Building an Interactive EDA Dashboard

for Movie Dataset

by Jenny Zhu MADS Student @ UMICH

Visualization Technique

Narrative Description of Each Visualization Type

1. **Bar Chart**:
   - A bar chart is used to compare categorical data. Each bar's length represents the value or frequency of the corresponding category.
   - This visualization is helpful when analyzing across different categories (e.g., the frequency of observations per group).
2. **Histogram**:
   - A histogram displays the distribution of numeric data as continuous intervals (bins). The height of each bar represents the count or density of data points within the interval.
   - A histogram allows you to assess the spread, central tendency, skewness, and outliers in numerical data.
3. **Boxplot**:
   - A boxplot summarizes the distribution of data by showing the median, quartiles, and potential outliers in one compact chart. Each "box" represents the interquartile range (IQR).
   - This visualization works well for identifying how data values vary among different groups and for spotting outliers.
4. **Scatterplot**:
   - A scatterplot shows the relationship between two continuous variables by marking data points on an x-y axis.
   - It is useful for identifying correlations, trends, clusters, or outliers between variables.

How These Visualizations Complement Each Other

- Together, these visualizations provide a comprehensive understanding of the data by analyzing different aspects:
  - **Bar Chart**: Works with categorical data to provide summaries and group-level insights.
  - **Histogram**: Provides insight into numerical data distributions without focusing on specific relationships.
  - **Boxplot**: Builds on insights from both histograms and bar charts by summarizing data distributions while emphasizing variability and outliers. It can also emphasize side-by-side comparison among different categories.
  - **Scatterplot**: Complements the other charts by showing relationships between two variables, enabling a deeper trend or pattern analysis.

When Each Visualization Should Be Used

- **Bar Chart**: Use it when comparing frequencies or values across categories with color on the bars showing portion of 'Blockboster', e.g., Genres, MPAA Rating, and Creative Type.
- **Histogram**: Use it for understanding the distribution of continuous data, e.g., Gross Revenue, Runtime, Buget.
- **Boxplot**: Use it when comparing variability, especially across groups, or spotting outliers, e.g., Revenue in different genres.
- **Scatterplot**: Use it to see the relationship between a numerical variable and revenue, with different groups in different colors, e.g., the correlation between Revenue (y-axis) and buget (x-xis) for different genres (color).

Dashboard-specific considerations

Interactivity Options:

- Enable **filtering and selection**: Allow users to interact with widgets or dropdown filters to select specific categories, variables, or ranges realse years for interest.
- Use **tooltips**: Show additional information (e.g., exact values) when a user hovers over specific points on a scatterplot or bars in a bar chart.
- Add **zoom and pan functionality**: Particularly useful for scatterplots and histograms to explore data points in more detail.

Visualization Library

Framework and Libraries Used

1. **Panel**:
   - **What it is**: Panel is an open-source dashboarding library provided by the `HoloViz` ecosystem, designed for creating interactive and highly customizable dashboards. It supports rendering visualizations, widgets, and layouts in Jupyter Notebooks or standalone web applications.
   - **Who created it**: It is developed and maintained by the HoloViz community.
   - **Why it's suitable**: Panel provides an intuitive way to create interactive applications, supports a wide range of plotting libraries, and integrates seamlessly with Jupyter Notebooks. Additionally, its support for widgets makes it a great choice for enabling interactivity.
2. **HvPlot**:
   - **What it is**: HvPlot is a high-level plotting library that integrates Pandas-like APIs for creating easy, declarative visualizations.
   - **Why it's suitable**: HvPlot provides simplified syntax for exploratory data analysis, enabling you to generate bokeh-backed plots directly from a Pandas DataFrame or other data analysis libraries like Dask or Xarray. Its integration with HoloViews improves interactivity and advanced plotting features.
3. **Other Libraries**:
   - **Pandas**: Used for data manipulation and analysis.
   - **NumPy**: For numerical operations and advanced data computations.
   - **Scipy**: Provides powerful statistical analysis functions (e.g., `stats` module for analysis like distributions).

Benefits and Limitations of the Framework

Benefits:

1. **Easy to Use Approach**: Panel and HvPlot is easy to use. For instance, HvPlot allows you to generate visualizations with a single statement (e.g., `df.hvplot()`).
2. **Jupyter Notebook Integration**: Both Panel and HvPlot integrate seamlessly into Jupyter Notebooks. The `pn.extension()` enables displaying Panel objects, widgets, and interactive features directly in notebooks for exploratory or static analysis.
3. **Support for Interactivity**: Panel enables creating rich, interactive widgets such as dropdowns, sliders, and buttons, making it highly suitable for creating dashboards with dynamic data exploration and user inputs experience.
4. **Display Dashboard on a Local Host Webpage**:

   My favorite feature of this framework is that the final dashboard is displayed on a webpage, although it's locally hosted, it's very neat to get everything in one place and easy to share with audience.

Limitations of the Framework:

1. **Complexity for Beginners**: While HvPlot simplifies basic plotting tasks, Panel's layout system and widget bindings can be overwhelming for users new to declarative-style frameworks. Use needs to equip with essential Python programming skills such as creating classes and functions. It will be even more difficult if user wants to create a reusable code snippets.
2. **Performance for Large Data**:
   - Interactivity and responsiveness can degrade when handling very large datasets, that's why I have to sample certain amount of data in first step.
3. **Tooling and Debugging**:
   - Debugging complex interactions between widgets and plots can sometimes be challenging, as all elements are linked to each other tightly. One slight oversight in one variable would cause big impact on final results.

Why I Chose This Framework

Panel is a great tool for building interactive dashboards, especially for small-scale data visualizations. It's easy to use in Jupyter, allowing you to quickly prototype and tweak interactive features without needing a complex web setup. It integrates seamlessly with popular Python libraries like Pandas, HvPlot, NumPy, and SciPy, making it a powerful choice for exploratory data analysis (EDA) and creating rich, interactive plots. As an open-source tool with strong community support, I can get any help with debugging if needed. It also offers a variety of interactive features, such as hover effects, tooltips, legends, and widgets like customized sliders and dropdowns, making it ideal for creating dynamic and engaging dashboards for viewers with any levels of data science knowledge.

Install essential libraries

Copy and paste into a new cell, delete the cell once installed:

```
!pip install panel hvplot pandas numpy vega_datasets
```

In [1]:
```
# Next, import them into this notebook
import panel as pn
import hvplot.pandas
import pandas as pd
import numpy as np
import holoviews as hv
from scipy import stats

# enable Jupyter to display Panel objects and interactive widgets
pn.extension()
```

In [2]:
```
import logging
import param

# Suppress warnings by setting param's logging level to ERROR
logging.getLogger('param').setLevel(logging.ERROR)

import warnings
warnings.simplefilter(action='ignore', category=UserWarning)
```

Demonstration

In this section, I will demonstrate how to build a compelling interactive dashboard to display information and insights for `movies` dataset from `vega_dataset` module step by step:

1. Load, clean and explore the dataset with Pandas.
2. Create individual plots with interactivity
3. Combine plots from step 2 and build a tabbed dashboard

The dataset we are going to use is downloaded from `vega_dataset` module directly accessed through:

```
from vega_datasets import data
df = data.movies()
```

I chose this dataset because it offers a fantastic opportunity to dive into patterns and uncover meaningful insights. Personally, I find it fascinating to explore how different variables interact and what stories they tell. By analyzing and visualizing a mix of numerical variables, we can gain a deeper understanding of key metrics of movie industry and how they are related.

This dataset presents an excellent opportunity to explore patterns and derive valuable insights. By analyzing and visualizing various variables such as:

- **Numerical Variables**: `['Gross Revnue', 'US DVD Sales', 'Production Budget', 'Release Date', 'Running Time', 'Ratings from Different Website']`
- **Categorical Variables**: `['MPAA Rating', 'Major Genre', 'Creative Type']`

This kind of exploration not only satisfies my curiosity but also has real-world applications—helping businesses identify early signs of churn and make smarter, data-driven decisions.

Load, Clean, and Display Summary Stats

In [3]:
```
from vega_datasets import data
df = data.movies()
```

In [4]:
```
# I developed a reusable function to load, process, and summarize a cleaned dataset.
def process_data(df, dropna=True, dropna_threshold=0.01, col_to_drop=None):
```

```
    """
    Processes and summarizes a  DataFrame. The input data should be mostly cleaned, with minor missing values.

    Parameters:
    - df (DataFrame): The input DataFrame to process.
    - dropna (bool): Whether to drop rows with NaN values below the threshold (default: True).
    - dropna_threshold (float): The percentage threshold for dropping rows with NaN values (default: 0.01).
    - col_to_drop (list): List of columns to drop from the DataFrame if they are irrelevant (optional).

    Returns:
    - tuple: A tuple containing:
        - Cleaned DataFrame
        - List of numeric columns
        - List of categorical columns
        - Data summary as a dictionary
    """

    # Display the first few rows of the DataFrame for an initial overview
    print("Let's take a quick look at our data:")
    display(df.head(2))

    # Step 2: Drop specified columns if provided
    if col_to_drop:
        print(f"\n1. Dropping specified columns: {col_to_drop}")
        df = df.drop(columns=col_to_drop)  # Drop irrelevant columns
        print(f"Remaining columns: {df.columns.tolist()}")

    # Step 3: Check and handle NaN values
    print("\n2. Checking and handling NaN values...")
    rows_with_nulls = df[df.isnull().any(axis=1)]  # Identify rows with NaN values
    num_rows_with_nulls = len(rows_with_nulls)
    print(f"Number of rows with NaN values: {num_rows_with_nulls}")

    # Drop rows with NaN values if they are below the threshold
    if dropna and num_rows_with_nulls < dropna_threshold * len(df):
        df = df.dropna()
        print(f"Dropped {num_rows_with_nulls} rows with NaN values.")
    else:
        print(f"Retained rows with NaN values as they exceed the threshold of {dropna_threshold * 100}%.")

    # Remove underscores from column names and replace them with spaces
    df.columns = df.columns.str.strip().str.replace('_', ' ', regex=False)
    print(f"\n3. Cleaned column names: {df.columns.tolist()}")

    # Convert 'Release_Date' to a timestamp format
    df['Release Date'] = pd.to_datetime(df['Release Date'], errors='coerce')
    # Extract only the year from the 'Release_Date' column
    df['Release Date'] = df['Release Date'].dt.year


    # Step 5: Summarize column types, missing values, and unique entries
    numeric_cols = list(df.select_dtypes(include=[np.number]).columns)  # Identify numeric columns
    categorical_cols = list(df.select_dtypes(exclude=[np.number]).columns)  # Identify categorical columns

    # Mannually Remove specific columns from the list of categorical columns
    categorical_cols = [col for col in categorical_cols if col not in ['Title', 'Distributor', 'Director']]


    print(f"\n4. The DataFrame shape is {df.shape}.")
    print(f"Numeric columns: {numeric_cols}")
    print(f"Categorical columns: {categorical_cols}")

    # add one column to categorize the revenue into two categories Yes and No for 'Blockbuster'
    df = df.assign(Blockbuster=np.where(df['Worldwide Gross']>=  df['Worldwide Gross'].quantile(0.90), 'Yes', 'No'))

    # review again
    display(df.head(2))

    print("\n5. Generating detailed data summary...")

    # Summarize all columns with their data type and unique value count
    data_summary = {
        col: {
            'type': str(df[col].dtype),
            'unique_values': len(df[col].unique())
        } for col in df.columns
    }

    # Add descriptive statistics for numeric columns
    for col in numeric_cols:
        data_summary[col].update({
            'mean': df[col].mean(),
            'median': df[col].median(),
            'std': df[col].std(),
            'min': df[col].min(),
            'max': df[col].max(),
            'range': df[col].max() - df[col].min()
        })

    # Convert the summary dictionary to a DataFrame for better visualization
    summary_df = pd.DataFrame(data_summary).T
    display(summary_df)

    # Step 6: Display unique values for each categorical column
    print("\n6. Unique values in categorical columns:")
    for col in categorical_cols:
        print(f"Column '{col}': {df[col].unique()}")

    return df, numeric_cols, categorical_cols, data_summary
```

In [5]:
```
# plug in our data path to function above, unpack processed df, numeric_cols and categorical_cols, and add a ";" at end) to prevent displaying returned stuff
df, numeric_cols, categorical_cols, data_summary = process_data(df, col_to_drop='Source');
```

```
Let's take a quick look at our data:
```

	Title	US_Gross	Worldwide_Gross	US_DVD_Sales	Production_Budget	Release_Date	MPAA_Rating	Running_Time_min	Distributor	Source	Major_Genre	Creative_Type	Director	Rotten_
0	The Land Girls	146083.0	146083.0	NaN	8000000.0	Jun 12 1998	R	NaN	Gramercy	None	None	None	None	
1	First Love, Last Rites	10876.0	10876.0	NaN	300000.0	Aug 07 1998	R	NaN	Strand	None	Drama	None	None	

```
1. Dropping specified columns: Source
Remaining columns: ['Title', 'US_Gross', 'Worldwide_Gross', 'US_DVD_Sales', 'Production_Budget', 'Release_Date', 'MPAA_Rating', 'Running_Time_min', 'Distributor', 'Major_Ge
nre', 'Creative_Type', 'Director', 'Rotten_Tomatoes_Rating', 'IMDB_Rating', 'IMDB_Votes']

2. Checking and handling NaN values...
Number of rows with NaN values: 3026
Retained rows with NaN values as they exceed the threshold of 1.0%.

3. Cleaned column names: ['Title', 'US Gross', 'Worldwide Gross', 'US DVD Sales', 'Production Budget', 'Release Date', 'MPAA Rating', 'Running Time min', 'Distributor', 'Ma
jor Genre', 'Creative Type', 'Director', 'Rotten Tomatoes Rating', 'IMDB Rating', 'IMDB Votes']

4. The DataFrame shape is (3201, 15).
Numeric columns: ['US Gross', 'Worldwide Gross', 'US DVD Sales', 'Production Budget', 'Release Date', 'Running Time min', 'Rotten Tomatoes Rating', 'IMDB Rating', 'IMDB Vot
es']
Categorical columns: ['MPAA Rating', 'Major Genre', 'Creative Type']
```

	Title	US Gross	Worldwide Gross	US DVD Sales	Production Budget	Release Date	MPAA Rating	Running Time min	Distributor	Major Genre	Creative Type	Director	Rotten Tomatoes Rating	IMDB Rating	IMDB Votes	Blockbuster
0	The Land Girls	146083.0	146083.0	NaN	8000000.0	1998	R	NaN	Gramercy	None	None	None	NaN	6.1	1071.0	No
1	First Love, Last Rites	10876.0	10876.0	NaN	300000.0	1998	R	NaN	Strand	Drama	None	None	NaN	6.9	207.0	No

```
5. Generating detailed data summary...
```

	type	unique_values	mean	median	std	min	max	range
Title	object	3177	NaN	NaN	NaN	NaN	NaN	NaN
US Gross	float64	3061	44002085.163745	22019465.5	62555311.390662	0.0	760167650.0	760167650.0
Worldwide Gross	float64	3075	85343400.141515	31168926.5	149947342.885362	0.0	2767891499.0	2767891499.0
US DVD Sales	float64	565	34901546.817376	20331557.5	45895121.596701	618454.0	352582053.0	351963599.0
Production Budget	float64	382	31069171.448438	20000000.0	35585913.444644	218.0	300000000.0	299999782.0
Release Date	int32	91	1998.37707	2001.0	11.932124	1928	2046	118
MPAA Rating	object	8	NaN	NaN	NaN	NaN	NaN	NaN
Running Time min	float64	110	110.193548	107.0	20.171014	46.0	222.0	176.0
Distributor	object	175	NaN	NaN	NaN	NaN	NaN	NaN
Major Genre	object	13	NaN	NaN	NaN	NaN	NaN	NaN
Creative Type	object	10	NaN	NaN	NaN	NaN	NaN	NaN
Director	object	551	NaN	NaN	NaN	NaN	NaN	NaN
Rotten Tomatoes Rating	float64	101	54.336924	55.0	28.076593	1.0	100.0	99.0
IMDB Rating	float64	78	6.283467	6.4	1.25229	1.4	9.2	7.8
IMDB Votes	float64	2840	29908.644578	15106.0	44937.582335	18.0	519541.0	519523.0
Blockbuster	object	2	NaN	NaN	NaN	NaN	NaN	NaN

```
6. Unique values in categorical columns:
Column 'MPAA Rating': ['R' None 'PG' 'Not Rated' 'PG-13' 'G' 'NC-17' 'Open']
Column 'Major Genre': [None 'Drama' 'Comedy' 'Musical' 'Thriller/Suspense' 'Adventure' 'Action'
 'Romantic Comedy' 'Horror' 'Western' 'Documentary' 'Black Comedy'
 'Concert/Performance']
Column 'Creative Type': [None 'Contemporary Fiction' 'Science Fiction' 'Historical Fiction'
 'Fantasy' 'Dramatization' 'Factual' 'Super Hero'
 'Multiple Creative Types' 'Kids Fiction']
```

Building Individual Dashboard Components

Create a Control Panel

In [6]:

```python
def create_control_panel(var=True, group=False, range_slider_col=None, filter_col=None,
                         var_default="Variable", group_default="Group By", range_name="Range Filter", filter_name="Filter By"):
    """
    Function to create control panel elements with optional variable selector, grouping selector,
    range slider components, and a filter selector for categorical columns.

    Parameters:
    - var (bool): Whether to create a variable selector. Default is True.
    - group (bool): Whether to create a grouping selector. Default is False.
    - range_slider_col: One numerical column of choice to create a range slider. Default is None.
    - filter_col (str): Column name to create a filter for specific values (e.g., 'Churn'). Default is None.
    - var_default (str): Default name for the variable selector. Default is "Variable".
    - group_default (str): Default name for the grouping selector. Default is "Group By".
    - range_name (str): Name for the range slider. Default is "Range Filter".
    - filter_name (str): Name for the filter selector. Default is "Filter By".

    Returns:
    - select_var: The variable selector widget.
    - select_group: The grouping selector widget.
    - slider: The range slider widget (if created).
```

```
    - filter_select: The filter selector widget (if created).
    - controls (pn.Column): A Panel Column layout of the created widgets.
    """
    widgets = []
    numeric_cols.remove(range_slider_col) # Remove specified columns from numeric_cols

    # Create variable selector if 'var' is True
    if var:
        select_var = pn.widgets.Select(
            options=numeric_cols,  # Options are the numeric columns
            name=f"{var_default}: Choose the variable to analyze",  # Auto-generate name
            value=numeric_cols[0],  # Default to the first value in numeric_cols
            description=f"Choose the {var_default} to analyze"
        )
        widgets.append(select_var)

    # Create grouping selector if 'group' is True
    if group:
        select_group = pn.widgets.Select(
            options=categorical_cols,  # Filter categorical columns
            name=group_default,  # Default or user-specified name
            value=categorical_cols[0],  # Default to the first value in categorical_cols
            description=f"Choose the column to {group_default.lower()} by"
        )
        widgets.append(select_group)

    # Create range slider if 'range_slider_cols' is provided
    if range_slider_col:
        slider = pn.widgets.RangeSlider(
            name=f"{range_name}: {range_slider_col}",  # Auto-generate name with the column
            start=df[range_slider_col].min(),
            end=df[range_slider_col].max(),
            value=(df[range_slider_col].min(), df[range_slider_col].max()),  # Default range matches column range
            step=1,
            format='0[.]0'
        )
        widgets.append(slider)

    # Create filter selector for a specific column if 'filter_col' is provided
    if filter_col:
        filter_select = pn.widgets.Select(
            options=['Yes', 'No'],  # Unique values in the specified column
            name=f"{filter_name}: {filter_col}",  # Auto-generate name with the column
            value='Yes',  # Set default to the first value in the column
            description=f"{filter_name} column: {filter_col}"
        )
        widgets.append(filter_select)

    # Create layout with all widgets
    controls = pn.Column(
        '## Dashboard Controls',
        *widgets,  # Unpack widgets in the column
        sizing_mode='stretch_width'
    )

    return range_slider_col, filter_col, select_var, select_group, slider, filter_select, controls  # Add filter_select to the returned values
```

In [7]:
```
# Create a control panel with all options enabled
range_slider_col, filter_col, select_var, select_group, slider, filter_select, controls = create_control_panel(
    var=True,
    group=True,
    filter_col='Blockbuster', # we are particularly interested in how this variable affecting others
    range_slider_col='Release Date', # we are particularly interested in how this variable affecting others
    var_default="Numerical Variable",
    group_default="Grouping Category",
    range_name="Filter by"
)

# preview the control features we just created
controls
```

Out[7]:

Dashboard Controls

Numerical Variable: Choose the variable to analyze

US Gross

Grouping Category

MPAA Rating

Filter by: Release Date: 1928 .. 2046

Filter By: Blockbuster

Yes

Barchart

In [8]:
```
@pn.depends(select_group, slider)
def barchart(select_group, slider):

    filtered_df = df[
        (df[range_slider_col] >= slider[0]) &
        (df[range_slider_col] <= slider[1])
    ]

    # Create barchart
    bar_chart = filtered_df.groupby([select_group, filter_col]).size().unstack().hvplot.bar(
        x=select_group,
        y=['Yes', 'No'],
        stacked=True,
        title=f'{filter_col} by {select_group} in {slider[0]}-{slider[1]}',
```

```
        ylabel='Count',  # Update x-axis label to reflect the count
        xlabel=select_group,  # Update y-axis label to reflect the grouping variable
        legend='top'
    ).opts(xrotation=45)

    return bar_chart

barchart_layout = pn.Column(
    pn.pane.Markdown("### Interactive Bar Chart (Demo Only, Do Not Click)"),
    select_group,
    slider,
    pn.panel(barchart),
    sizing_mode="stretch_both"
)

# Display the dashboard
barchart_layout.servable()
```

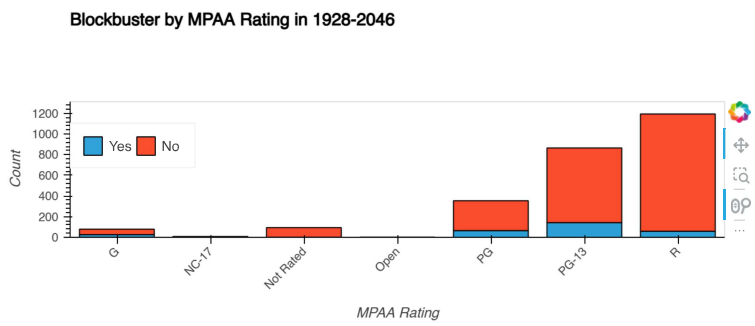
Out[8]:

Interactive Bar Chart (Demo Only, Do Not Click)

Grouping Category

MPAA Rating

Filter by: Release Date: 1928 .. 2046



Barchart for Ranking

One more barchart to show rankings of movie by selected metrics, e.g. ranking of movies by `US revenue`.

In [9]:

```
@pn.depends(select_var, slider)
def barchart_rank(select_var, slider):

    filtered_df = df[
        (pd.to_numeric(df[range_slider_col], errors='coerce') >= float(slider[0])) &
        (pd.to_numeric(df[range_slider_col], errors='coerce') <= float(slider[1]))
    ]

    # Convert select_var column to numeric and drop NaNs
    filtered_df[select_var] = pd.to_numeric(filtered_df[select_var], errors='coerce')
    top_movies = filtered_df.dropna(subset=[select_var]).sort_values(by=select_var, ascending=False)[:10]

    # Create bar chart
    barchart_rank = top_movies.sort_values(by=select_var, ascending=True).hvplot.barh(
        x='Title',
        y=select_var,
        title=f'Top 20 Movies by {select_var} in {slider[0]}-{slider[1]}',
        ylabel=select_var,
        xlabel='Movie Title',

    )

    return barchart_rank

barchart_rank_layout = pn.Column(
    pn.pane.Markdown("### Interactive Bar Chart (Demo Only, Do Not Click)"),
    select_var,
    slider,
    pn.panel(barchart_rank),
    sizing_mode="stretch_both"
)

# Display the dashboard
barchart_rank_layout.servable()
```

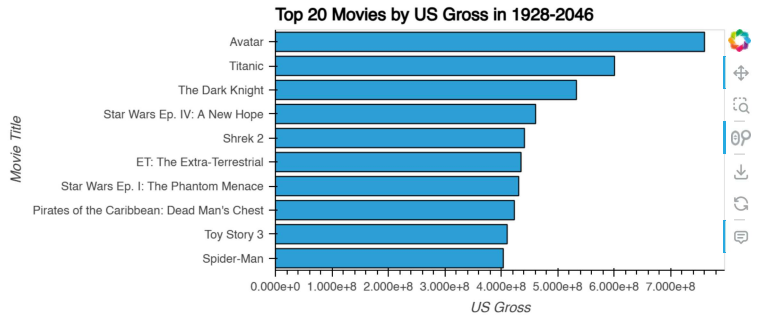
Out[9]:

Interactive Bar Chart (Demo Only, Do Not Click)

Numerical Variable: Choose the variable to analyze

US Gross

Filter by: Release Date: 1928 .. 2046



Histogram

In [10]:

```python
# @pn.depends tells Panel which widgets should trigger updates
@pn.depends(select_var, slider, filter_select)
def histogram_plot(select_var, slider, filter_select):
    """
    Creates an interactive histogram for selected variable with density curves.

    Args:
        select_var (str): The variable to plot
        select_group (str): The grouping variable

    Returns:
        hvplot: Interactive histogram plot
    """

    # filter data based on parameters
    selected_df = df[df[filter_col] == filter_select]

    filtered_df = selected_df[
        (selected_df[range_slider_col] >= slider[0]) &
        (selected_df[range_slider_col] <= slider[1])
    ]

    # add density curve
    x_min = filtered_df[select_var].min()
    x_max = filtered_df[select_var].max()
    sw = np.linspace(x_min, x_max, 1000)
    fit = stats.norm.pdf(sw, np.mean(filtered_df[select_var]), np.std(filtered_df[select_var]))
    bin_width = (x_max - x_min) / 20
    fit_scaled = fit * len(filtered_df) * bin_width

    density_curve = hv.Curve((sw, fit_scaled)).opts(
    line_width=2,
    color='red'
)

    # Create the plot with some customization
    histogram = filtered_df.hvplot.hist(
        y=select_var,
        bins=20,
        height=300,
        alpha=0.5,
        title=f'Histogram for {select_var}',
        xlabel=select_var,
        ylabel='Count',
        **{'responsive': True,
           'legend_position': 'right'}
    )

    return density_curve * histogram

hist_layout = pn.Column(
    pn.pane.Markdown("### Interactive Historgram (Demo Only, Do Not Click)"),
    select_var, select_group, slider, filter_select,
    pn.panel(histogram_plot),
    sizing_mode="stretch_both"
)

# Display the dashboard
hist_layout.servable()
```

Out[10]:

Interactive Historgram (Demo Only, Do Not Click)

Numerical Variable: Choose the variable to analyze

US Gross

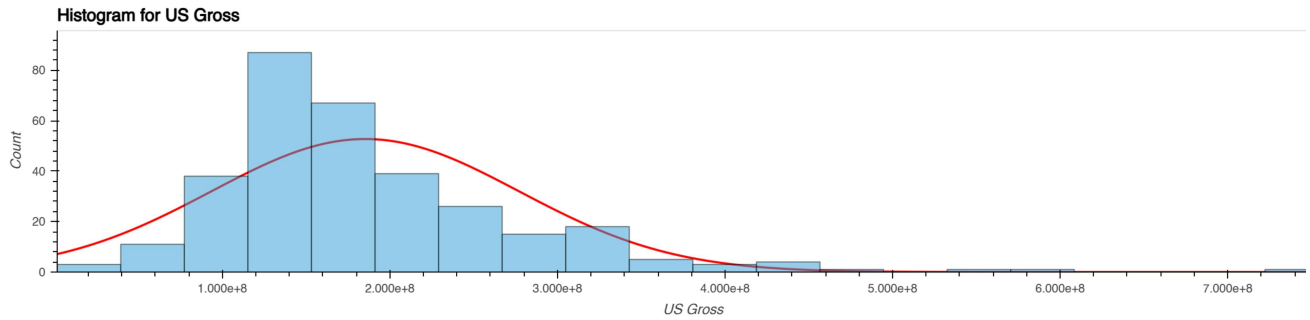
Grouping Category

MPAA Rating

Filter by: Release Date: 1928 .. 2046

Filter By: Blockbuster

Yes



Boxplot

In [11]:

```
@pn.depends(select_var, select_group, slider, filter_select)
def box_plot(select_var, select_group, slider, filter_select):
    """Creates a box plot for selected variables and range, if provided. It also includes hover tooltips for more interactivity.

    Args:
        select_var (str): Variable to plot on y-axis
        select_group (str): Grouping variable for colors

    Returns:
        hvplot: Interactive box plot
    """

    # filter data based on parameters
    selected_df = df[df[filter_col] == filter_select]

    filtered_df = selected_df[
        (selected_df[range_slider_col] >= slider[0]) &
        (selected_df[range_slider_col] <= slider[1])
    ].copy() # Create a copy to avoid SettingWithCopyWarning


    # get statistics for hover tooltips
    stats = {
        group: {
            'median': filtered_df[filtered_df[select_group]==group][select_var].median(),
            'mean': filtered_df[filtered_df[select_group]==group][select_var].mean(),
            'std': filtered_df[filtered_df[select_group]==group][select_var].std()
        } for group in filtered_df[select_group].unique()
    }

    # Create the box plot
    plot = filtered_df.hvplot.box(
        y=select_var,
        by=select_group,
        height=300,
        whisker_color='black',
        title=f'Boxplot for {select_var} by {select_group}',

        # Customize appearance
        box_alpha=0.7,
        outlier_alpha=0.7,
        width=400,
        legend='top',

        # Add statistical hover texts
        tools=['hover'],
        tooltips=[
            ('Group', '@{' + select_group + '}'),
            ('Value', '@{' + select_var + '}{0.00}'),
            ('Count', '@count'),
            ('Median', '@median{0.00}')
        ]
    )

    return plot

boxplot_layout = pn.Column(
    pn.pane.Markdown("### Interactive Boxplot (Demo Only, Do Not Click)"),
    select_var, select_group, slider, filter_select,
    pn.panel(box_plot),
    sizing_mode="stretch_both"
)

# Display the dashboard
boxplot_layout.servable()
```

Out[11]:

Interactive Boxplot (Demo Only, Do Not Click)

Numerical Variable: Choose the variable to analyze

US Gross

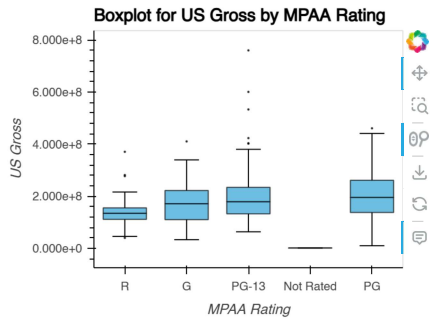
Grouping Category

MPAA Rating

Filter by: Release Date: 1928 .. 2046

Filter By: Blockbuster

Yes



Scatterplot

In [12]:

```
@pn.depends(select_var, slider, filter_select, select_group)
def create_scatter(x_var, slider, filter_select, group_var):
    """
    Creates an interactive scatter plot with points grouped by a specified column.

    Args:
        x_var (str): The variable to use on the x-axis.
        slider (RangeSlider): RangeSlider object to filter data by a numeric range.
        filter_select (str): Column value used to filter rows (e.g., 'Yes' for column 'Churn').
        group_var (str): Column name to group data points by (e.g., 'Gender').

    Returns:
        hvPlot: An interactive scatter plot, grouped by the specified column with colors, with hover tools and interactivity.
    """

    # Remove 'Worldwide Gross' from the options
    filtered_numeric_cols = [col for col in numeric_cols if col != 'Worldwide Gross']

    # Create the select_var widget with the filtered options
    select_var = pn.widgets.Select(
    options=filtered_numeric_cols,  # Use the filtered list of numeric columns
    name="Select Variable",
    value=filtered_numeric_cols[0] if filtered_numeric_cols else None  # Default to the first option if available
    )

    # Filter data based on parameters
    selected_df = df[df[filter_col] == filter_select]

    filtered_df = selected_df[
        (selected_df[range_slider_col] >= slider[0]) &
        (selected_df[range_slider_col] <= slider[1])
    ]

    # Determine the y-variable
    y_var = 'Worldwide Gross'

    # Identify unique groups
    groups = filtered_df[group_var].unique()

    combined = None

    for i, g in enumerate(groups):
        group_data = filtered_df[filtered_df[group_var] == g]

        # Introduce jitter to reduce overlapping points
        group_data[x_var] += np.random.uniform(-0.5, 0.5, size=len(group_data))
        group_data[y_var] += np.random.uniform(-0.5, 0.5, size=len(group_data))

        # Create scatter plot for this group with transparency
        scatter = group_data.hvplot.scatter(
            x=x_var,
            y=y_var,
            alpha=0.6,
            label=str(g),
            size=5
        )

        # Combine with previous groups
        if combined is None:
            combined = scatter
        else:
            combined = combined * scatter

    # Add options to the combined plot
    plot = combined.opts(
        width=600,
        height=400,
```

```
        title=f'Relationship between {x_var} and {y_var}\n(grouped by {group_var})',
        tools=['hover', 'box_zoom', 'reset'],
        show_grid=True,
        toolbar='above'
    )

    return plot

scatter_layout = pn.Column(
    pn.pane.Markdown("### Interactive Scartterplot (Demo Only, Do Not Click)"),
    select_var, select_group, filter_select,
    pn.panel(create_scatter),
    sizing_mode="stretch_both"
)

# Display the dashboard
scatter_layout.servable()
```

Out[12]:

Interactive Scartterplot (Demo Only, Do Not Click)

Numerical Variable: Choose the variable to analyze

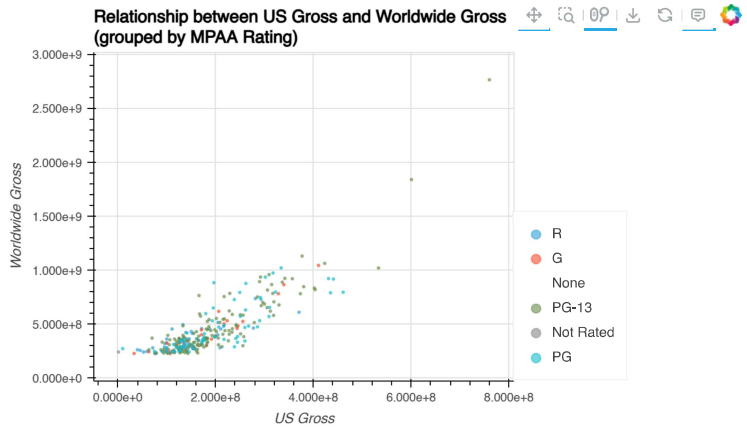
US Gross

Grouping Category

MPAA Rating

Filter By: Blockbuster

Yes



Build a Tabbed Dashboard by Combining All Visualizations

In [13]:

```
pn.extension()

def create_dashboard(widgets, plots):
    # Extract widgets
    select_var, select_group, slider, filter_select = widgets

    # Create separate control layouts for each tab
    # General Tab: Only select_group and slider
    general_controls = pn.Column(
        select_group,
        slider,
        sizing_mode="stretch_width"
    )

    # Distributions and Correlations Tabs: All widgets
    other_controls = pn.Column(
        *widgets,  # Include all widgets
        sizing_mode="stretch_width"
    )

    # Create specific widgets for barchart_rank
    barchart_rank_controls = pn.Column(
        select_var,  # Widget for selecting variable for barchart_rank
        slider,      # Slider for barchart_rank
        sizing_mode="stretch_width"
    )

    # Create tabs
    tabs = pn.Tabs(
        ('Basic', pn.Column(
            general_controls,  # Only select_group and slider
            pn.Row(plots['barchart'], sizing_mode='stretch_both'),
            sizing_mode='stretch_both'
        )),
        ('Ranking', pn.Column(
            barchart_rank_controls,  # Only select_group and slider
            pn.Row(plots['barchart_rank'], sizing_mode='stretch_both'),
            sizing_mode='stretch_both'
        )),
        ('Distribution', pn.Column(
            other_controls,  # All widgets
            pn.Row(plots['boxplot'], plots['histogram'], sizing_mode='stretch_both'),
            sizing_mode='stretch_both'
        )),
        ('Correlation', pn.Row(
            other_controls,  # All widgets
```

```
            plots['scatter'],
            sizing_mode='stretch_both'
        )),
        ('Statistics', pn.Column(
            # No widgets for stats tab
            plots['stats'],
            sizing_mode='stretch_both'
        )),
        sizing_mode='stretch_both'
    )

    main_layout = pn.Column(tabs, sizing_mode='stretch_both').servable()

    template = pn.template.VanillaTemplate(
        title="Interactive EDA Dashboard",
        sidebar=[],
        main=[main_layout],
    )

    return template

# Initialize and display the dashboard
dashboard = create_dashboard(
    widgets=[select_var, select_group, slider, filter_select],
    plots={
        'barchart': barchart,
        'barchart_rank': barchart_rank,
        'histogram': histogram_plot,
        'boxplot': box_plot,
        'scatter': create_scatter,
        'stats': pd.DataFrame(data_summary)
    }
)

dashboard.show()
```

```
Launching server at http://localhost:39731
```

Out[13]: <panel.io.server.Server at 0x73de2cbf1e70>