'MCU', 'MCU Entry', 'release_year', 'release_quarter', 'Treatment',
       'Lead Role', 'Supporting Role', 'MCU Entry Year'],
      dtype='object')
```

Preliminary Data Analysis

```
import plotly.express as px

# ✅ Step 1: Choose the metric
df = treatment_control.copy()
#df = df[df['Treatment']==1]
df['profit'] = df['revenue'] - df['budget']

# ✅ Step 2: Filter data for years and valid entries
df = df[df['release_year'].between(2003, 2024)]
df = df.dropna(subset=['Treatment', 'release_year', 'budget', 'revenue'])

# Optional: limit to lead roles only
# df = df[df['Lead Role'] == 1]

# ✅ Step 3: Aggregate metric per actor per year
agg = df.groupby(['Treatment', 'release_year']).agg({
    'budget': 'mean',
    'revenue': 'mean',
    'opening_weekend': 'mean'
}).reset_index()

# ✅ Step 4: Plot with Plotly
fig = px.line(
    agg,
    x='release_year',
    y='opening_weekend',  # Change to 'budget' or 'revenue' if needed
    color='Treatment',
    markers=True,
    title='Profit per Treatment Group over Time (2003–2024)',
    labels={'release_year': 'Year', 'opening_weekend': 'Opening Weekend Performan
    hover_name='Treatment'
)

fig.update_layout(
    xaxis=dict(tickmode='linear', dtick=1),
    hovermode='x unified',
    legend_title='Treatment'
)

fig.show()
```



```
treatment_control['opening_weekend_log'] = np.log(treatment_control["opening_week
import statsmodels.formula.api as smf
```

```
# 1) TWFE regression
twfe = smf.ols('opening_weekend_log ~ Treatment + C(actor_name) + C(release_year)
print("TWFE treat coef:", twfe.params['Treatment'], "p=", twfe.pvalues['Treatment
```

```
TWFE treat coef: 1.9147393638815493 p= 0.0035044674802845077
```

```
treatment_control.T
```

	272	274	1	335	169	250
title	My Life Without Me	View from the Top	The Shape of Things	S.W.A.T.	Lost in Translation	In the Cut
budget	2500000	30000000	0	80000000	4000000	12000000
revenue	12300000	19526014	735992	207700000	119723856	23726793
runtime	106	87	96	117	102	119
release_date	2003-03-07 00:00:00	2003-03-21 00:00:00	2003-07-24 00:00:00	2003-08-08 00:00:00	2003-09-18 00:00:00	2003-10-22 00:00:00
cast_order	3	2	0	5	1	1
actor_name	Mark Ruffalo	Mark Ruffalo	Paul Rudd	Jeremy Renner	Scarlett Johansson	Mark Ruffalo
imdb_votes	26,182	29,610	11,962	156,467	507,784	26,960
Internet Movie Database	7.4	5.3	6.6	6.1	7.7	5.4
Rotten Tomatoes	NaN	14.0	64.0	48.0	95.0	35.0
Metacritic	57.0	27.0	59.0	45.0	91.0	47.0
opening_weekend	40515.0	7600000.0	173246.0	37062535.0	925087.0	97625.0
Action	0	0	0	1	0	0
Adventure	0	0	0	0	0	0
Comedy	0	1	1	0	1	0

Drama	1	1	1	0	1	1
Family	0	0	0	0	0	0
Fantasy	0	0	0	0	0	0
Horror	0	0	0	0	0	0
Music	0	0	0	0	0	0
Mystery	0	0	0	0	0	1
Romance	1	1	1	0	1	1
Science Fiction	0	0	0	0	0	0
Thriller	0	0	0	1	0	1
War	0	0	0	0	0	0
Western	0	0	0	0	0	0
MCU	0	0	0	0	0	0
MCU Entry	0	0	0	0	0	0
release_year	2003	2003	2003	2003	2003	2003
release_quarter	1	1	3	3	3	4
Treatment	1	1	1	1	1	1
Lead Role	0	0	1	0	1	1
Supporting Role	1	1	0	1	0	0
MCU Entry Year	0	0	0	0	0	0
opening_weekend_log	10.609428	15.843659	12.062468	17.428117	13.737643	11.488889

35 rows × 670 columns

```
model2 = treatment_control.groupby(["actor_name","release_year"]).agg(count_lead_
                                                                     average_rat
                                                                     film_count=
                                                                     opening_wee
                                                                     Treatment=(
model2.dropna(axis=0,inplace=True)
```

```python
model2.head()
```

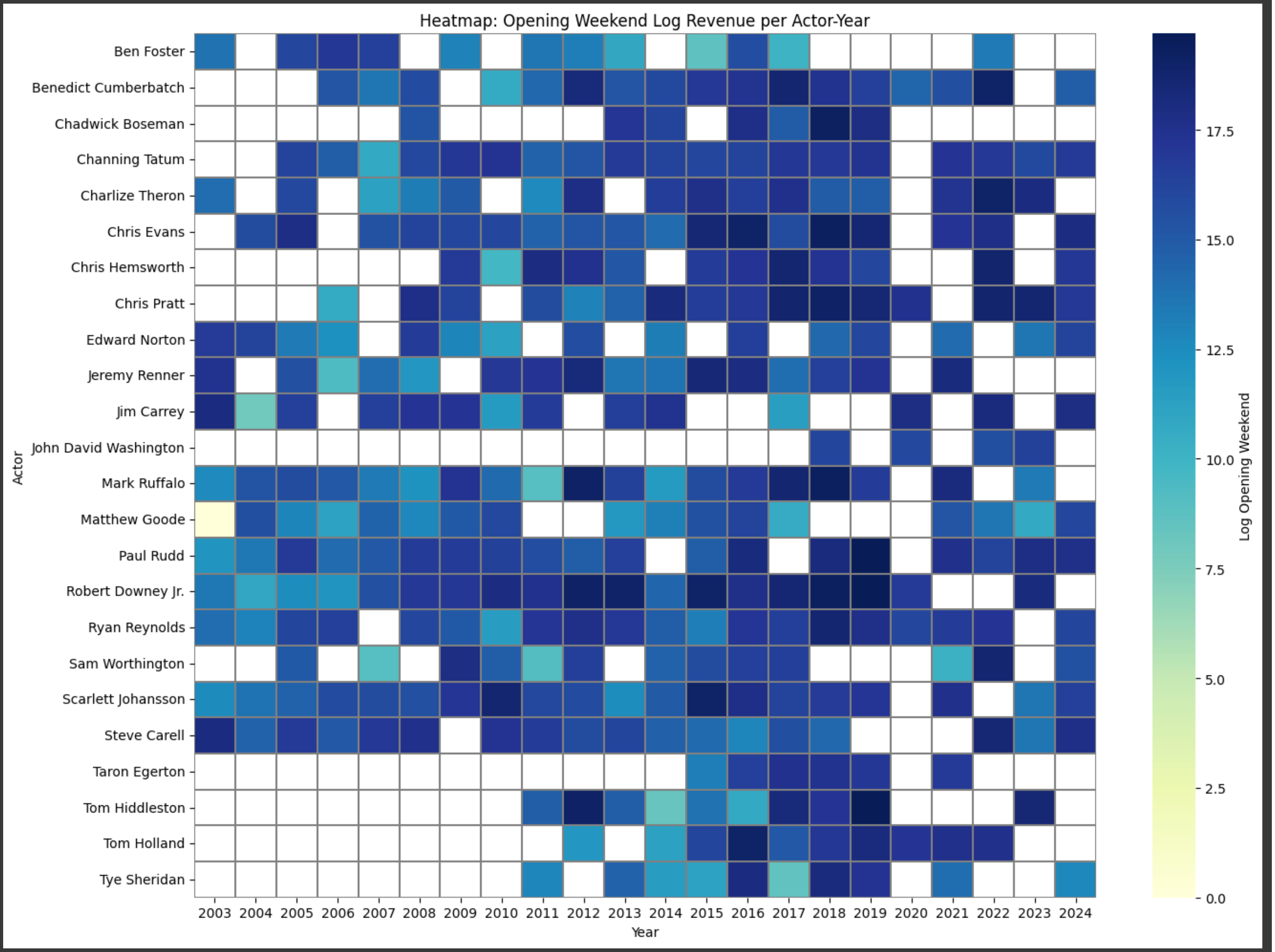
	actor_name	release_year	count_lead_roles	average_rating	film_count	open
0	Ben Foster	2003	0	6.65	2	
2	Ben Foster	2005	1	6.50	1	
3	Ben Foster	2006	0	6.75	2	
4	Ben Foster	2007	0	7.10	2	
6	Ben Foster	2009	2	6.90	2	

Exploratory Data Analysis

```python
df_event = df_event[df_event['Treatment'] == 1]  # only treated actors
df_event['event_time'] = df_event['release_year'] - df_event['MCU Entry Year']
df_event = df_event[df_event['event_time'].between(-5, 10)]

# Group: average outcome by event time
agg = df_event.groupby('event_time')['opening_weekend_log'].mean().reset_index()

pivot = treatment_control.pivot_table(
    index='actor_name',
    columns='release_year',
    values='opening_weekend_log',
    aggfunc='mean'
)

plt.figure(figsize=(14, 10))
sns.heatmap(pivot, cmap='YlGnBu', linewidths=0.1, linecolor='gray', cbar_kws={'la
plt.title("Heatmap: Opening Weekend Log Revenue per Actor-Year")
plt.xlabel("Year")
plt.ylabel("Actor")
plt.tight_layout()
plt.show()
```

```
<ipython-input-224-a3a6793fb099>:2: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
```



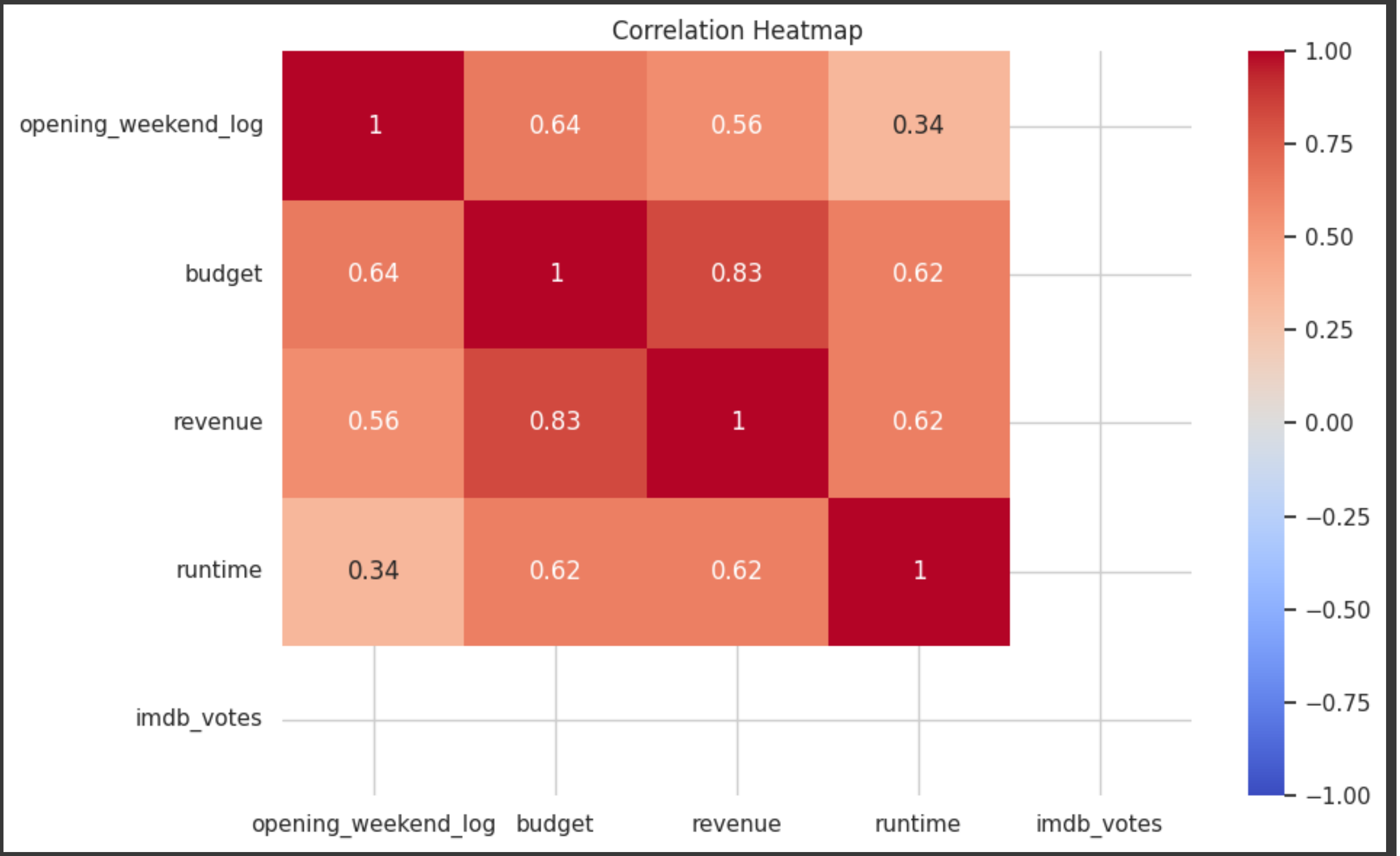
```
treatment_control.reset_index(drop=True).to_json("treatment_control.json",orient=
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Load your dataset (adjust path if needed)
df = pd.read_json("treatment_control.json")
```

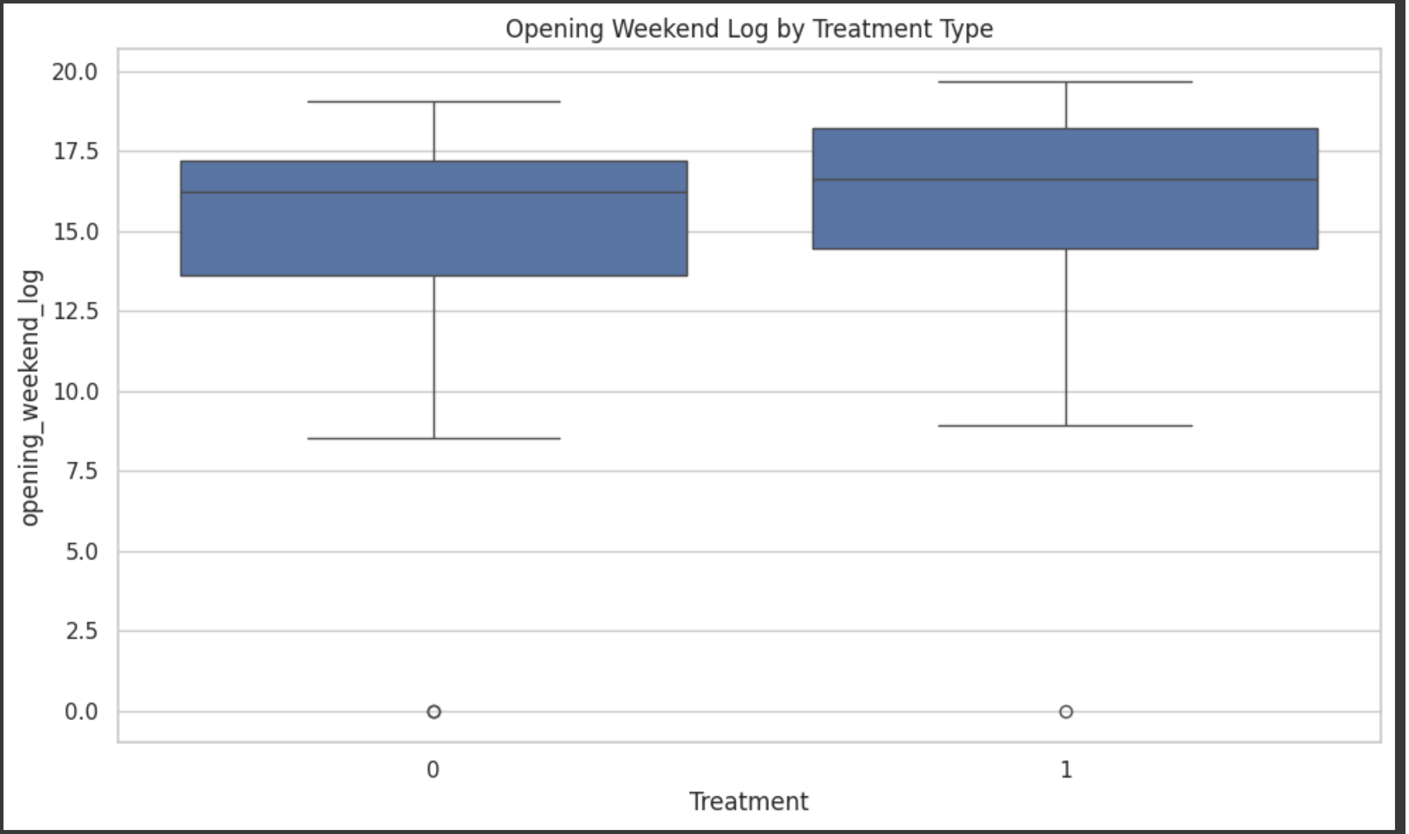
```
# Convert key numeric columns
df['opening_weekend_log'] = pd.to_numeric(df.get('opening_weekend_log'), errors='
df['budget'] = pd.to_numeric(df.get('budget'), errors='coerce')
df['revenue'] = pd.to_numeric(df.get('revenue'), errors='coerce')
df['runtime'] = pd.to_numeric(df.get('runtime'), errors='coerce')
df['imdb_votes'] = pd.to_numeric(df.get('imdb_votes'), errors='coerce')
df['release_year'] = pd.to_numeric(df.get('release_year'), errors='coerce')

# Set seaborn style
sns.set(style="whitegrid")
```

```
numeric_cols = ['opening_weekend_log', 'budget', 'revenue', 'runtime', 'imdb_vote
```

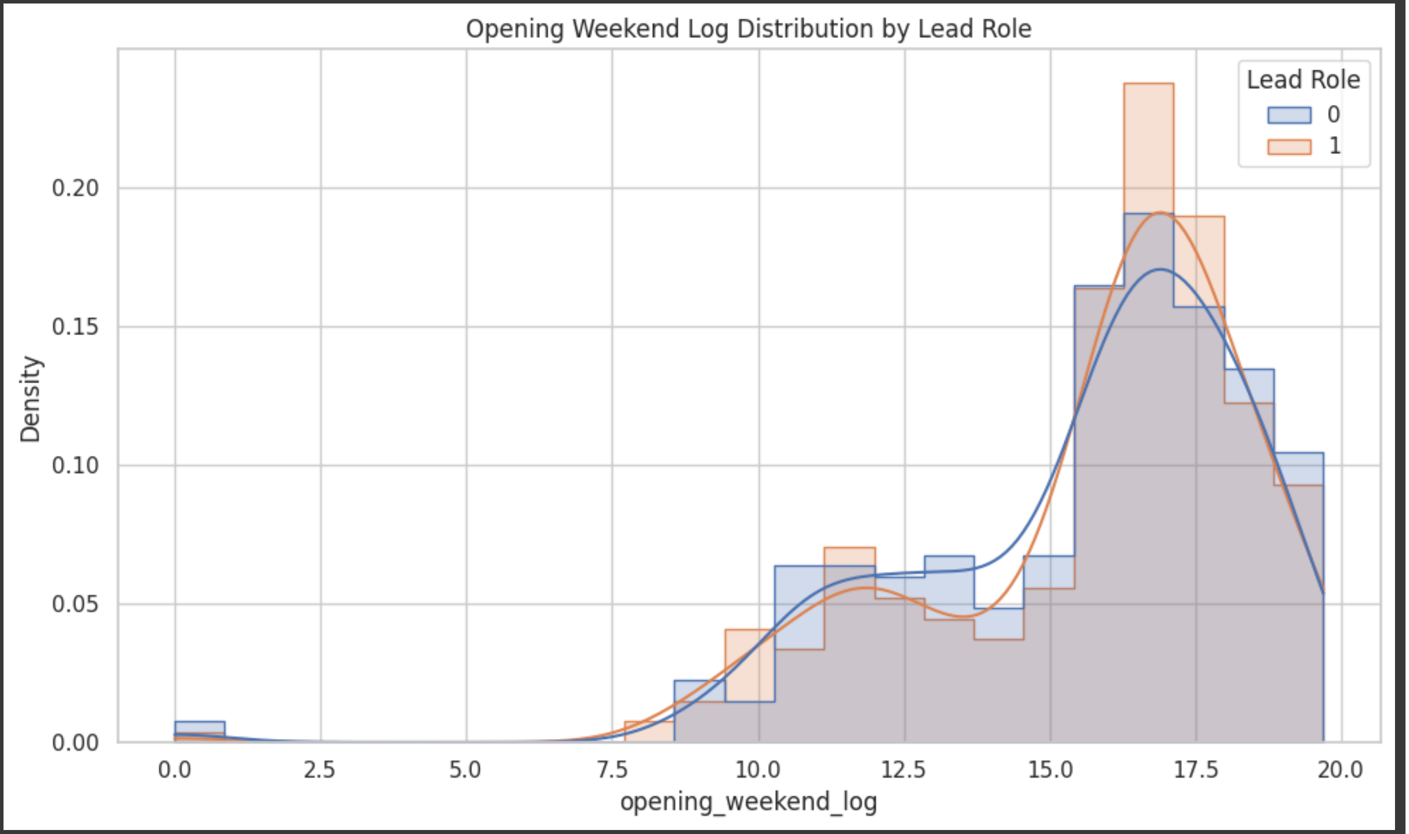
```
# 2. Correlation Heatmap
plt.figure(figsize=(10, 6))
corr = df[numeric_cols].corr()
sns.heatmap(corr, annot=True, cmap="coolwarm", vmin=-1, vmax=1)
plt.title("Correlation Heatmap")
plt.tight_layout()
plt.show()
```



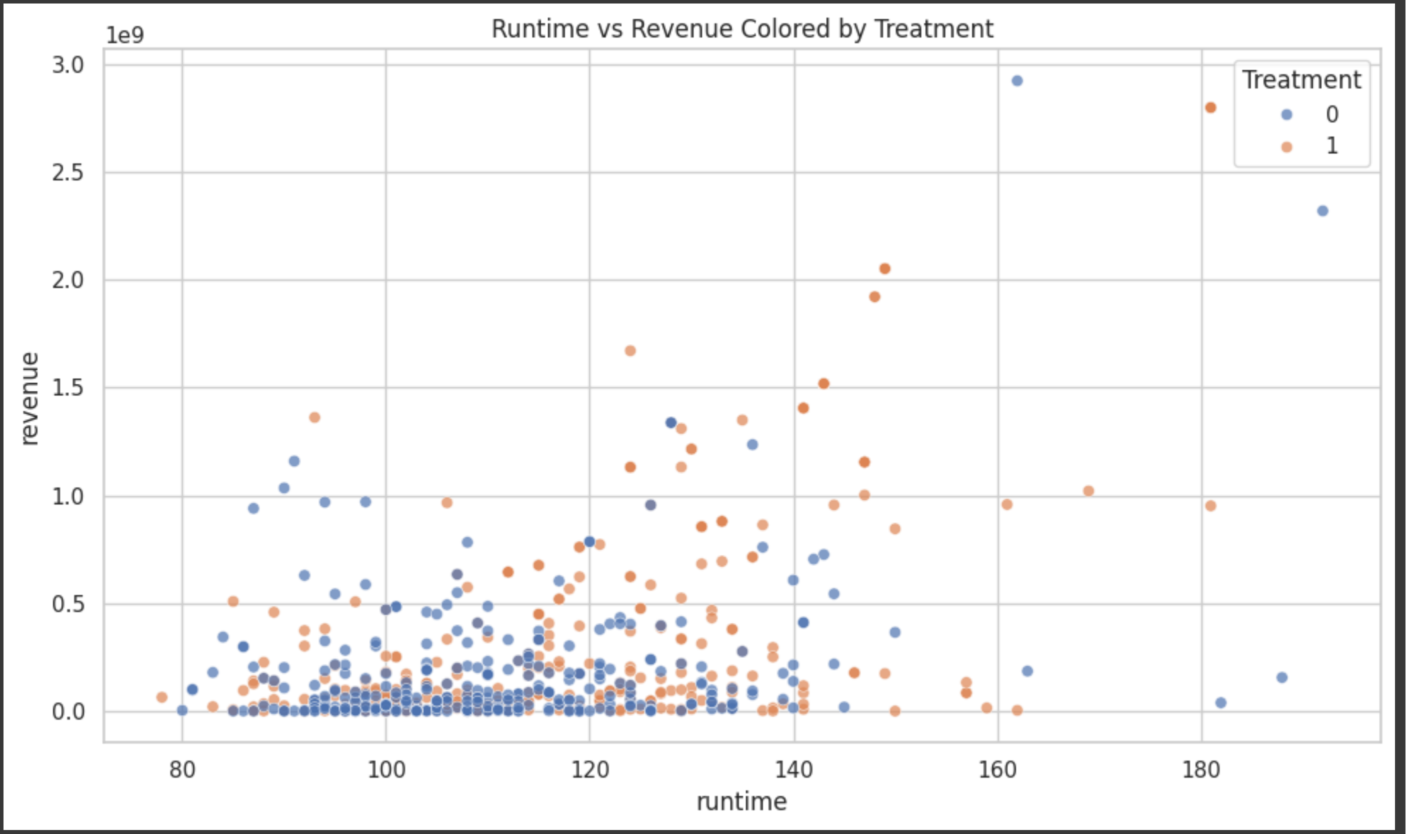
```
# 3. Boxplot: Treatment vs Opening Weekend
plt.figure(figsize=(10, 6))
sns.boxplot(x='Treatment', y='opening_weekend_log', data=df)
plt.title("Opening Weekend Log by Treatment Type")
plt.tight_layout()
plt.show()
```



```
# 4. Distribution: Lead Role
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='opening_weekend_log', hue='Lead Role', kde=True, element
plt.title("Opening Weekend Log Distribution by Lead Role")
plt.tight_layout()
plt.show()
```

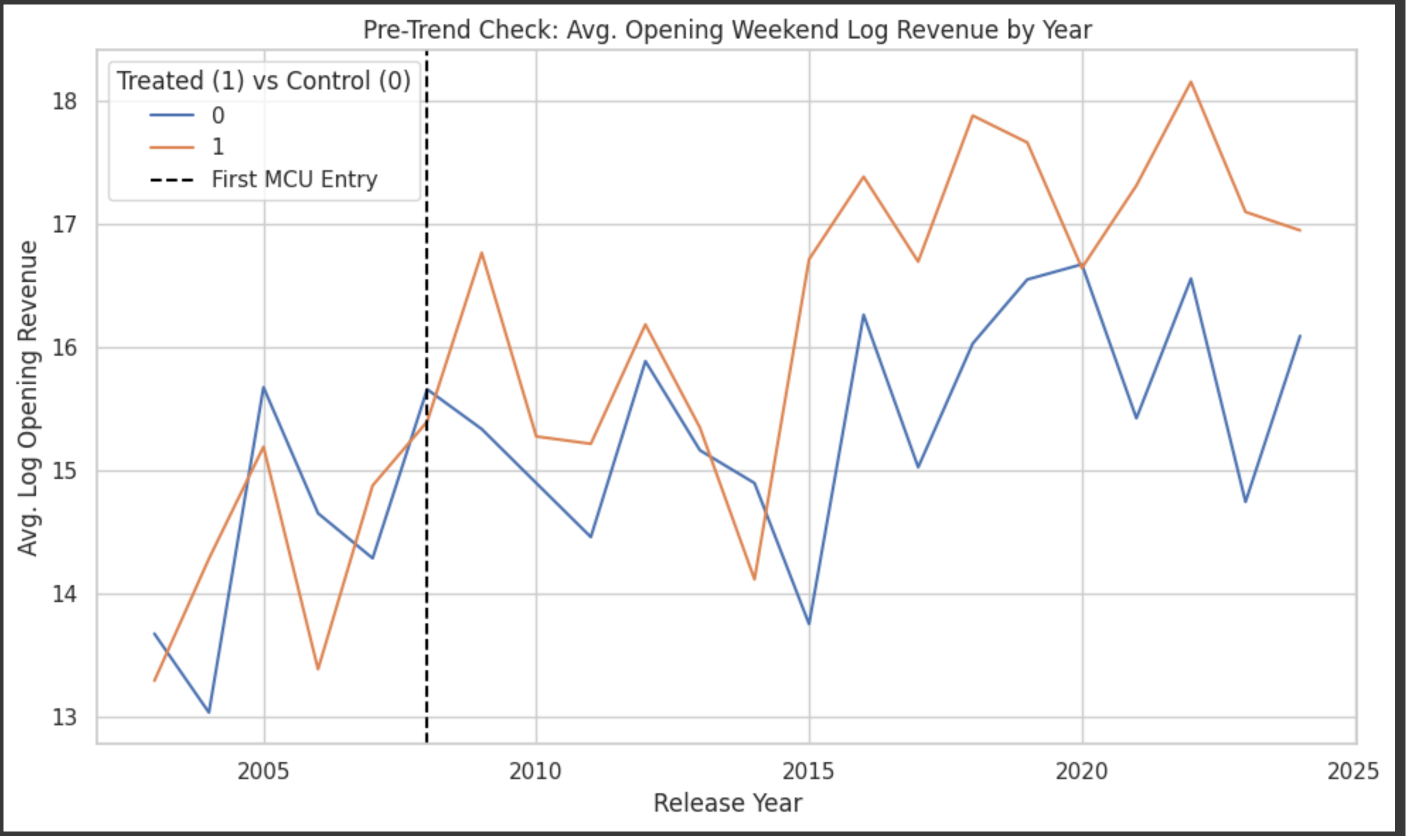


```
a# 5. Scatterplot: Runtime vs Revenue by Treatment
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='runtime', y='revenue', hue='Treatment', alpha=0.7)
plt.title("Runtime vs Revenue Colored by Treatment")
plt.tight_layout()
plt.show()
```



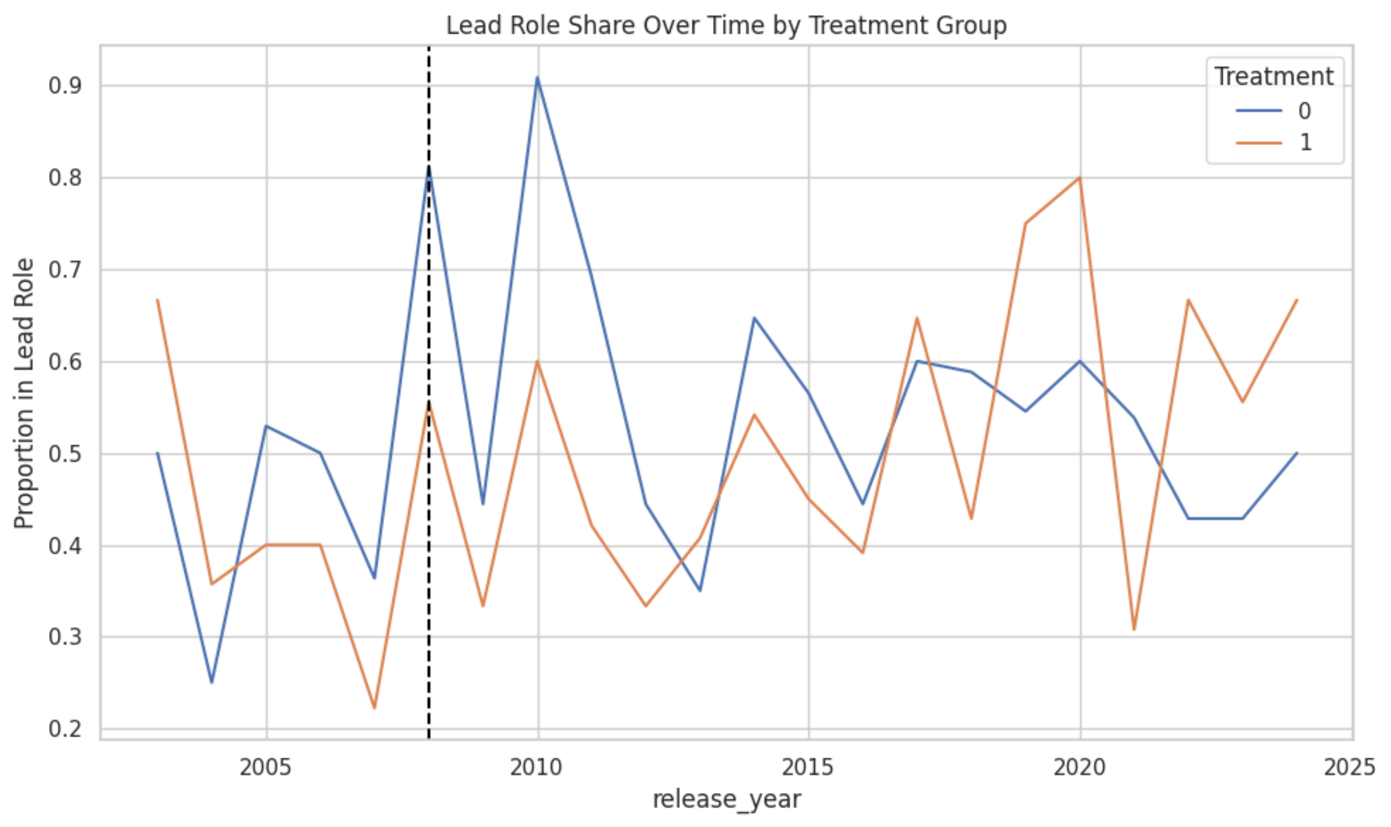
```
plt.figure(figsize=(10, 6))
sns.lineplot(data=df, x='release_year', y='opening_weekend_log', hue='Treatment',
plt.title("Pre-Trend Check: Avg. Opening Weekend Log Revenue by Year")
plt.axvline(x=2008, color='black', linestyle='--', label="First MCU Entry")
plt.xlabel("Release Year")
plt.ylabel("Avg. Log Opening Revenue")
plt.legend(title="Treated (1) vs Control (0)")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
<ipython-input-248-318f76580552>:2: FutureWarning:


The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.
```

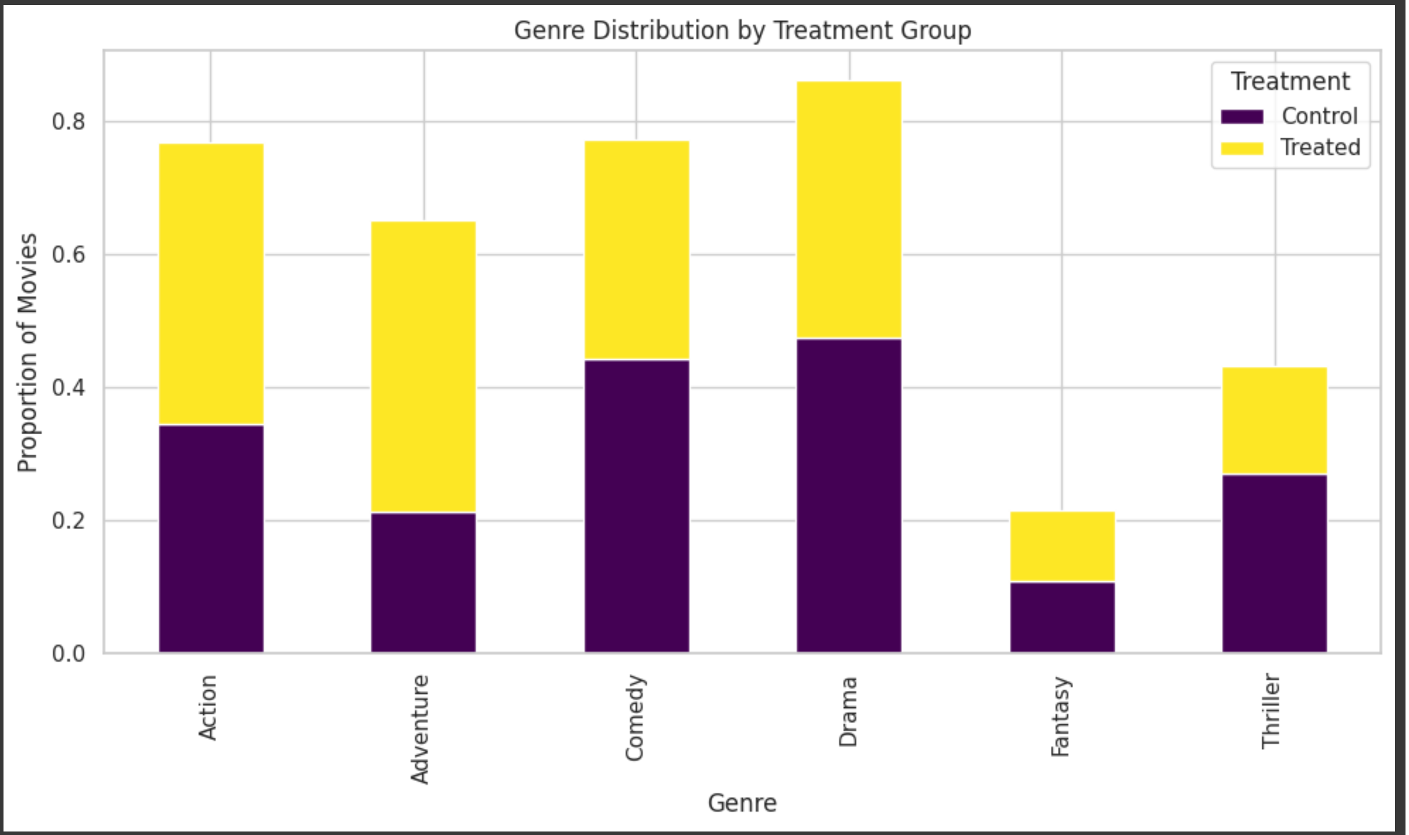


```
plt.figure(figsize=(10, 6))
plt.figure(figsize=(10, 6))
sns.lineplot(
    data=df,
    x='release_year',
    y='Lead Role',
    hue='Treatment',
    estimator='mean',
    ci=None
)
plt.axvline(x=2008, color='black', linestyle='--', label="First MCU Entry")
plt.title("Lead Role Share Over Time by Treatment Group")
plt.ylabel("Proportion in Lead Role")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
<ipython-input-250-41f3763148ea>:2: FutureWarning:


The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.
```

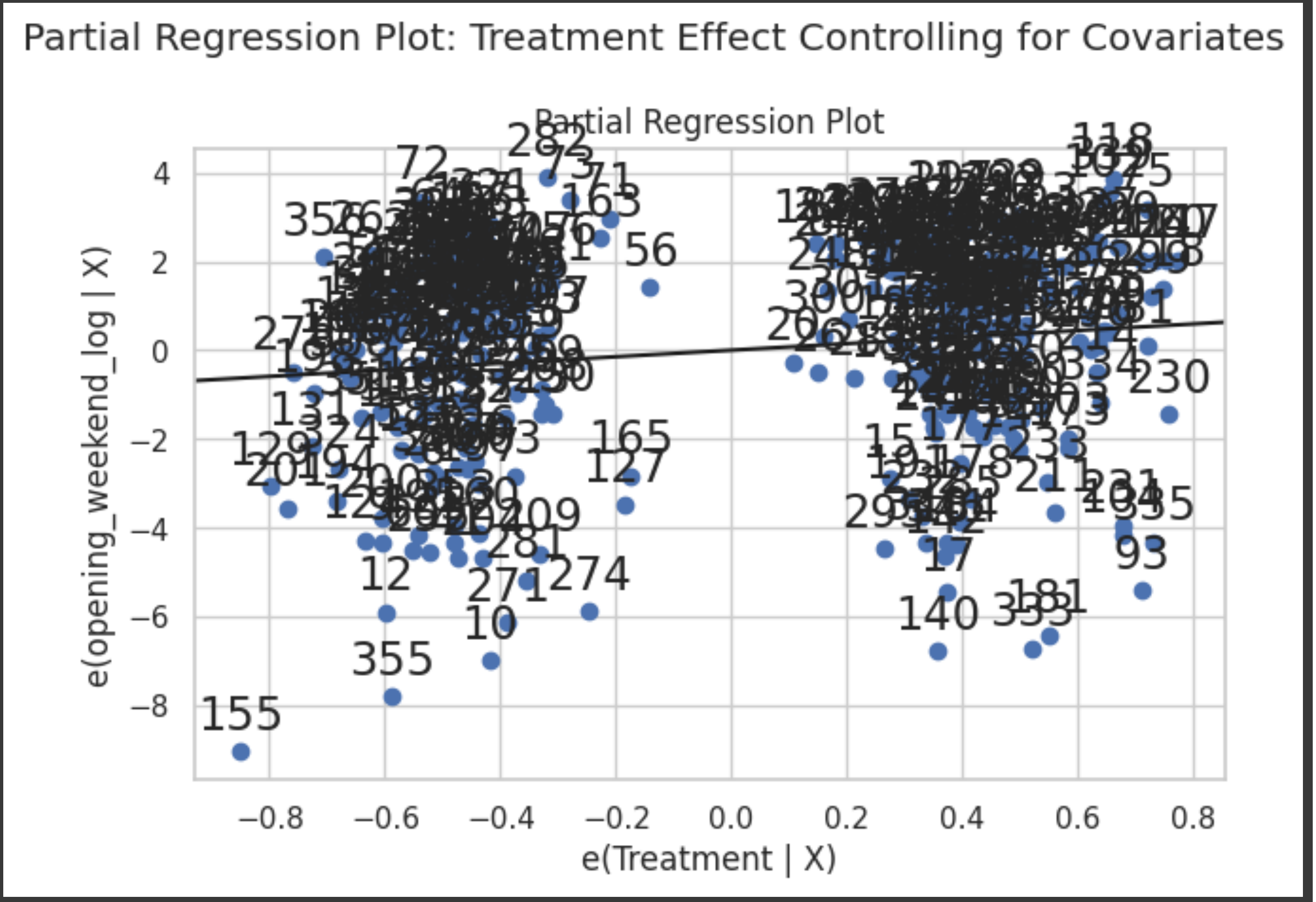


```
genre_cols = ['Action', 'Adventure', 'Comedy', 'Drama', 'Fantasy', 'Thriller']
genre_share = df.groupby('Treatment')[genre_cols].mean().T

genre_share.plot(kind='bar', stacked=True, figsize=(10, 6), colormap='viridis')
plt.title("Genre Distribution by Treatment Group")
plt.ylabel("Proportion of Movies")
plt.xlabel("Genre")
plt.legend(title="Treatment", labels=['Control', 'Treated'])
plt.tight_layout()
plt.show()
```



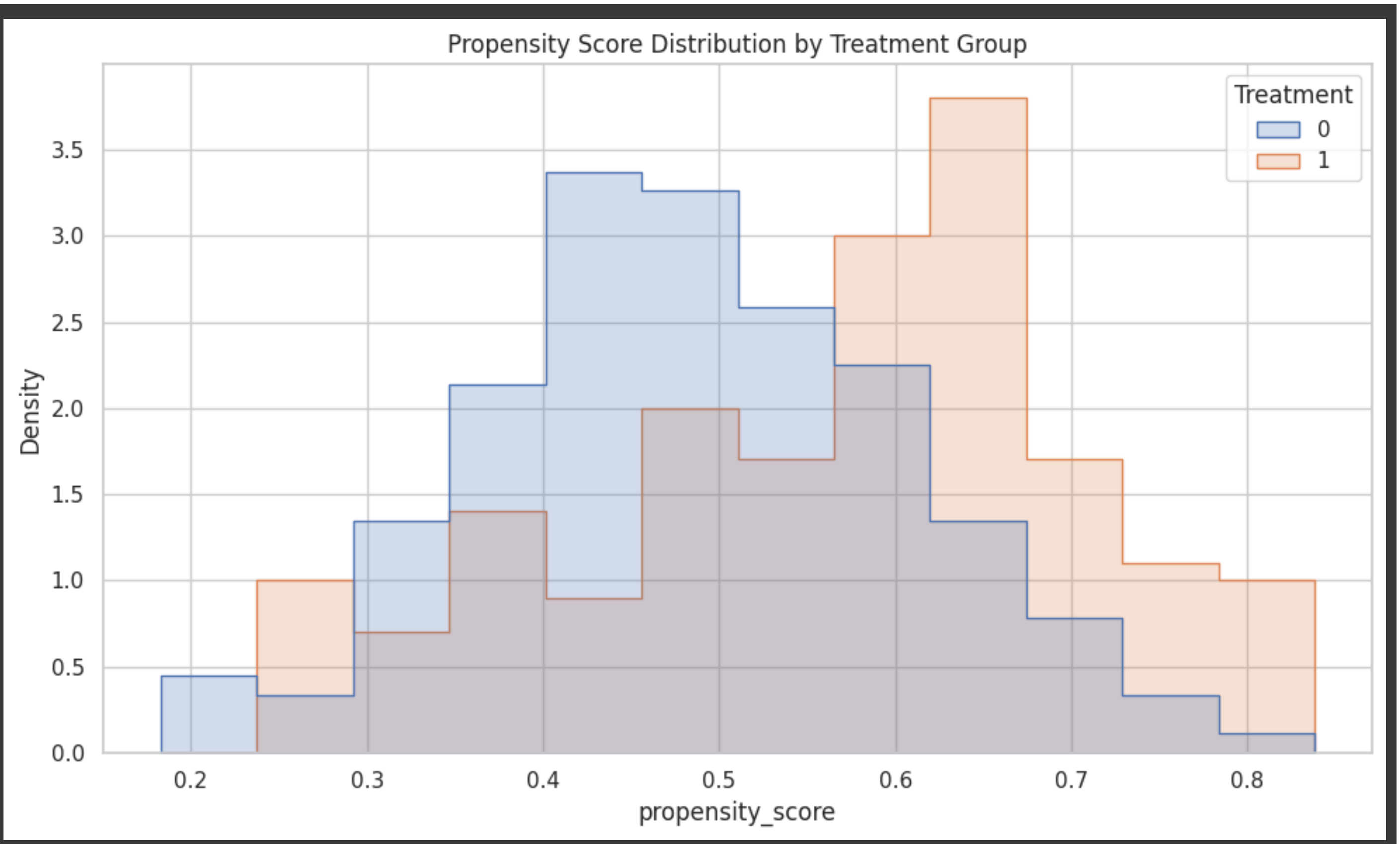
```
import statsmodels.api as sm
```

```
# Run OLS model with relevant controls
model = smf.ols(
    formula='opening_weekend_log ~ Treatment + average_rating + film_count + coun
    data=model2
).fit()

# Plot partial regression (added variable plot)
fig = sm.graphics.plot_partregress('opening_weekend_log', 'Treatment', ['average_
fig.suptitle("Partial Regression Plot: Treatment Effect Controlling for Covariate
plt.tight_layout()
plt.show()
```

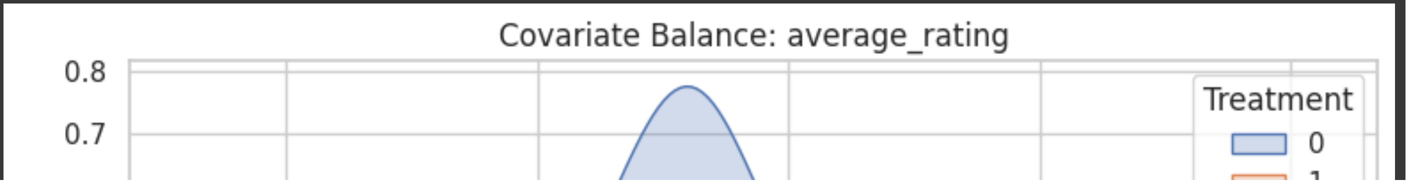


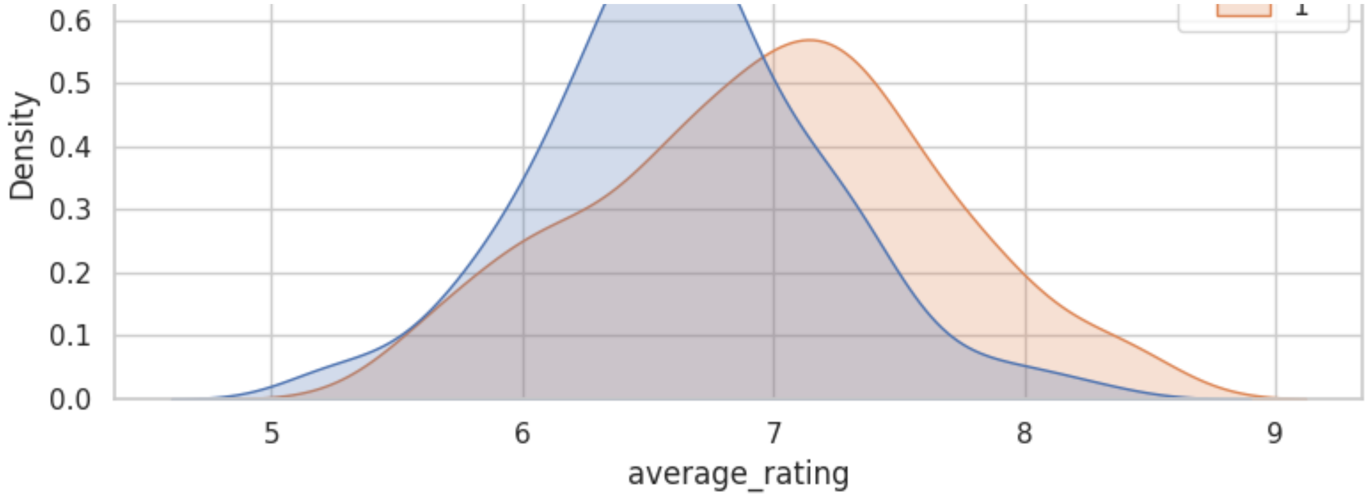
```
from sklearn.linear_model import LogisticRegression

# Define feature set for matching
```

```
X = model2[['average_rating', 'film_count', 'count_lead_roles']].dropna()
y = model2.loc[X.index, 'Treatment']

# Fit logistic model
ps_model = LogisticRegression()
ps_model.fit(X, y)
model2.loc[X.index, 'propensity_score'] = ps_model.predict_proba(X)[:, 1]

# Plot
plt.figure(figsize=(10, 6))
sns.histplot(data=model2, x='propensity_score', hue='Treatment', element='step',
plt.title("Propensity Score Distribution by Treatment Group")
plt.tight_layout()
plt.show()
```

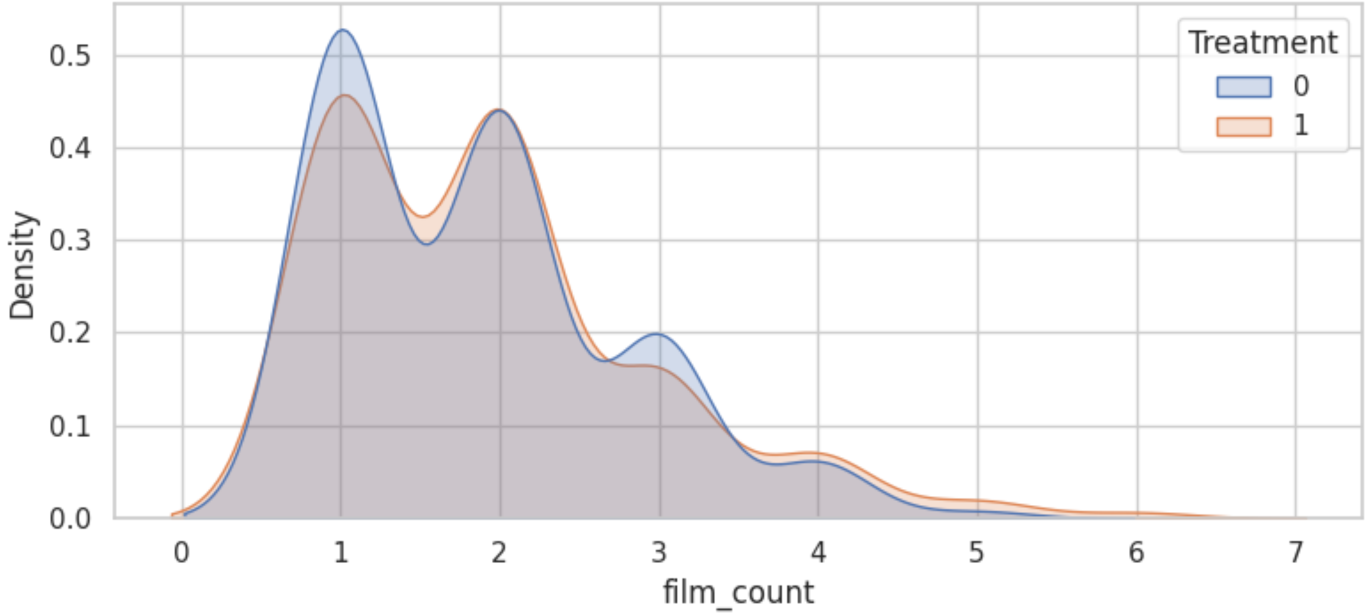


```
# Visualize distribution of covariates
for var in ['average_rating', 'film_count', 'count_lead_roles']:
    plt.figure(figsize=(8, 4))
    sns.kdeplot(data=model2, x=var, hue='Treatment', fill=True, common_norm=False
    plt.title(f"Covariate Balance: {var}")
    plt.tight_layout()
    plt.show()
```

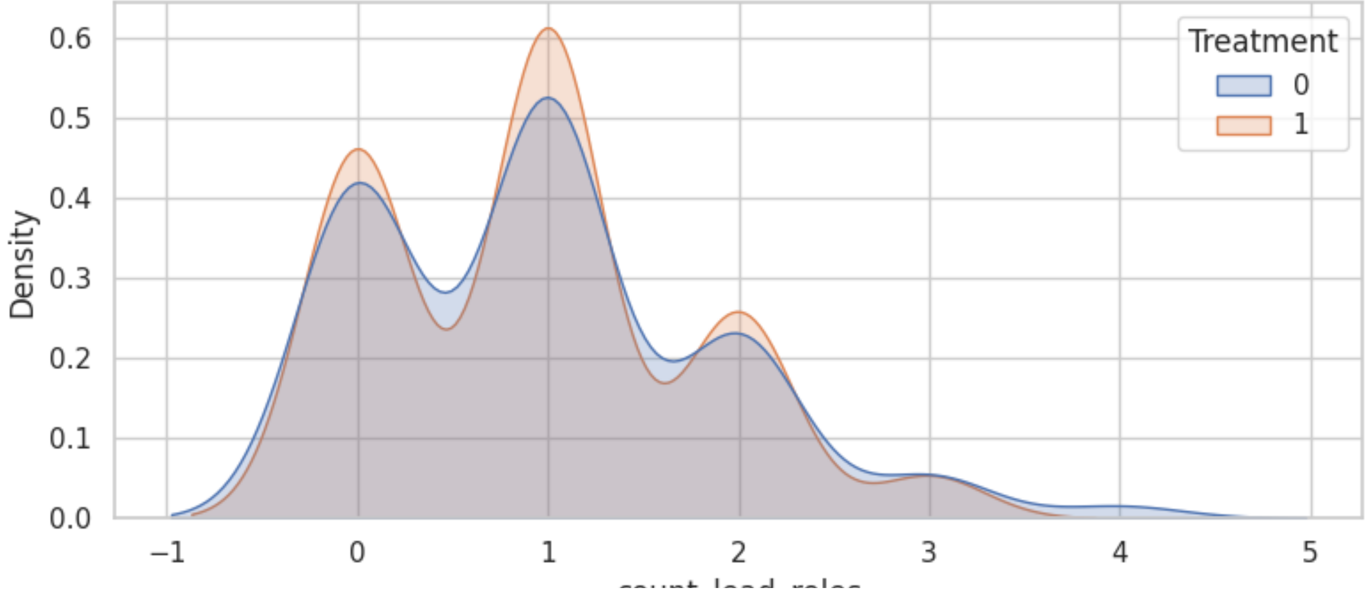


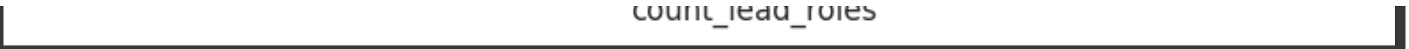


Covariate Balance: film_count



Covariate Balance: count_lead_roles





Weighted SMF model

```
# Add inverse probability weights (IPW)
model2['ipw'] = np.where(
    model2['Treatment'] == 1,
    1 / model2['propensity_score'],
    1 / (1 - model2['propensity_score'])
)

# Weighted regression
weighted_model = smf.wls(
    formula='opening_weekend_log ~ Treatment',
    data=model2,
    weights=model2['ipw']
).fit()

print(weighted_model.summary())
```

```
                            WLS Regression Results
==============================================================================
Dep. Variable:     opening_weekend_log   R-squared:                       0.03
Model:                             WLS   Adj. R-squared:                  0.02
Method:                  Least Squares   F-statistic:                     10.8
Date:                 Wed, 07 May 2025   Prob (F-statistic):            0.0010
Time:                         18:00:04   Log-Likelihood:                -799.8
No. Observations:                  346   AIC:                             1604
Df Residuals:                      344   BIC:                             1611
Df Model:                            1
Covariance Type:             nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     15.1462      0.184     82.364      0.000      14.784      15.508
Treatment      0.8504      0.258      3.296      0.001       0.343       1.358
==============================================================================
Omnibus:                     82.047   Durbin-Watson:                   1.732
Prob(Omnibus):                0.000   Jarque-Bera (JB):              156.772
Skew:                        -1.274   Prob(JB):                     9.07e-35
Kurtosis:                     5.094   Cond. No.                        2.64
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
```

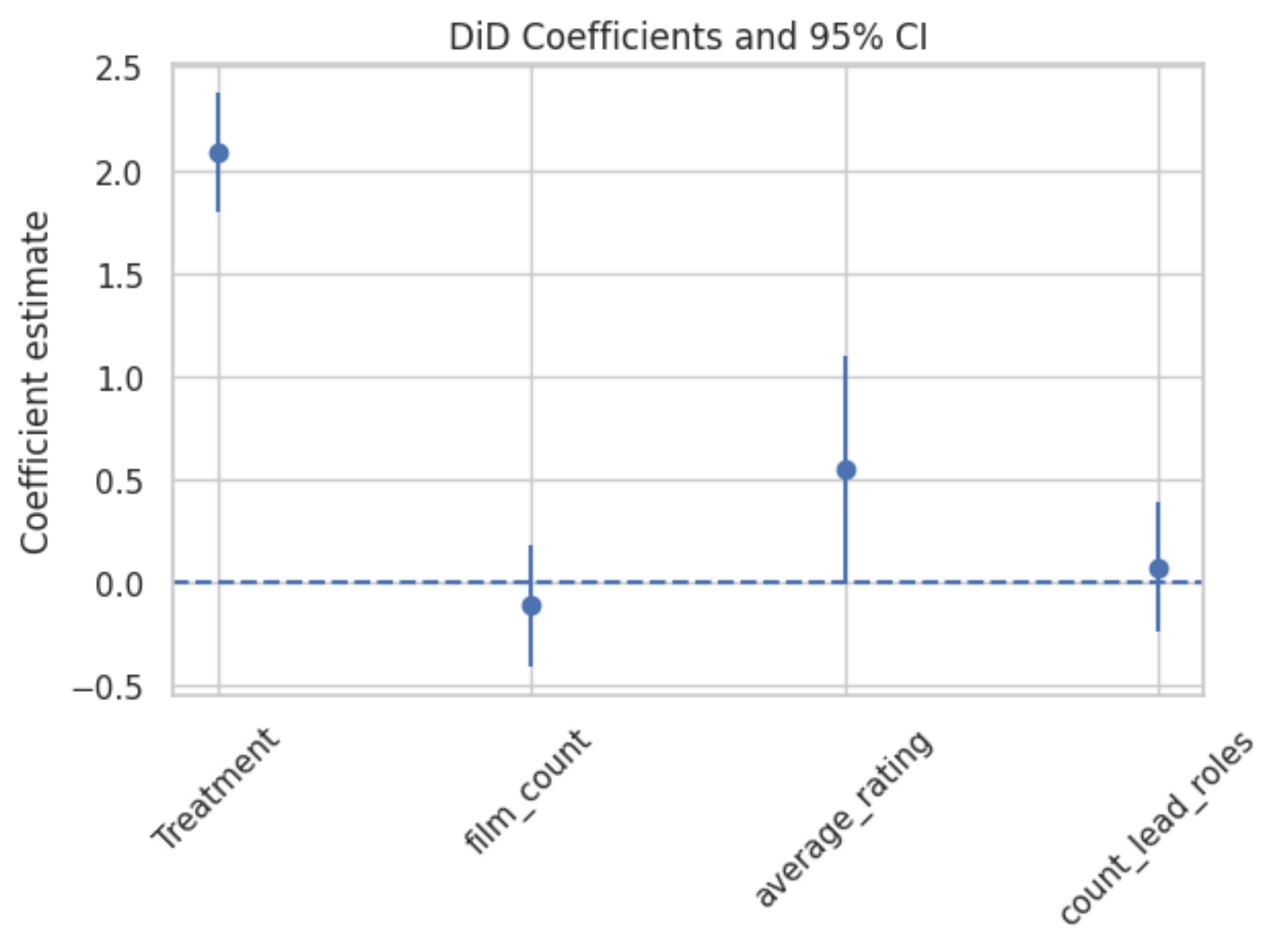
Difference-in-Differences (DiD) Model Estimation

```
# 2. Estimate the DiD with controls + fixed effects
# 3. Estimate the DiD with controls + fixed effects
formula = (
    'opening_weekend_log ~ Treatment'
    ' + film_count + average_rating + count_lead_roles'
    ' + C(actor_name) + C(release_year)'
)

did_model = smf.ols(formula, data=model2) \
              .fit(cov_type='cluster', cov_kwds={'groups': model2['actor_name']}

print(did_model.summary())
```

```
                                     OLS Regression Results
    ==========================================================================================
    Dep. Variable:     opening_weekend_log   R-squared:                          0.3
    Model:                             OLS   Adj. R-squared:                     0.2
    Method:                  Least Squares   F-statistic:                        107.
    Date:                 Wed, 07 May 2025   Prob (F-statistic):              2.60e-
    Time:                         18:05:40   Log-Likelihood:                   -718.
    No. Observations:                  346   AIC:                                153
    Df Residuals:                      298   BIC:                                171
    Df Model:                           47
    Covariance Type:               cluster
    ==========================================================================================
                                                 coef    std err          z      P
    ------------------------------------------------------------------------------------------
    Intercept                                  9.4373      1.999      4.722      0
    C(actor_name)[T.Benedict Cumberbatch]     -0.2466      0.095     -2.596      0
    C(actor_name)[T.Chadwick Boseman]          0.6344      0.157      4.042      0
    C(actor_name)[T.Channing Tatum]            2.5380      0.135     18.792      0
    C(actor_name)[T.Charlize Theron]           1.9890      0.149     13.372      0
    C(actor_name)[T.Chris Evans]               0.7595      0.078      9.711      0
    C(actor_name)[T.Chris Hemsworth]           0.2444      0.126      1.939      0
    C(actor_name)[T.Chris Pratt]               0.7973      0.114      7.004      0
    C(actor_name)[T.Edward Norton]             0.8664      0.165      5.253      0
    C(actor_name)[T.Jeremy Renner]            -0.1319      0.085     -1.546      0
    C(actor_name)[T.Jim Carrey]                2.4628      0.198     12.439      0
    C(actor_name)[T.John David Washington]     1.2770      0.339      3.769      0
    C(actor_name)[T.Mark Ruffalo]             -0.4173      0.094     -4.422      0
    C(actor_name)[T.Matthew Goode]             0.3049      0.174      1.752      0
    C(actor_name)[T.Paul Rudd]                 0.5115      0.096      5.339      0
    C(actor_name)[T.Robert Downey Jr.]         0.8082      0.115      7.024      0
    C(actor_name)[T.Ryan Reynolds]             2.2230      0.182     12.206      0
    C(actor_name)[T.Sam Worthington]           1.0058      0.178      5.655      0
    C(actor_name)[T.Scarlett Johansson]        0.1605      0.090      1.784      0
    C(actor_name)[T.Steve Carell]              2.4944      0.146     17.045      0
    C(actor_name)[T.Taron Egerton]             1.7221      0.277      6.213      0
    C(actor_name)[T.Tom Hiddleston]           -0.5398      0.156     -3.466      0
    C(actor_name)[T.Tom Holland]              -0.4950      0.121     -4.095      0
```

```
     C(actor_name)[T.Tye Sheridan]              -0.1266      0.199     -0.636
     C(release_year)[T.2004]                    -1.1611      1.346     -0.862
     C(release_year)[T.2005]                     0.7634      0.790      0.966
     C(release_year)[T.2006]                    -0.9767      1.233     -0.792
     C(release_year)[T.2007]                    -0.6335      0.981     -0.646
     C(release_year)[T.2008]                     0.3939      0.893      0.441
     C(release_year)[T.2009]                     1.2495      0.922      1.356
     C(release_year)[T.2010]                    -0.2065      1.275     -0.162
     C(release_year)[T.2011]                    -0.0612      0.846     -0.072
     C(release_year)[T.2012]                     1.3094      0.932      1.405
     C(release_year)[T.2013]                     0.2939      0.970      0.303
     C(release_year)[T.2014]                    -0.6417      0.976     -0.658
     C(release_year)[T.2015]                     0.7931      1.029      0.771
     C(release_year)[T.2016]                     1.8709      0.991      1.889
     C(release_year)[T.2017]                     0.7094      1.135      0.625
     C(release_year)[T.2018]                     2.0673      0.997      2.073
     C(release_year)[T.2019]                     2.2219      0.896      2.481
     C(release_year)[T.2020]                     1.3749      0.871      1.579
     C(release_year)[T.2021]                     1.6262      0.991      1.642
     C(release_year)[T.2022]                     2.6143      0.834      3.134
     C(release_year)[T.2023]                     0.8428      1.137      0.741
     C(release_year)[T.2024]                     1.6230      0.792      2.050
```

```python
# 3. Pull out the four key coefficients + CIs
vars_to_plot = ['Treatment','film_count','average_rating','count_lead_roles']  #
params = did_model.params[vars_to_plot]
ci_low, ci_high = did_model.conf_int().loc[vars_to_plot].T.values

# 4. Draw the coefficient plot
plt.figure()
plt.errorbar(
    vars_to_plot,
    params.values,
    yerr=[params.values - ci_low, ci_high - params.values],
    fmt='o'
)
plt.axhline(0, linestyle='--')
plt.xticks(rotation=45)
plt.ylabel('Coefficient estimate')
plt.title('DiD Coefficients and 95% CI')
plt.tight_layout()
plt.show()
```



```
df_event = treatment_control.copy()
```