- Exercise: Netflix Viewership
- Name: Barath Anandaraman
- Course: DSC640-T301
- Weeks 1 & 2: Importance of Context in Storytelling
- Date: 03/21/2025

Load necessary packages

In [1]:

```
import pandas as pd
import numpy as np
import plotly.express as px
import pycountry
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.dates as mdates
```

In [49]:

```
# Suppress warnings
import warnings
from openpyxl import Workbook

# Suppress the warning
warnings.filterwarnings("ignore", message="Workbook contains no default style", category=UserWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
```

Load the netflix files into dataframes using pandas

In [3]:

```
countries_df = pd.read_excel("all-weeks-countries-netflix.xlsx")
global_df = pd.read_excel("all-weeks-global-netflix.xlsx")
popular_df = pd.read_excel("most-popular-netflix.xlsx")
```

Function to check for basic issues on the dataframe data

In [4]:

```
def checkIssues(df):
    """ Check for Issues on the dataframe """
    df_state = []
    columns = df.columns
    for i in columns :
        types = df[i].dtypes
        unique_value = df[i].nunique()
        nan_value= df[i].isnull().sum()
        value_count= df[i].isnull().count()
        nan_percentage= round(nan_value/value_count*100,2)
        duplicated= df.duplicated().sum()

        df_state.append ([i , types , unique_value , nan_value, nan_percentage,duplicated])

    df_state = pd.DataFrame(df_state)
    df_state.columns =['Name of column' , 'Types' ,'Unique_data' , 'NAN value', "NAN_percentage","Duplicated"]
    print(df_state)
```

In [5]:
```
# Read countries dataset
countries_df.head()
```

Out[5]:

	country_name	country_iso2	week	category	weekly_rank	show_title	season_title	cumulative_weeks_in_top_10
0	Argentina	AR	2024-04-14	Films	1	The Tearsmith	NaN	2
1	Argentina	AR	2024-04-14	Films	2	Stolen	NaN	1
2	Argentina	AR	2024-04-14	Films	3	Love, Divided	NaN	1
3	Argentina	AR	2024-04-14	Films	4	Woody Woodpecker Goes to Camp	NaN	1
4	Argentina	AR	2024-04-14	Films	5	Rest In Peace	NaN	3

In [6]:
```
# Check for issues visually on countries dataset
checkIssues(countries_df)
```

```
              Name of column   Types  Unique_data  NAN value  NAN_percentage  \
0               country_name  object           94          0            0.00   
1               country_iso2  object           94          0            0.00   
2                       week  object          146          0            0.00   
3                   category  object            2          0            0.00   
4                weekly_rank   int64           10          0            0.00   
5                 show_title  object         7146          0            0.00   
6               season_title  object         2375     140675           51.67   
7  cumulative_weeks_in_top_10   int64          113          0            0.00   

   Duplicated  
0           0  
1           0  
2           0  
3           0  
4           0  
5           0  
6           0  
7           0  
```

In [7]:

```
# Read global dataset
global_df.head()
```

Out[7]:

	week	category	weekly_rank	show_title	season_title	weekly_hours_viewed	runtime	weekly_views	cumulative_weeks_in_top_10
0	2024-04-14	Films (English)	1	What Jennifer Did	NaN	26100000	1.4500	18000000.0	1
1	2024-04-14	Films (English)	2	Woody Woodpecker Goes to Camp	NaN	19600000	1.6667	11800000.0	1
2	2024-04-14	Films (English)	3	Scoop	NaN	14600000	1.7167	8500000.0	2
3	2024-04-14	Films (English)	4	Glass	NaN	11000000	2.1500	5100000.0	2
4	2024-04-14	Films (English)	5	Megan Leavey	NaN	9700000	1.9333	5000000.0	1

In [8]:

```
# Check for issues visually on global dataset
checkIssues(global_df)
```

```
              Name of column    Types  Unique_data  NAN value  \
0                       week   object          146          0   
1                   category   object            4          0   
2                weekly_rank    int64           10          0   
3                 show_title   object         1915          0   
4               season_title   object          835       3026   
5         weekly_hours_viewed    int64         2713          0   
6                    runtime  float64          340       4080   
7               weekly_views  float64          196       4080   
8   cumulative_weeks_in_top_10    int64           30          0   
9         is_staggered_launch     bool            2          0   
10      episode_launch_details   object           72       5768   

    NAN_percentage  Duplicated  
0             0.00           0  
1             0.00           0  
2             0.00           0  
3             0.00           0  
4            51.82           0  
5             0.00           0  
6            69.86           0  
7            69.86           0  
8             0.00           0  
9             0.00           0  
10           98.77           0  
```

In [9]:

```
# Read popular dataset
popular_df.head()
```

Out[9]:

	category	rank	show_title	season_title	hours_viewed_first_91_days	runtime	views_first_91_days
0	Films (English)	1	Red Notice	NaN	454200000	1.9667	230900000
1	Films (English)	2	Don't Look Up	NaN	408600000	2.3833	171400000
2	Films (English)	3	The Adam Project	NaN	281000000	1.7833	157600000
3	Films (English)	4	Bird Box	NaN	325300000	2.0667	157400000
4	Films (English)	5	Leave the World Behind	NaN	339300000	2.3667	143400000

In [10]:

```
# Check for issues visually on popular dataset
checkIssues(popular_df)
```

```
              Name of column    Types  Unique_data  NAN value  \
0                   category   object            4          0
1                       rank    int64           10          0
2                 show_title   object           35          0
3               season_title   object           20         20
4  hours_viewed_first_91_days    int64           40          0
5                    runtime  float64           38          0
6         views_first_91_days    int64           38          0

   NAN_percentage  Duplicated
0             0.0           0
1             0.0           0
2             0.0           0
3            50.0           0
4             0.0           0
5             0.0           0
6             0.0           0
```

Function to merge categories so both can in same format

In [11]:

```
def merge_categories(country_df, globals_df):
    """
    Merges category information from global_df into country_df based on show_title.

    Args:
        country_df (pd.DataFrame): DataFrame with country-specific data.
        global_df (pd.DataFrame): DataFrame with global data and detailed categories.

    Returns:
        pd.DataFrame: Modified country_df with updated category information.
    """

    # Create a mapping dictionary from global_df
    global_category_map = globals_df[['show_title', 'category']].drop_duplicates().set_index('show_title')['category'].

    def map_category(row):
        show_title = row['show_title']
        if show_title in global_category_map:
            return global_category_map[show_title]
        else:
            return row['category'] #if show_title not found in global_category_map, keep original category

    # Apply the mapping to the country_df
    country_df['category'] = country_df.apply(map_category, axis=1)
    return country_df
```

In [12]:

```
# Merge categories and check the output
country_updated = merge_categories(countries_df, global_df)
country_updated.head(5)
```

Out[12]:

	country_name	country_iso2	week	category	weekly_rank	show_title	season_title	cumulative_weeks_in_top_10
0	Argentina	AR	2024-04-14	Films (Non-English)	1	The Tearsmith	NaN	2
1	Argentina	AR	2024-04-14	Films (Non-English)	2	Stolen	NaN	1
2	Argentina	AR	2024-04-14	Films (Non-English)	3	Love, Divided	NaN	1
3	Argentina	AR	2024-04-14	Films (English)	4	Woody Woodpecker Goes to Camp	NaN	1
4	Argentina	AR	2024-04-14	Films (Non-English)	5	Rest In Peace	NaN	3

In [13]:

```
# Check the value counts of merged outout
country_updated['category'].value_counts()
```

Out[13]:

```
category
Films (English)        74948
TV (English)           71971
TV (Non-English)       48092
Films                  31845
Films (Non-English)    30099
TV                     15305
Name: count, dtype: int64
```

In [14]:

```
# Check few merged cases that didnt have mapped values
country_updated[country_updated['category'] == 'TV'].head(5)
```

Out[14]:

	country_name	country_iso2	week	category	weekly_rank	show_title	season_title	cumulative_weeks_in_top_10
79	Argentina	AR	2024-03-24	TV	10	ARA San Juan: The Submarine that Disappeared	ARA San Juan: The Submarine that Disappeared: ...	3
92	Argentina	AR	2024-03-17	TV	3	ARA San Juan: The Submarine that Disappeared	ARA San Juan: The Submarine that Disappeared: ...	2
110	Argentina	AR	2024-03-10	TV	1	ARA San Juan: The Submarine that Disappeared	ARA San Juan: The Submarine that Disappeared: ...	1
299	Argentina	AR	2024-01-07	TV	10	Somewhere Between	Somewhere Between: Limited Series	1
336	Argentina	AR	2023-12-24	TV	7	Yellowjackets	Yellowjackets: Season 1	1

We can understand that the rows that didnt have mapping is most likely local content, can be both English or Non English

In [15]:

```
def calculate_category_percentages(df):
    """
    Calculates the percentage of each category for each country in a DataFrame.

    Args:
        df (pd.DataFrame): DataFrame with country and category information.

    Returns:
        pd.DataFrame: DataFrame with category percentages per country.
    """

    # Calculate the total count of each category for each country
    category_counts = df.groupby(['country_name', 'category']).size().unstack(fill_value=0)

    # Calculate the total count of all categories for each country
    total_counts = category_counts.sum(axis=1)

    # Calculate the percentage of each category
    category_percentages = category_counts.div(total_counts, axis=0) * 100
    category_percentages = category_percentages.round(2)
    # Reset the index to make 'country_name' a column
    category_percentages = category_percentages.reset_index()

    return category_percentages
```

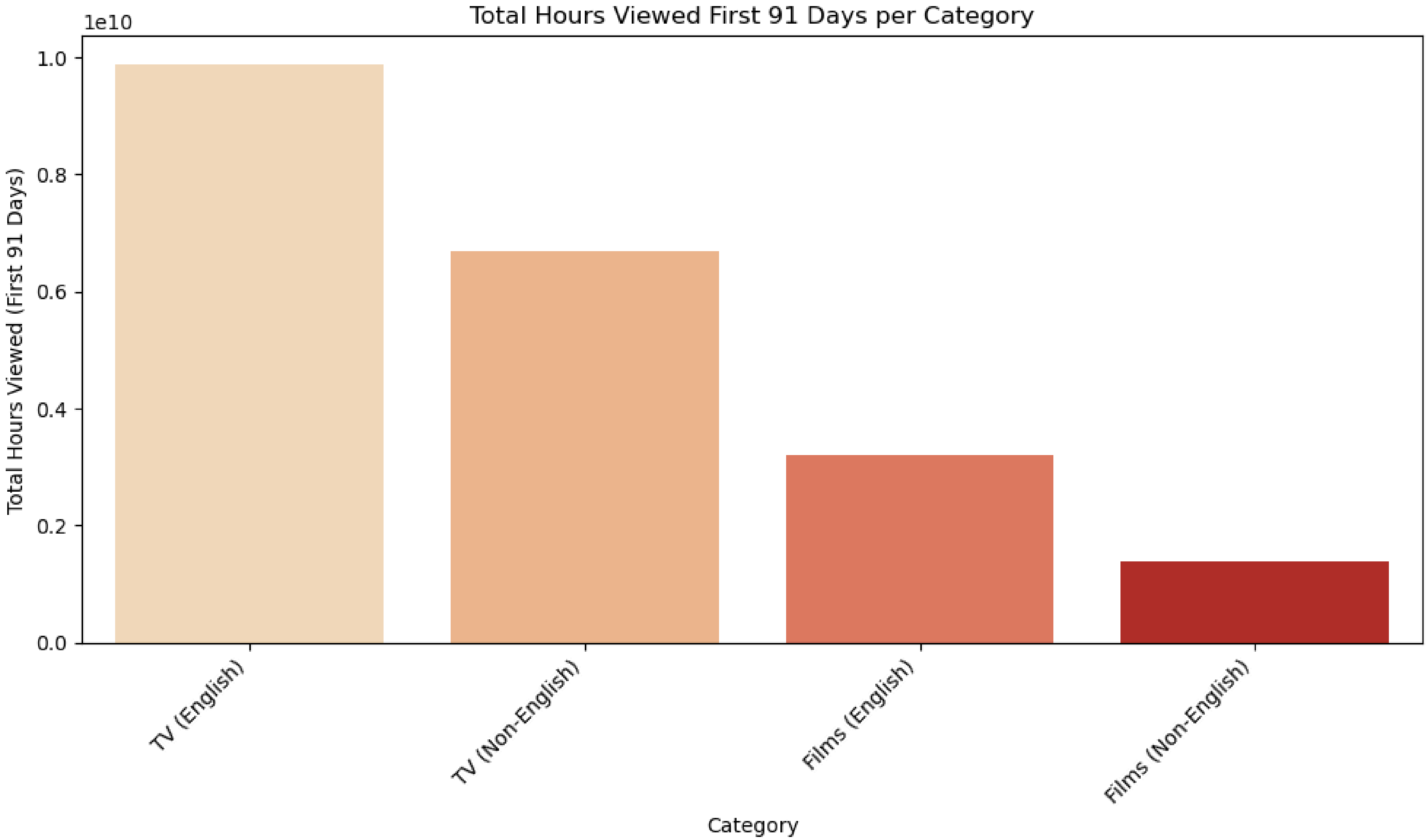
In [16]:
```
category_percentages_df = calculate_category_percentages(country_updated)
print(category_percentages_df)
```

```
category    country_name  Films  Films (English)  Films (Non-English)     TV  \
0              Argentina   6.85            28.46                14.79   2.57
1              Australia  18.25            29.32                 2.60   6.16
2                Austria  12.33            27.77                10.17   4.32
3                Bahamas  12.67            27.26                10.55   5.92
4                Bahrain  11.30            23.77                15.14   4.97
..                   ...    ...              ...                  ...    ...
89        United Kingdom  17.74            29.76                 2.77   7.53
90         United States  10.41            37.02                 2.77   2.81
91               Uruguay   7.12            27.12                15.92   2.91
92             Venezuela   5.68            29.52                15.24   4.55
93               Vietnam  20.31            21.30                 8.49  14.45

category  TV (English)  TV (Non-English)
0                19.32             28.01
1                39.08              4.59
2                33.87             11.54
3                33.29             10.31
4                26.78             18.05
..                 ...               ...
89               38.70              3.49
90               42.36              4.62
91               21.10             25.82
92               15.55             29.45
93                8.94             26.51

[94 rows x 7 columns]
```

Check Total hours viewed in First 91 days

In [17]:
```
# Group by category and sum hours viewed
category_hours = popular_df.groupby('category')['hours_viewed_first_91_days'].sum().reset_index()

# Sort by hours viewed
category_hours = category_hours.sort_values(by='hours_viewed_first_91_days', ascending=False)
```

```python
# Create the bar chart
plt.figure(figsize=(10, 6))
sns.barplot(x='category', y='hours_viewed_first_91_days', data=category_hours, palette='OrRd')
plt.title('Total Hours Viewed First 91 Days per Category')
plt.xlabel('Category')
plt.ylabel('Total Hours Viewed (First 91 Days)')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for readability
plt.tight_layout()
plt.show()
```



It is interesting to see TV Non English shows are having more viewership than English Films

Function to map country code needed for mapping in plotly

In [18]:

```
def get_country_code(country_name):
    """Converts country name to ISO 3166-1 alpha-3 code."""
    try:
        return pycountry.countries.get(name=country_name).alpha_3
    except AttributeError:
        return None  # Handle cases where country name is not found
```

In [19]:

```
# Apply the function to create a 'country_code' column
category_percentages_df['country_code'] = category_percentages_df['country_name'].apply(get_country_code)

# Clean up any rows where country code couldn't be found (optional)
category_percentages_df = category_percentages_df.dropna(subset=['country_code'])
```

In [20]:

```
# Get the category columns dynamically
category_columns = [col for col in category_percentages_df.columns if col not in ['country_name', 'country_code']]

# Reshape the data to long format
category_percentages_long = category_percentages_df.melt(
    id_vars=['country_name', 'country_code'],
    value_vars=category_columns,
    var_name='category',
    value_name='percentage'
)
```

Create dataframe filtered excluding TV English to get local and Non English content

In [21]:

```
# Filter for Non-English TV shows
df_tv = country_updated[(~(country_updated['category'] == 'TV (English)')) & country_updated['category'].str.contains('

# Aggregate cumulative weeks by country
country_weeks = df_tv.groupby('country_name')['cumulative_weeks_in_top_10'].sum().reset_index()
country_weeks.head(10)
```

Out[21]:

	country_name	cumulative_weeks_in_top_10
0	Argentina	7536
1	Australia	805
2	Austria	1209
3	Bahamas	1431
4	Bahrain	1897
5	Bangladesh	4174
6	Belgium	1169
7	Bolivia	12879
8	Brazil	10191
9	Bulgaria	946

Create dataframe filtered excluding TV - Non English to get local and English content

In [22]:

```
df_tv1 = country_updated[(~(country_updated['category'] == 'TV (Non-English)')) & country_updated['category'].str.conta
```

In [23]:

```
# Aggregate cumulative weeks by country
country_weeks1 = df_tv1.groupby('country_name')['cumulative_weeks_in_top_10'].sum().reset_index()
country_weeks1.head(5)
```

Out[23]:

	country_name	cumulative_weeks_in_top_10
0	Argentina	1715
1	Australia	3779
2	Austria	3615
3	Bahamas	3529
4	Bahrain	2657

Add country code column based on country name

In [24]:

```
# Apply the function to create a 'country_code' column
country_weeks['country_code'] = country_weeks['country_name'].apply(get_country_code)
```

```
# Clean up any rows where country code couldn't be found (optional)
country_weeks_df = country_weeks.dropna(subset=['country_code'])
```

In [25]:

```
# Check if country code got added
country_weeks_df.head(2)
```

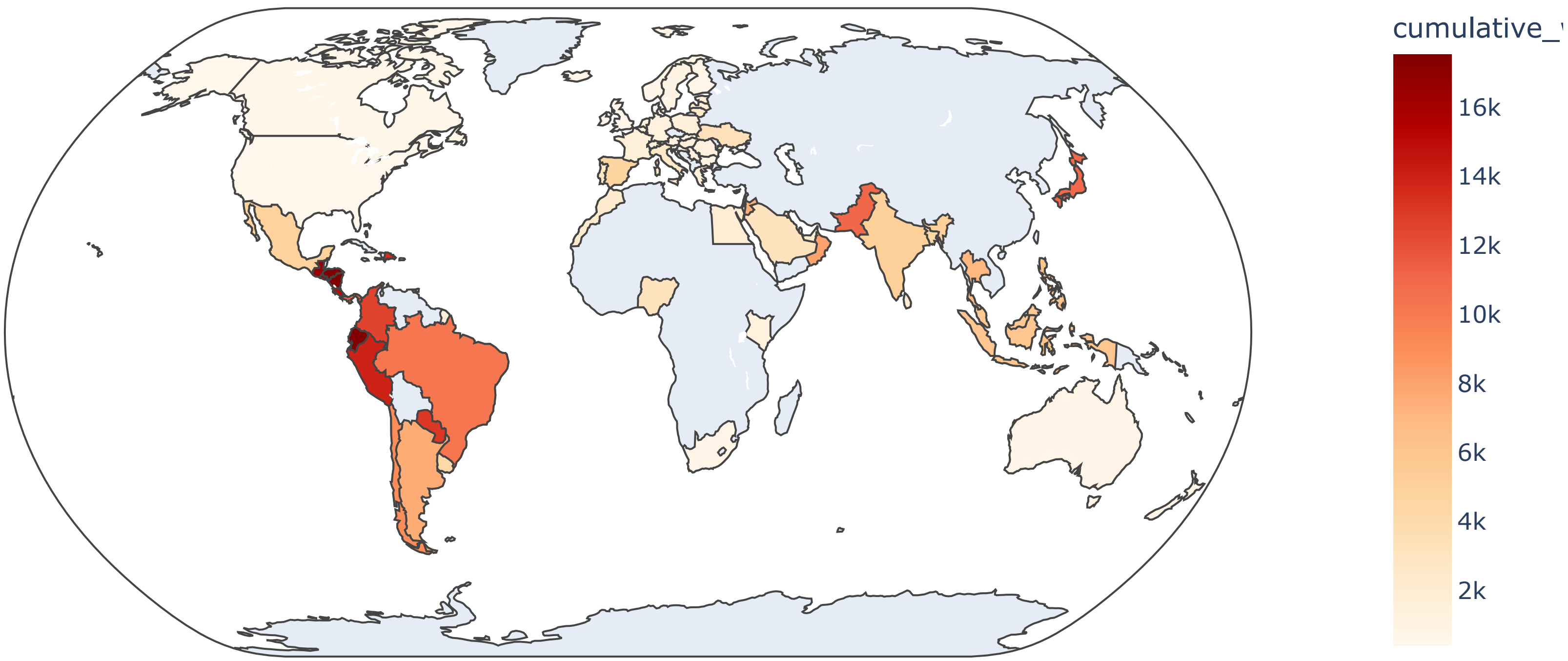
Out[25]:

	country_name	cumulative_weeks_in_top_10	country_code
0	Argentina	7536	ARG
1	Australia	805	AUS

Create a map for the aggregated data based on TV Non English and Local content

In [52]:

```
# Create a choropleth map
fig = px.choropleth(
    country_weeks_df,
    locations="country_code",
    color="cumulative_weeks_in_top_10",  # Choose a category column to visualize
    hover_name="country_name",
    color_continuous_scale=px.colors.sequential.OrRd,
    title="TV-Non English & Local cumulative weeks in Top 10",
)

fig.update_geos(projection_type="natural earth")
fig.show()
```



TV-Non English & Local cumulative weeks in Top 10

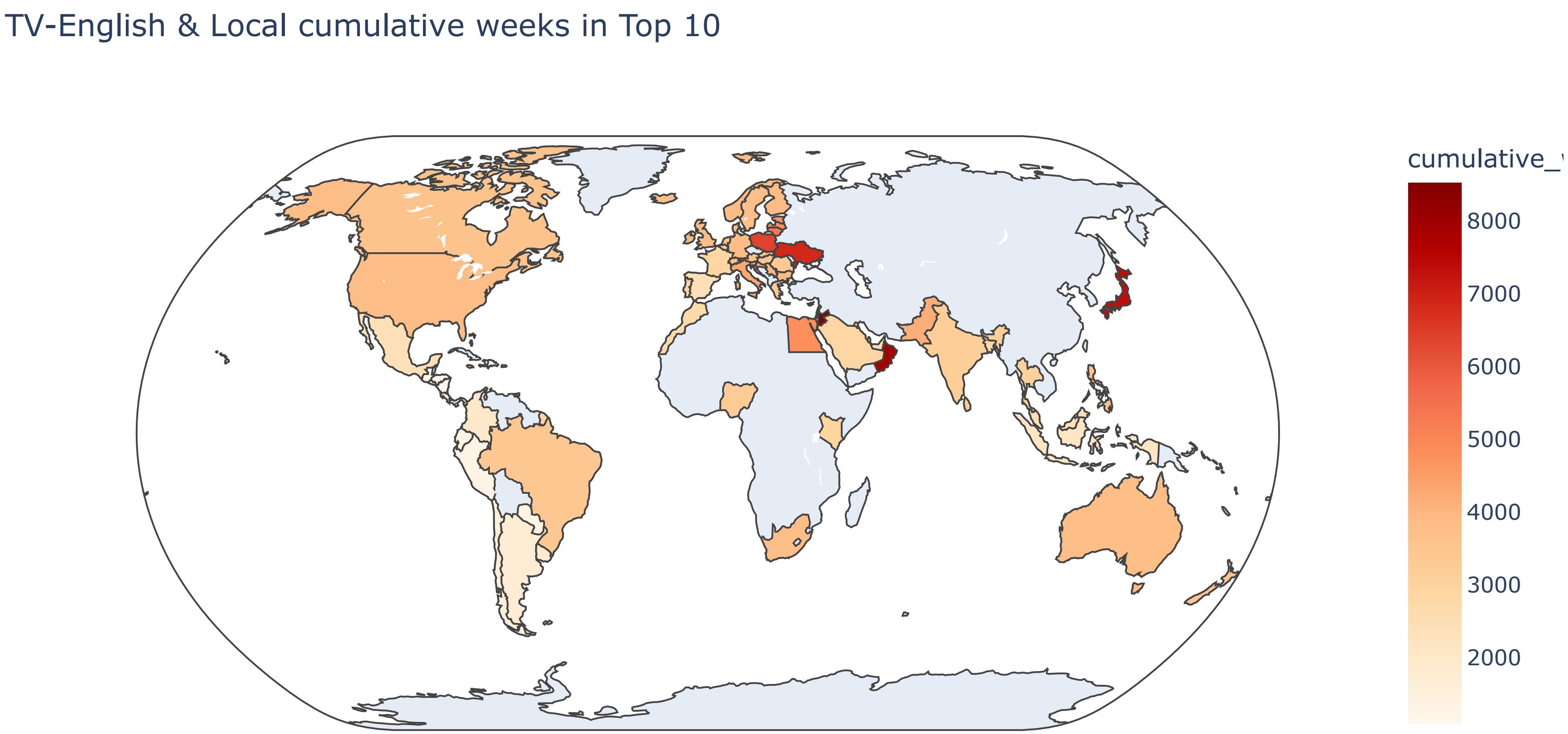
Perform similar operation for TV English and Local content

In [27]:

```
# Apply the function to create a 'country_code' column
country_weeks1['country_code'] = country_weeks1['country_name'].apply(get_country_code)

# Clean up any rows where country code couldn't be found (optional)
country_weeks_df1 = country_weeks1.dropna(subset=['country_code'])
```

In [53]:

```
# Create a choropleth map
fig = px.choropleth(
    country_weeks_df1,
    locations="country_code",
    color="cumulative_weeks_in_top_10",  # Choose a category column to visualize
    hover_name="country_name",
    color_continuous_scale=px.colors.sequential.OrRd,
    title="TV-English & Local cumulative weeks in Top 10",
)

fig.update_geos(projection_type="natural earth")
fig.show()
```



Compare Top and bottom 5 countries watching TV Local & Non English content

In [29]:
```
# Sort by cumulative weeks and get top 5
top_5_countries = country_weeks.sort_values(by='cumulative_weeks_in_top_10', ascending=False).head(5)
```

```
top_5_countries.head()
```

Out[29]:

	country_name	cumulative_weeks_in_top_10	country_code
29	Honduras	17541	HND
57	Nicaragua	17497	NIC
19	Ecuador	17456	ECU
28	Guatemala	16396	GTM
21	El Salvador	16202	SLV

In [30]:

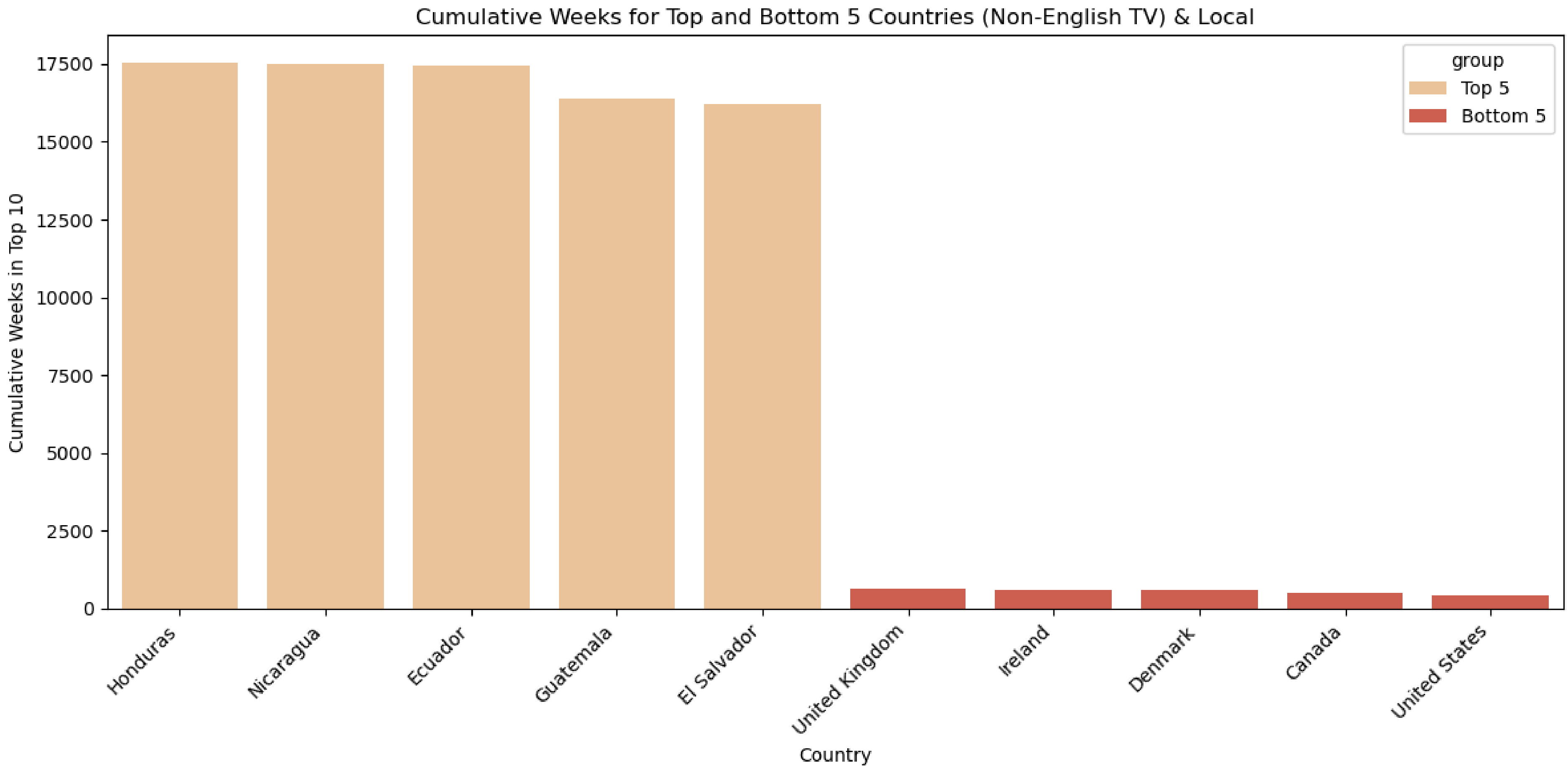
```
bottom_5_countries = country_weeks.sort_values(by='cumulative_weeks_in_top_10', ascending=False).tail(5)
bottom_5_countries.head()
```

Out[30]:

	country_name	cumulative_weeks_in_top_10	country_code
89	United Kingdom	635	GBR
35	Ireland	592	IRL
17	Denmark	584	DNK
10	Canada	523	CAN
90	United States	402	USA

In [31]:

```
# Assuming you have top_5_countries and bottom_5_countries DataFrames

# Add a 'group' column to distinguish top and bottom
top_5_countries['group'] = 'Top 5'
bottom_5_countries['group'] = 'Bottom 5'

# Concatenate the DataFrames
combined_countries = pd.concat([top_5_countries, bottom_5_countries])

# Create the bar plot
plt.figure(figsize=(12, 6))  # Adjust figure size for better readability

sns.barplot(
    x='country_name',
    y='cumulative_weeks_in_top_10',
    hue='group',  # Use 'hue' to differentiate top and bottom
    data=combined_countries,
    palette='OrRd'
)

plt.title('Cumulative Weeks for Top and Bottom 5 Countries (Non-English TV) & Local')
plt.xticks(rotation=45, ha='right')
plt.xlabel('Country')
plt.ylabel('Cumulative Weeks in Top 10')
plt.tight_layout()
plt.show()
```



Non-English and local content is pretty popular and even bottom most countries have those contents on English speaking countries

Using global dataset for Local and Non English TV content

```
In [32]: global_df_1 = global_df[(~(global_df['category'] == 'TV (English)')) & global_df['category'].str.contains('TV')]
```

```
In [33]: # Group by show title and calculate average rank and total hours viewed
         show_performance = global_df_1.groupby('show_title').agg(
             average_rank=('weekly_rank', 'mean'),
             total_hours_viewed=('weekly_hours_viewed', 'sum')
         ).reset_index()
```

```
In [34]: show_performance.head()
```

```
Out[34]:
                show_title  average_rank  total_hours_viewed
   0     42 Days of Darkness          7.00             9920000
   1      6ixtynin9 The Series          8.00             6500000
   2          A Killer Paradox          3.75            85300000
   3            A Model Family          7.00            34960000
   4   A Nearly Normal Family          2.75           113400000
```

```
In [35]: # Filter for shows with average rank <= 5 (arbitrary threshold for "high-performing")
         top_shows = show_performance[show_performance['average_rank'] <= 5].sort_values(by='total_hours_viewed', ascending=Fals
         top_shows.head()
```

```
Out[35]:
                       show_title  average_rank  total_hours_viewed
   240                Squid Game      3.954545          2315500000
    39    Café con aroma de mujer      4.678571           813480000
    90  Extraordinary Attorney Woo      4.000000           662090000
    13          All of Us Are Dead      3.545455           659510000
   263                  The Glory      3.230769           560990000
```

```
In [36]: # Get the detailed weekly data for the top shows
         top_shows_data = global_df_1[global_df_1['show_title'].isin(top_shows['show_title'])]
         top_shows_data.head()
```

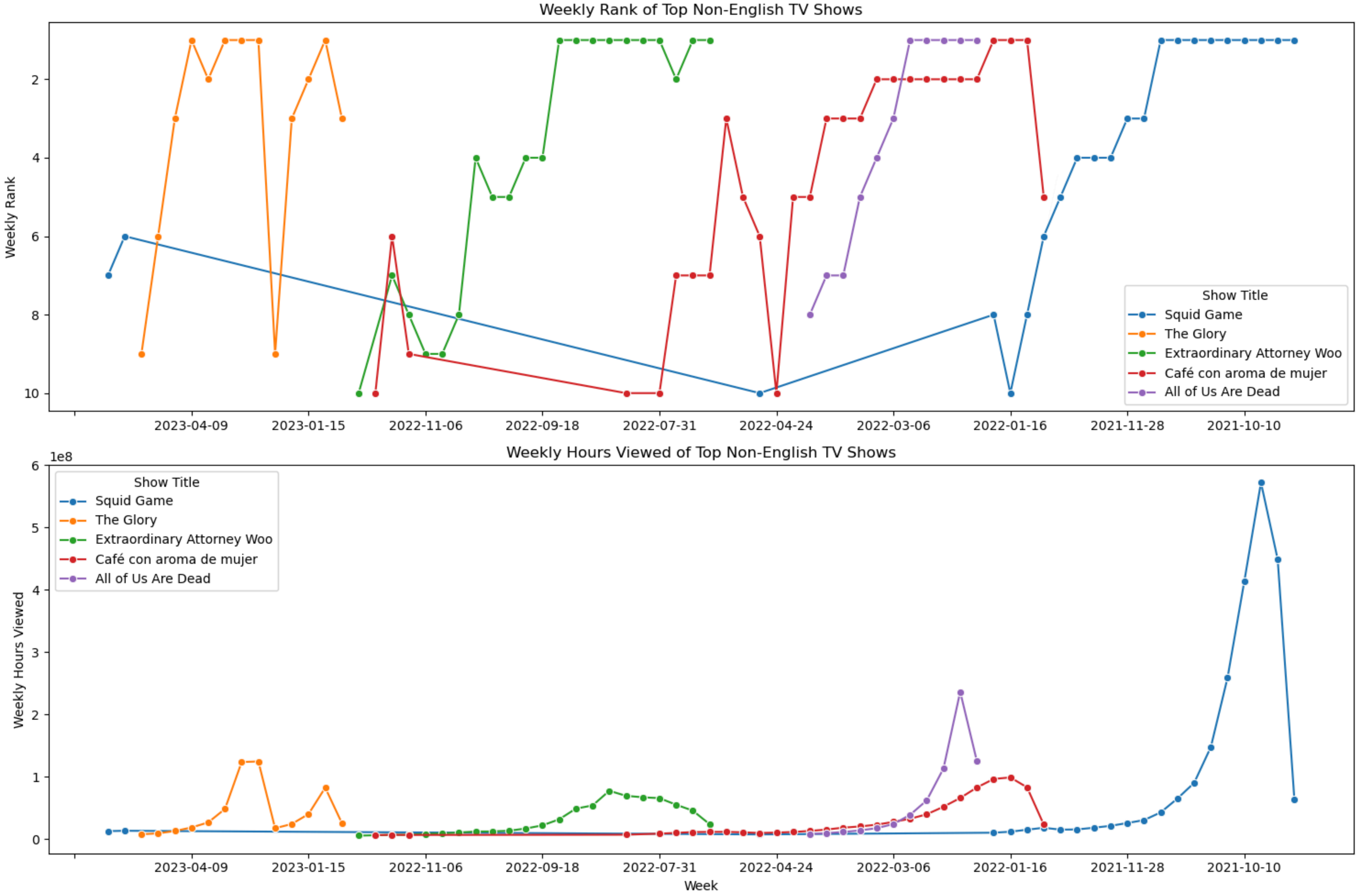
Out[36]:

	week	category	weekly_rank	show_title	season_title	weekly_hours_viewed	runtime	weekly_views	cumulative_weeks_in_top_
796	2023-12-03	TV (Non-English)	7	Squid Game	Squid Game: Season 1	12700000	8.3167	1500000.0	
835	2023-11-26	TV (Non-English)	6	Squid Game	Squid Game: Season 1	13300000	8.3167	1600000.0	
2038	2023-04-30	TV (Non-English)	9	The Glory	The Glory: Season 1	7680000	NaN	NaN	
2075	2023-04-23	TV (Non-English)	6	The Glory	The Glory: Season 1	9160000	NaN	NaN	
2112	2023-04-16	TV (Non-English)	3	The Glory	The Glory: Season 1	12900000	NaN	NaN	

Line chart for weekly Rank and weekly hours viewed

In [37]:

```
# Create line charts for weekly rank and hours viewed
plt.figure(figsize=(15, 10))

# Weekly Rank Chart
plt.subplot(2, 1, 1)  # 2 rows, 1 column, 1st subplot
sns.lineplot(x='week', y='weekly_rank', hue='show_title', data=top_shows_data, marker='o')
plt.title('Weekly Rank of Top Non-English TV Shows')
plt.xlabel(None)
plt.ylabel('Weekly Rank')
plt.gca().invert_yaxis()  # Invert y-axis so lower ranks are at the top
# Set x-axis ticks and labels for time series
plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator(interval=1)) # Show each week
plt.legend(title='Show Title')

# Weekly Hours Viewed Chart
plt.subplot(2, 1, 2)  # 2 rows, 1 column, 2nd subplot
sns.lineplot(x='week', y='weekly_hours_viewed', hue='show_title', data=top_shows_data, marker='o')
plt.title('Weekly Hours Viewed of Top Non-English TV Shows')
plt.xlabel('Week')
plt.ylabel('Weekly Hours Viewed')
plt.legend(title='Show Title')
# Set x-axis ticks and labels for time series
plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator(interval=1)) # Show each week
plt.tight_layout()
plt.show()
```



New shows are having high popularity

Use popular dataset to check runtime vs views in Non English shows

In [38]:

```python
popular_df_1 = popular_df[(~(popular_df['category'] == 'TV (English)')) & popular_df['category'].str.contains('TV')]
```

In [39]:

```python
popular_df_1.sort_values(by='runtime', ascending=False).head(10)
```

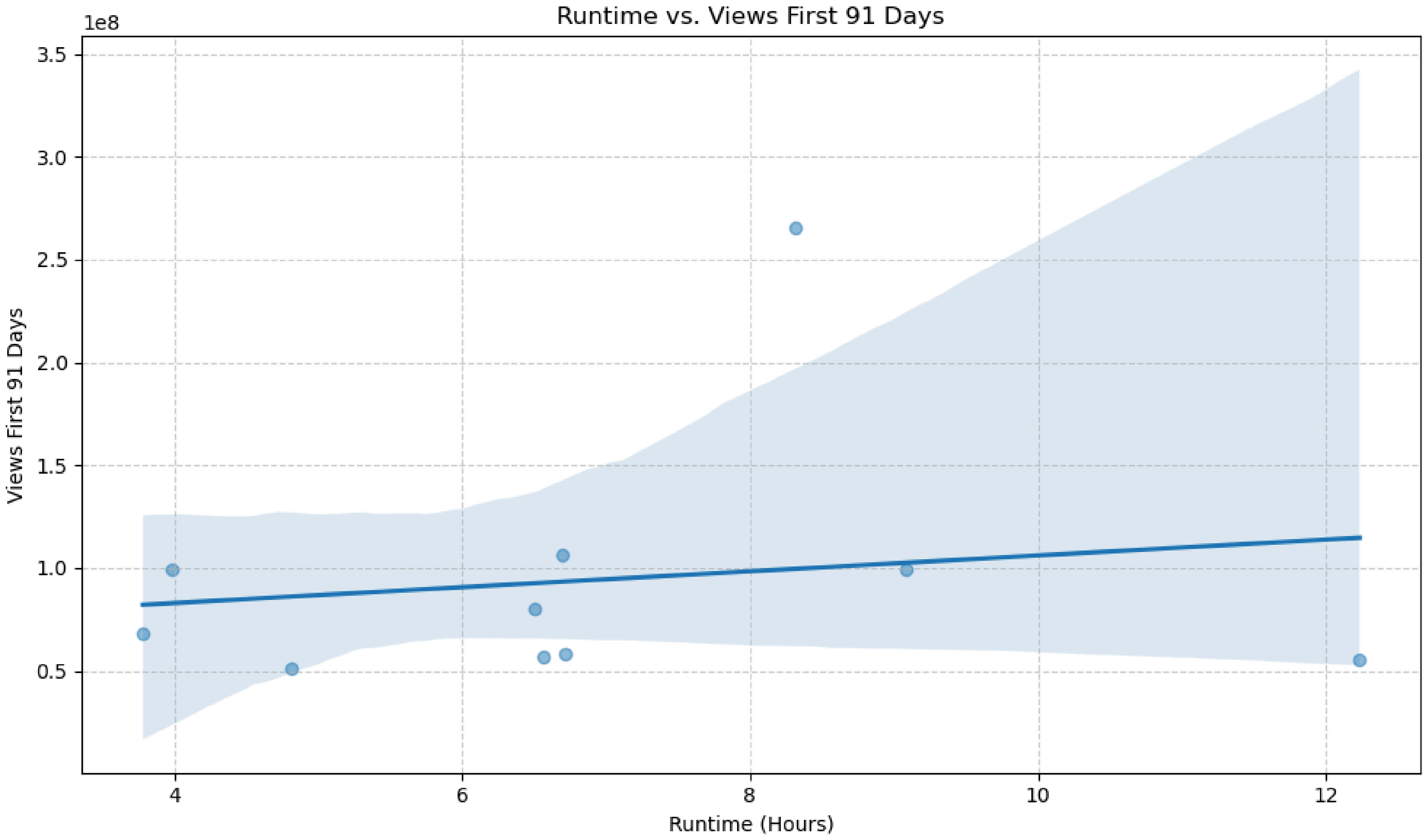
Out[39]:

	category	rank	show_title	season_title	hours_viewed_first_91_days	runtime	views_first_91_days
38	TV (Non-English)	9	All of Us Are Dead	All of Us Are Dead: Season 1	679300000	12.2333	55500000
33	TV (Non-English)	4	Money Heist	Money Heist: Part 5	900700000	9.0833	99200000
30	TV (Non-English)	1	Squid Game	Squid Game: Season 1	2205200000	8.3167	265200000
36	TV (Non-English)	7	Who Killed Sara?	Who Killed Sara?: Season 1	392400000	6.7167	58400000
31	TV (Non-English)	2	Money Heist	Money Heist: Part 4	710200000	6.7000	106000000
37	TV (Non-English)	8	Berlin	Berlin: Season 1	372600000	6.5667	56700000
34	TV (Non-English)	5	Money Heist	Money Heist: Part 3	519800000	6.5000	80000000
39	TV (Non-English)	10	Dear Child	Dear Child: Limited Series	245400000	4.8167	50900000
32	TV (Non-English)	3	Lupin	Lupin: Part 1	396300000	3.9833	99500000
35	TV (Non-English)	6	Lupin	Lupin: Part 2	258900000	3.7833	68400000

Scatterplot with trend line to look at hours and runtime correlation

In [40]:

```python
# Create the scatter plot with a trend line
plt.figure(figsize=(10, 6))
sns.regplot(x='runtime', y='views_first_91_days', data=popular_df_1, scatter_kws={'alpha': 0.5})
plt.title('Runtime vs. Views First 91 Days')
plt.xlabel('Runtime (Hours)')
plt.ylabel('Views First 91 Days')
plt.grid(True, linestyle='--', alpha=0.6)  # Add a grid for better readability
plt.tight_layout()
plt.show()
```



Runtime vs. Views First 91 Days

Sweet spot is between 4-8 hours run time

Clean up isStaggered column from global dataset

In [45]:

```
# Clean up 'isStaggered launch' column
global_df_1 = global_df[ global_df['category'].str.contains('TV')]
global_df_1['is_staggered_launch'] = global_df_1['is_staggered_launch'].fillna('No')
global_df_1['is_staggered_launch'] = global_df_1['is_staggered_launch'].replace({True: 'Yes', False: 'No'})
```

```
/var/folders/07/0p9rkpr50g1fdk4v_c1k0j540000gn/T/ipykernel_7186/435063545.py:3: 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/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy

/var/folders/07/0p9rkpr50g1fdk4v_c1k0j540000gn/T/ipykernel_7186/435063545.py:4: 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/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
```

In [50]:

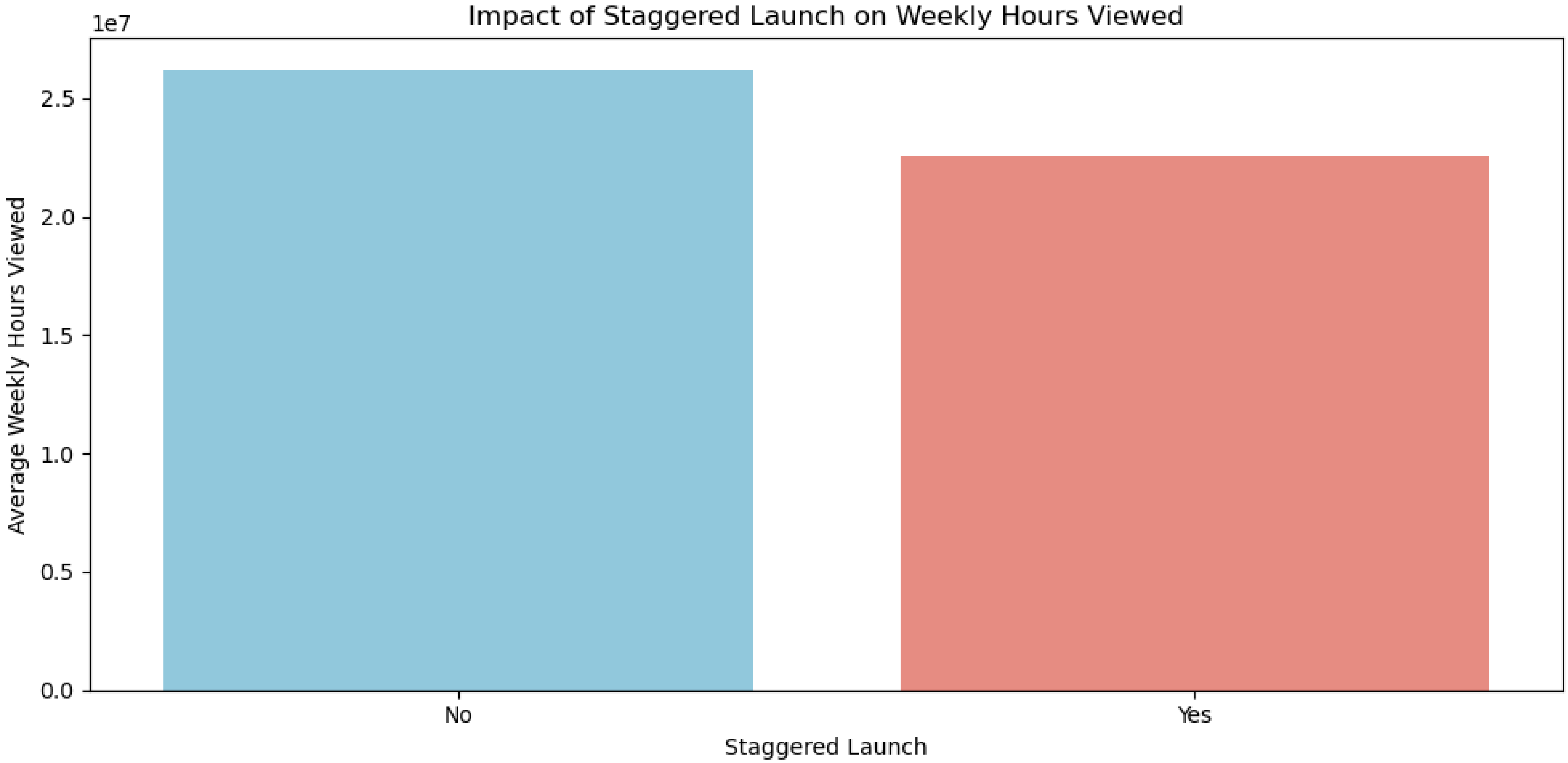
```python
# Group by 'isStaggered launch' and calculate average weekly hours viewed
staggered_impact = global_df_1.groupby('is_staggered_launch')['weekly_hours_viewed'].mean().reset_index()
```

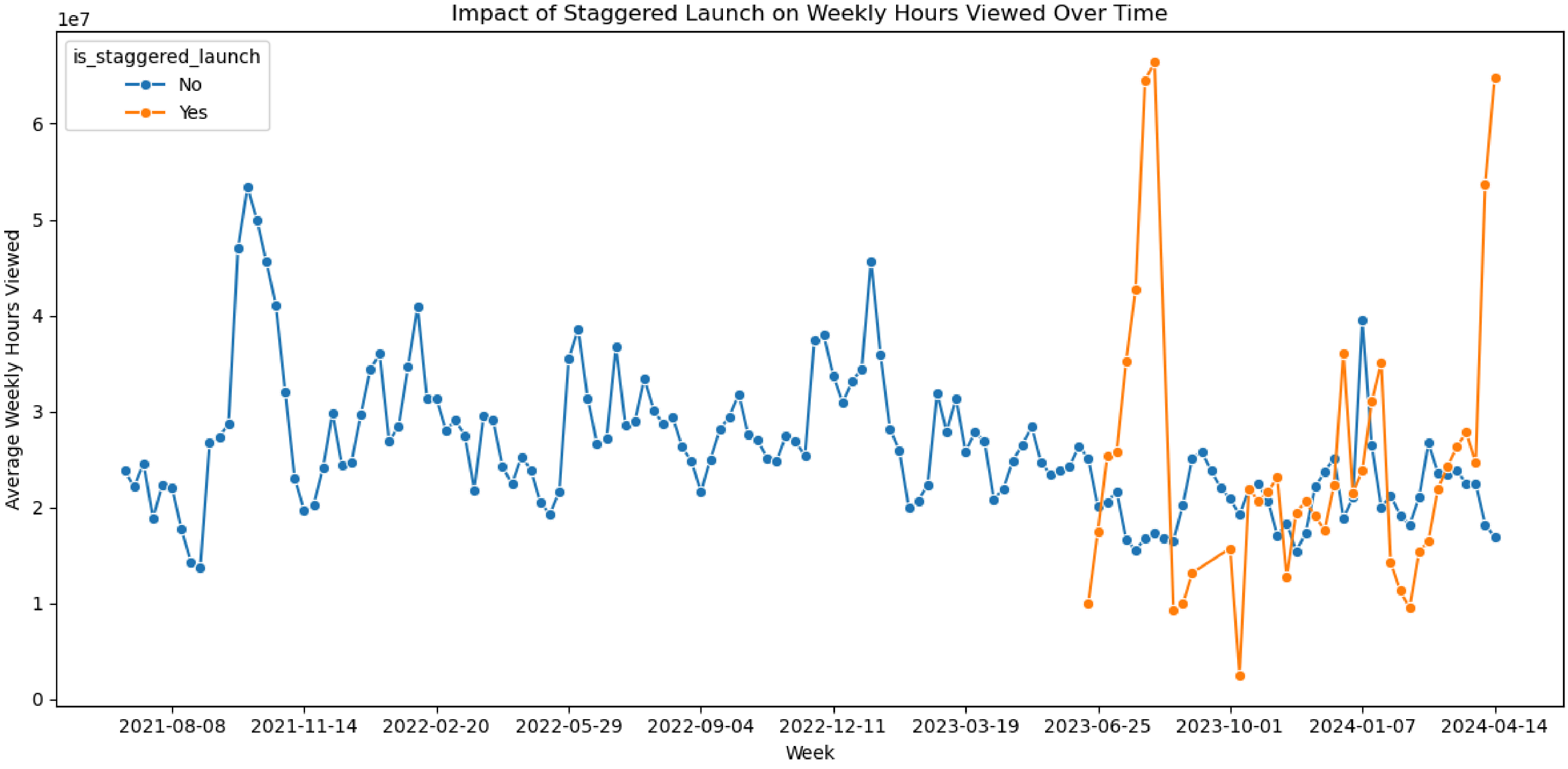
Check for staggering launch of episodes and weekly hours viewed

In [55]:

```python
# Create the side-by-side bar chart
plt.figure(figsize=(10, 5))
sns.barplot(x='is_staggered_launch', y='weekly_hours_viewed', data=staggered_impact, palette=['skyblue', 'salmon'])
plt.title('Impact of Staggered Launch on Weekly Hours Viewed')
plt.xlabel('Staggered Launch')
plt.ylabel('Average Weekly Hours Viewed')
plt.tight_layout()
plt.show()

# Optional: Further analysis by week
# Group by 'isStaggered launch' and week, then calculate average weekly hours viewed
staggered_impact_by_week = global_df_1.groupby(['is_staggered_launch', 'week'])['weekly_hours_viewed'].mean().reset_inc

# Create a line plot to show the impact over time
plt.figure(figsize=(12, 6))
sns.lineplot(x='week', y='weekly_hours_viewed', hue='is_staggered_launch', data=staggered_impact_by_week, marker='o')
plt.title('Impact of Staggered Launch on Weekly Hours Viewed Over Time')
plt.xlabel('Week')
plt.ylabel('Average Weekly Hours Viewed')
plt.tight_layout()
```

```
# Set x-axis ticks and labels for time series
plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator(interval=2)) # Show each week
plt.show()
```





Showing staggering launches have higher peaks than regular launches