Task1 : Movie Genre Classification

Background:

In the realm of movie analysis, determining a film's genre based on its plot summary or textual information is a fundamental challenge. Accurately classifying movies into genres provides valuable insights for recommendation systems, content tagging, and audience targeting. Traditional genre classification methods often rely on manual tagging, which is time-consuming and subjective. Leveraging machine learning techniques can automate this process, making it more efficient and objective.

Problem Statement:

The problem at hand is movie genre classification, where the objective is to develop a machine learning model capable of predicting a movie's genre based on its plot summary or textual data. This task is a classic example of text classification in the field of natural language processing. The challenge lies in extracting relevant features from textual information and training a model that can generalize well to accurately categorize movies into different genres.

Approach:

To tackle this problem, we will employ text processing techniques such as TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings. TF-IDF captures the importance of words in a document relative to a collection of documents, while word embeddings like Word2Vec or GloVe represent words in a continuous vector space. These techniques transform textual data into numerical features suitable for machine learning models.

We will utilize classification algorithms like Naive Bayes, Logistic Regression, or Support Vector Machines (SVM) to train our model. Naive Bayes is based on probabilistic principles, while Logistic Regression and SVM are linear classifiers that work well with high-dimensional data.

Approach & Process:

1. Data Gathering

2. Data Cleaning and Preprocessing

3. Data Visualization and Exploration

4. Feature Enhancement and Refinement

Import necessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from wordcloud import WordCloud
```

In [2]:
```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[2]: True

In [3]:
```
import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")
```

1. Data Gathering:

In the creation of our Movie Genre classifier, we acquired our dataset from Kaggle, a renowned platform for datasets. The dataset was meticulously curated from the Internet Movie Database (IMDb), a trusted online repository offering extensive insights into movies, TV shows, cast, crew, ratings, and reviews.

Comprising a rich array of textual content, including plot summaries and pertinent movie information, this dataset forms the foundational material for training our machine learning model. Its diverse textual data will be harnessed to teach the model the intricate patterns necessary for predicting a movie's genre.

In [4]:
```
# Read the txt files using pandas

train_data = pd.read_csv("./Genre Classification Dataset/train_data.txt", delimiter=':::' ,header = None ,
test_data  = pd.read_csv("./Genre Classification Dataset/test_data.txt", delimiter=':::' ,header = None ,e

test_data_solution  = pd.read_csv("./Genre Classification Dataset/test_data_solution.txt", delimiter=':::'
```

In [5]:
```
## View train data
print("shape",train_data.shape)
train_data.head()
```

```
shape (54214, 4)
```

Out[5]:

	0	1	2	3
0	1	Oscar et la dame rose (2009)	drama	Listening in to a conversation between his do...
1	2	Cupid (1997)	thriller	A brother and sister with a past incestuous r...
2	3	Young, Wild and Wonderful (1980)	adult	As the bus empties the students for their fie...
3	4	The Secret Sin (1915)	drama	To help their unemployed father make ends mee...
4	5	The Unrecovered (2007)	drama	The film's title refers not only to the un-re...

The code snippet displays the shape of the train_data DataFrame (54214 rows and 4 columns) and the first few rows of the dataset.

In [6]:
```
## View the test solution data
print("shape",test_data_solution.shape)
test_data_solution.head()
```

```
shape (54200, 4)
```

Out[6]:

	0	1	2	3
0	1	Edgar's Lunch (1998)	thriller	L.R. Brane loves his life - his car, his apar...
1	2	La guerra de papá (1977)	comedy	Spain, March 1964: Quico is a very naughty ch...
2	3	Off the Beaten Track (2010)	documentary	One year in the life of Albin and his family ...
3	4	Meu Amigo Hindu (2015)	drama	His father has died, he hasn't spoken with hi...
4	5	Er nu zhai (1955)	drama	Before he was known internationally as a mart...

The code snippet shows the shape of the test_data_solution DataFrame (54200 rows and 4 columns) and displays the first few rows of the dataset.

In [7]:
```
## We will concat the test and train file

df = pd.concat((train_data ,test_data_solution))
df.columns = ["id" ,"Title","Genre","Description"]
df.head()
```

Out[7]:

	id	Title	Genre	Description
0	1	Oscar et la dame rose (2009)	drama	Listening in to a conversation between his do...
1	2	Cupid (1997)	thriller	A brother and sister with a past incestuous r...
2	3	Young, Wild and Wonderful (1980)	adult	As the bus empties the students for their fie...
3	4	The Secret Sin (1915)	drama	To help their unemployed father make ends mee...
4	5	The Unrecovered (2007)	drama	The film's title refers not only to the un-re...

In [8]:
```
## Check the size
df.shape
```

Out[8]:
```
(108414, 4)
```

The code concatenates the train_data and test_data_solution DataFrames, renaming their columns, resulting in a merged DataFrame df with 108,414 rows and 4 columns, containing ID, Title, Genre, and Description information.

2. Data Cleaning and Preprocessing

In the Data Cleaning and Preprocessing step, the dataset undergoes essential operations:

1. **Removing Duplicates:** Identical entries are eliminated, ensuring each data point is unique.
2. **Removing NaN Rows and Columns:** Rows and columns with missing values are deleted, ensuring data integrity.
3. **Preprocessing the Data:** Textual data is standardized (lowercasing, removing special characters) for consistency, aiding machine learning analysis.

In [9]:
```
## Check for Duplicates and Remove them
df.duplicated().sum() ## Will give us a number of duplicates
```

Out[9]:
```
0
```

In [10]:
```
df.drop_duplicates(inplace = True)  ## Will drops any duplicates
```

In [11]:
```
## Check for nan values

df.isna().sum()  # Will check for any duplicates
```

Out[11]:
```
id             0
Title          0
Genre          0
Description    0
dtype: int64
```

In [12]:
```
df.dropna( inplace = True ) ## Will drop any nan containing row if exists
```

The code `df.isna().sum()` simply counts and displays the number of missing values (NaN) in each column of the dataset (`id` , `Title` , `Genre` , `Description`). In this case, all columns have `0` missing values, indicating there are no NaN values present, ensuring the dataset is complete and ready for analysis.

In [13]:
```
## function to preprocess the data
stopword = set(stopwords.words('english'))

def preprocessing(text):
    # Convert text to lowercase
    text = text.lower()

    # Remove punctuation using regular expressions
    text = re.sub(r'[^\w\s]', '', text)

    # Remove specific characters #, @, and $
    text = re.sub(r'[#@\$]', '', text)

    # tokenize and convert to list
    tokens = word_tokenize(text)

    ## Lemmatize it
    lemmatizer  = WordNetLemmatizer()

    ## lemmatize each token
    text = [lemmatizer.lemmatize(token) for token in tokens]

    text = [word for word in text if word not in stopword]


    return " ".join(text)
```

The `preprocessing` function first converts the input text to lowercase for consistency. It then removes all punctuation marks and specific characters (#, @, $) using regular expressions, ensuring only words and spaces remain. Next, it tokenizes the text into words and lemmatizes each word, reducing them to their base forms. Finally, it removes common English stopwords (like "and" or "the") to focus on meaningful words. The processed words are joined back into a string, which is returned for further analysis, providing a clean and standardized text ready for natural language processing tasks.

In [14]:
```
## Create list of words in discription column
df["Despcription_clean"] =  df["Description"].apply(preprocessing)
```

In [15]:
```
df.head()
```

Out[15]:

	id	Title	Genre	Description	Despcription_clean
0	1	Oscar et la dame rose (2009)	drama	Listening in to a conversation between his do...	listening conversation doctor parent 10yearold...
1	2	Cupid (1997)	thriller	A brother and sister with a past incestuous r...	brother sister past incestuous relationship cu...
2	3	Young, Wild and Wonderful (1980)	adult	As the bus empties the students for their fie...	bus empty student field trip museum natural hi...
3	4	The Secret Sin (1915)	drama	To help their unemployed father make ends mee...	help unemployed father make end meet edith twi...
4	5	The Unrecovered (2007)	drama	The film's title refers not only to the un-re...	film title refers unrecovered body ground zero...

3. Data Visualization and Exploration

In this step, we'll analyze the dataset using histograms and bar charts to understand numerical and categorical distributions. Additionally, we'll create word clouds for each genre, visually representing the most common words in their descriptions. These visualizations provide both quantitative and qualitative insights, aiding in a comprehensive exploration of the dataset.

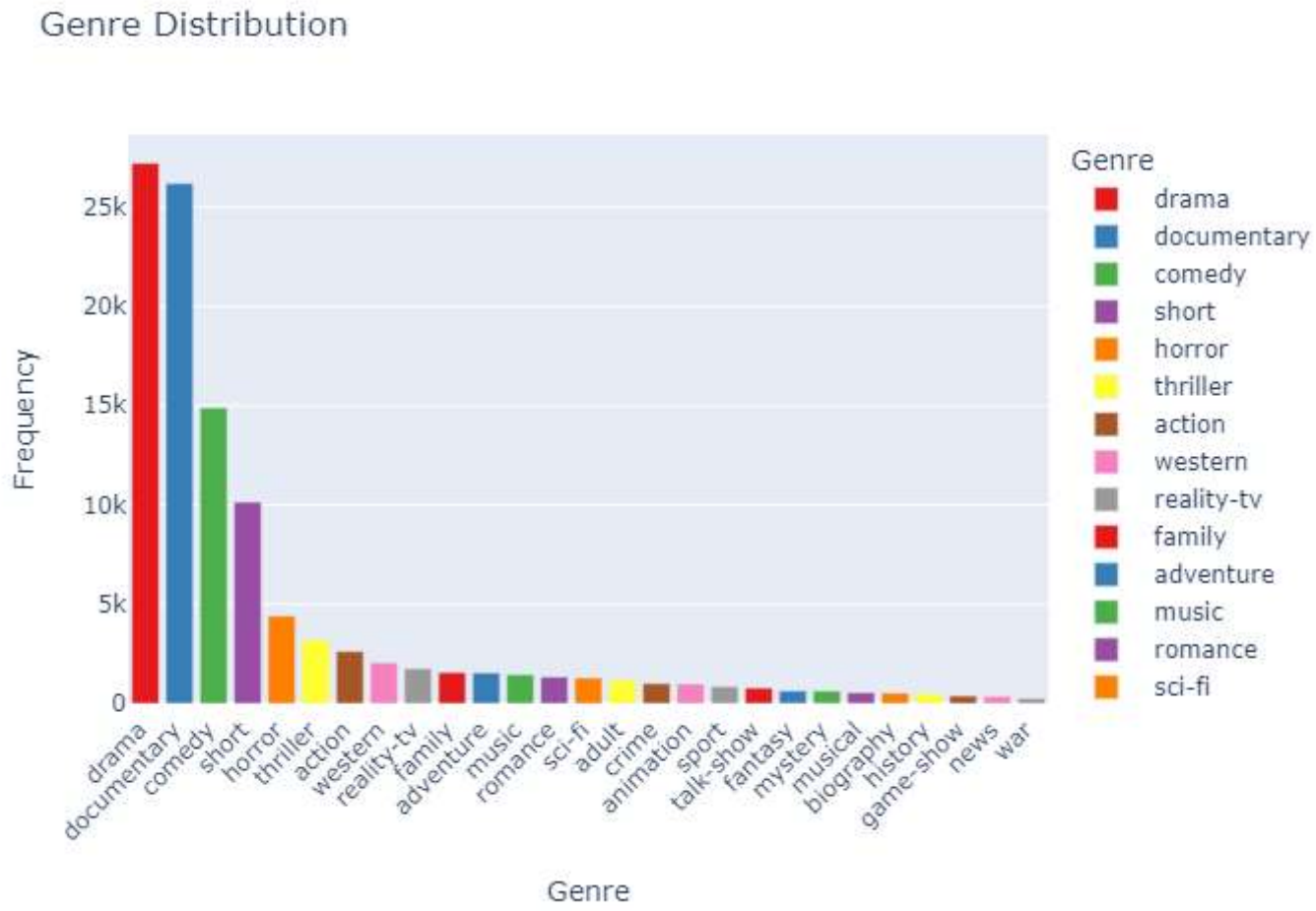
In [16]:
```
## Shows us the label counts
df["Genre"].value_counts()
```

Out[16]:
```
Genre
 drama           27225
 documentary     26192
 comedy          14893
 short           10145
 horror           4408
 thriller         3181
 action           2629
 western          2064
 reality-tv       1767
 family           1567
 adventure        1550
 music            1462
 romance          1344
 sci-fi           1293
 adult            1180
 crime            1010
 animation         996
 sport             863
 talk-show         782
 fantasy           645
 mystery           637
 musical           553
 biography         529
 history           486
 game-show         387
 news              362
 war               264
Name: count, dtype: int64
```

In [17]:

```
import plotly.express as px
import plotly.io as pio
from IPython.display import Image

# Count the frequency of each genre
genre_counts = df["Genre"].value_counts()

# Create a histogram using Plotly Express
fig = px.bar(genre_counts, x=genre_counts.index, y=genre_counts.values,
             labels={'x': 'Genres', 'y': 'Frequency'},
             title='Genre Distribution',
             color=genre_counts.index,
             color_discrete_sequence=px.colors.qualitative.Set1)

fig.update_layout(xaxis_tickangle=-45)

# Save the plot as a static image (PNG format)
pio.write_image(fig, 'genre_distribution.png', format='png')

# Show the static image in Jupyter Notebook

Image('genre_distribution.png')
```

Out[17]:



In [18]:

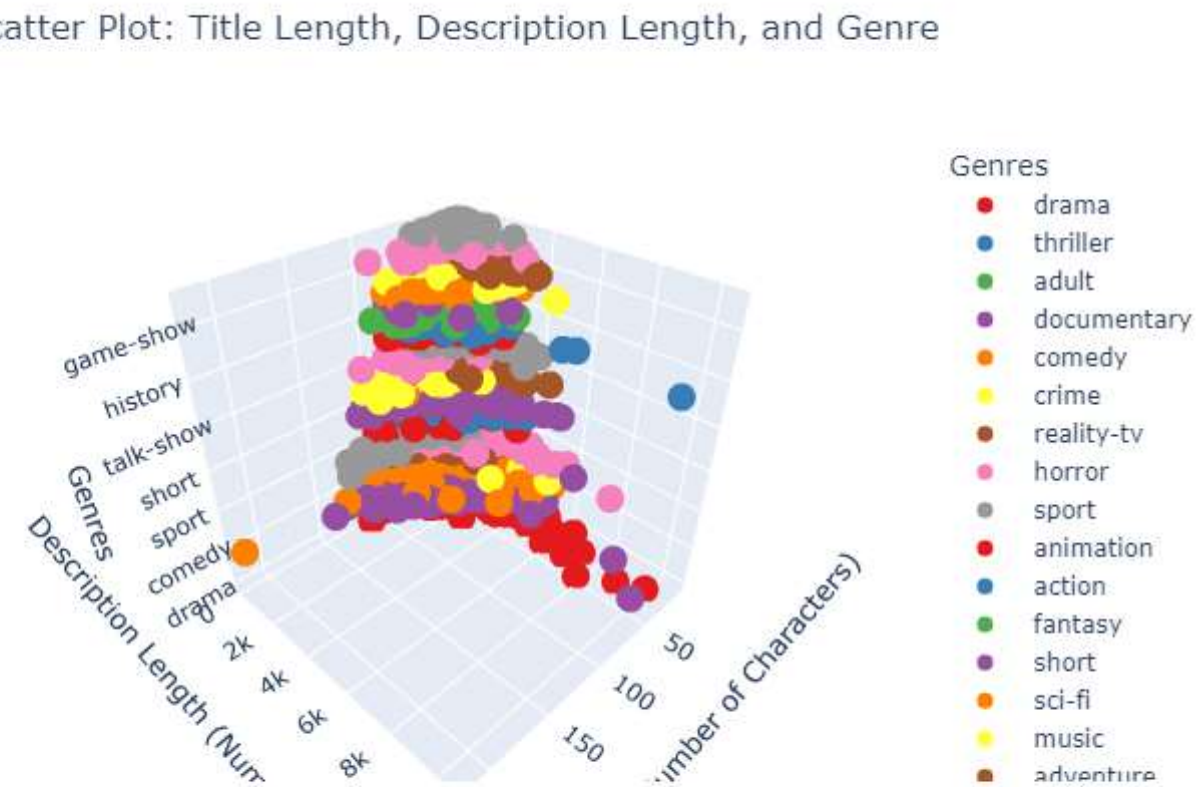
```
# pip install -U kaleido
```

In [19]:

```
R
# Calculate the length of movie titles and descriptions
df['Title Length'] = df['Title'].apply(len)
df['Description Length'] = df['Description'].apply(len)

# Create a 3D scatter plot using Plotly Express
fig = px.scatter_3d(df, x='Title Length', y='Description Length', z='Genre',
                    color='Genre', size_max=10,
                    labels={'Title Length': 'Title Length (Number of Characters)',
                            'Description Length': 'Description Length (Number of Characters)',
                            'Genre': 'Genres'},
                    title='3D Scatter Plot: Title Length, Description Length, and Genre',
                    color_discrete_sequence=px.colors.qualitative.Set1)

# Save the 3D scatter plot as a static image (PNG format)
pio.write_image(fig, '3d_scatter_plot.png', format='png')
```

In [20]:
```
from IPython.display import Image
Image('3d_scatter_plot.png')
```

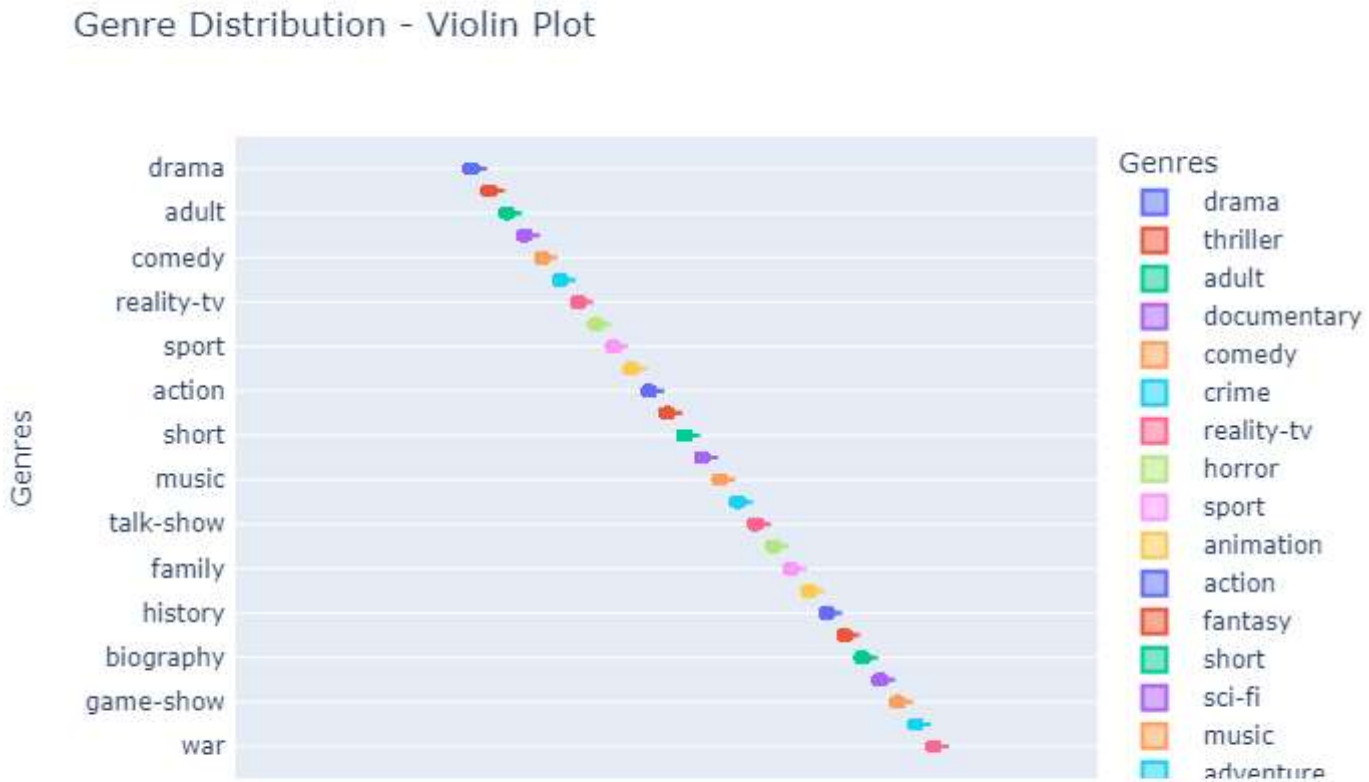
Out[20]:



In [21]:

```
# Create a violin plot using Plotly Express
fig = px.violin(df, y="Genre", box=True, points="all",
                color="Genre",
                title='Genre Distribution - Violin Plot',
                labels={'Genre': 'Genres'})

fig.update_layout(yaxis_title='Genres')

# Save the violin plot as a static image (PNG format)
pio.write_image(fig, 'violin_plot.png', format='png')

# Display the static image in Jupyter Notebook
from IPython.display import Image
Image('violin_plot.png')
```

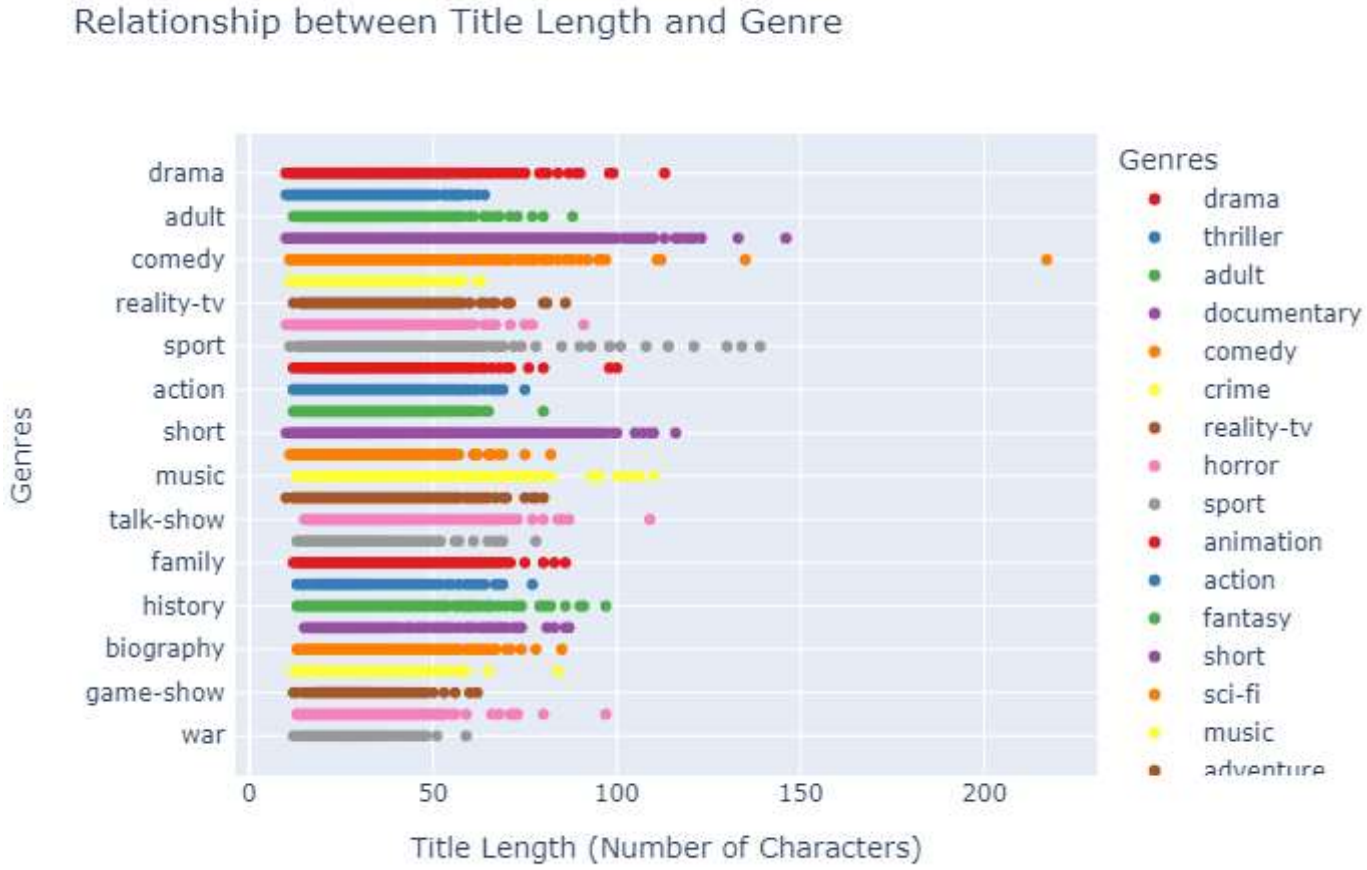
Out[21]:


Genre Distribution - Violin Plot
Genres
drama
adult
comedy
reality-tv
sport
action
short
music
talk-show
family
history
biography
game-show
war
Genres
drama
thriller
adult
documentary
comedy
crime
reality-tv
horror
sport
animation
action
fantasy
short
sci-fi
music
adventure

In [23]:

```
# Calculate the length of movie titles
df['Title Length'] = df['Title'].apply(len)

# Create an interactive scatter plot using Plotly Express
fig = px.scatter(df, x='Title Length', y='Genre', color='Genre',
                 title='Relationship between Title Length and Genre',
                 labels={'Title Length': 'Title Length (Number of Characters)', 'Genre': 'Genres'},
                 color_discrete_sequence=px.colors.qualitative.Set1)

# Show the interactive plot
#fig.show()

# Save the scatter plot as a static image (PNG format)
pio.write_image(fig, 'title_length_genre_scatter.png', format='png')

# Display the static image in Jupyter Notebook
Image('title_length_genre_scatter.png')
```

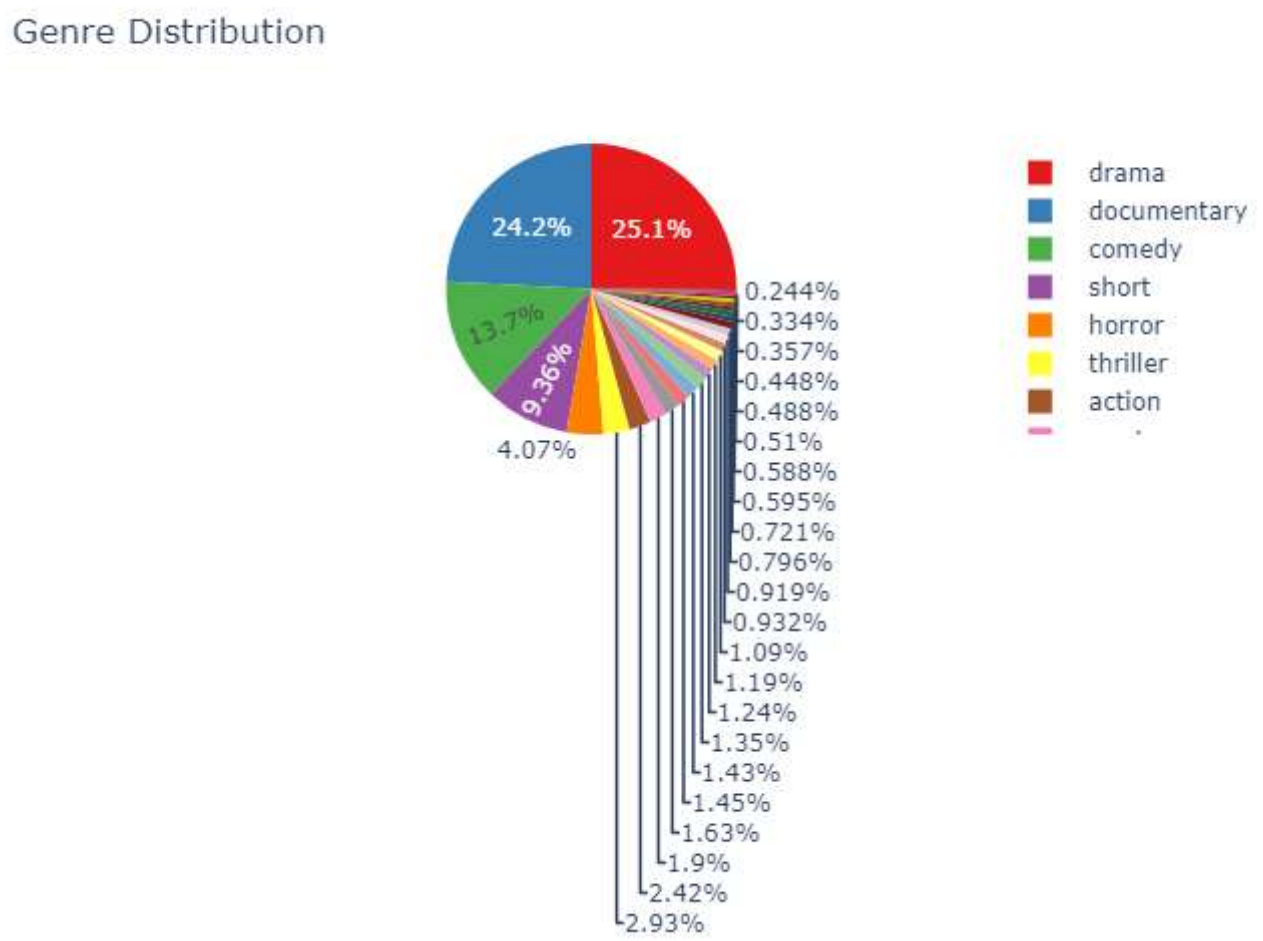
Out[23]:


Relationship between Title Length and Genre
Genres
drama
adult
comedy
reality-tv
sport
action
short
music
talk-show
family
history
biography
game-show
war
0
50
100
150
200
Title Length (Number of Characters)
Genres
drama
thriller
adult
documentary
comedy
crime
reality-tv
horror
sport
animation
action
fantasy
short
sci-fi
music
adventure

In [24]:

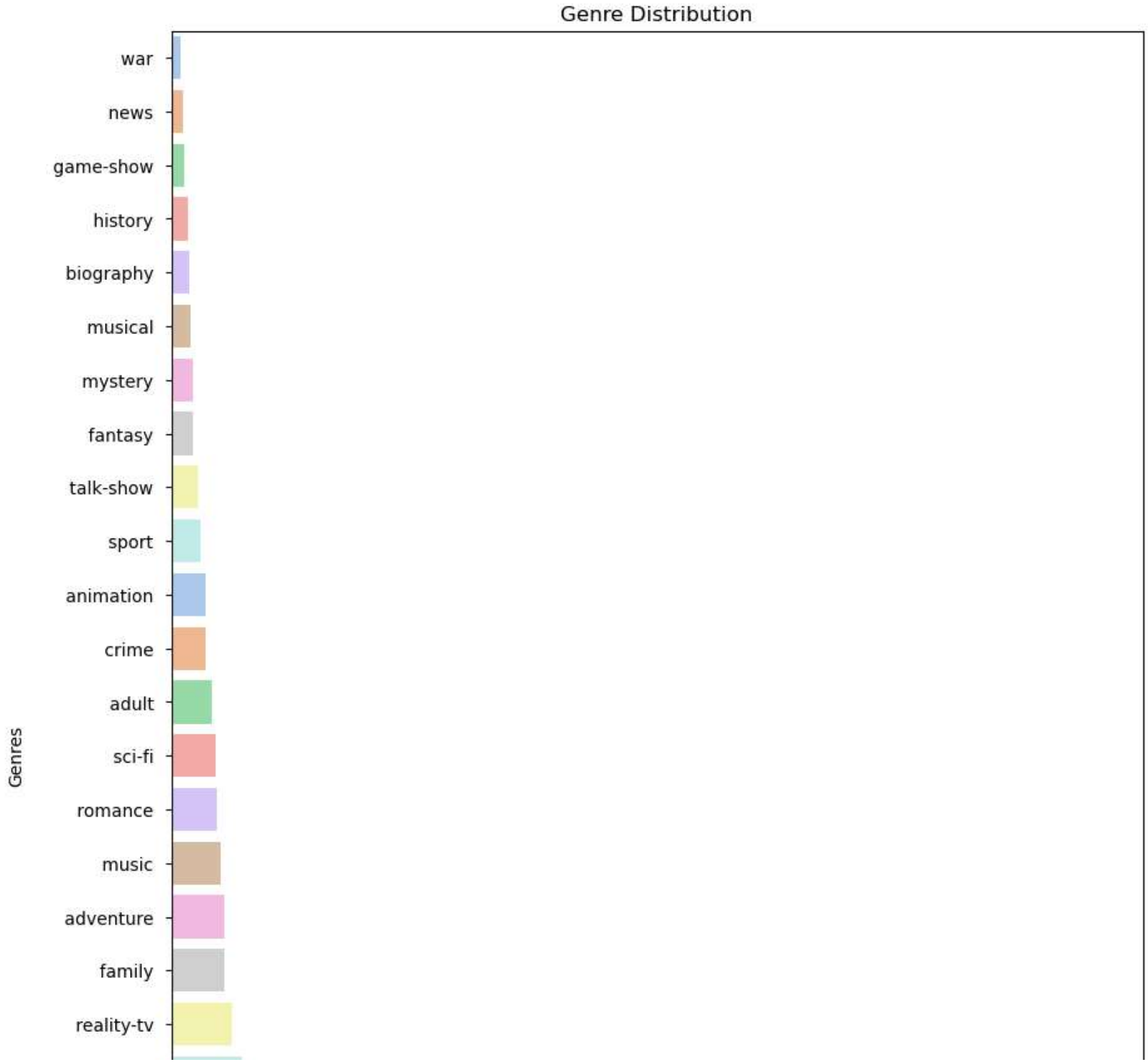
```
# Count the frequency of each genre
genre_counts = df["Genre"].value_counts()

# Create an interactive pie chart using Plotly Express
fig = px.pie(genre_counts,
             names=genre_counts.index,
             values=genre_counts.values,
             title='Genre Distribution',
             color_discrete_sequence=px.colors.qualitative.Set1)

# Show the interactive pie chart
#fig.show()

# Save the pie chart as a static image (PNG format)
pio.write_image(fig, 'genre_distribution_pie_chart.png', format='png')

# Display the static image in Jupyter Notebook
Image('genre_distribution_pie_chart.png')
```

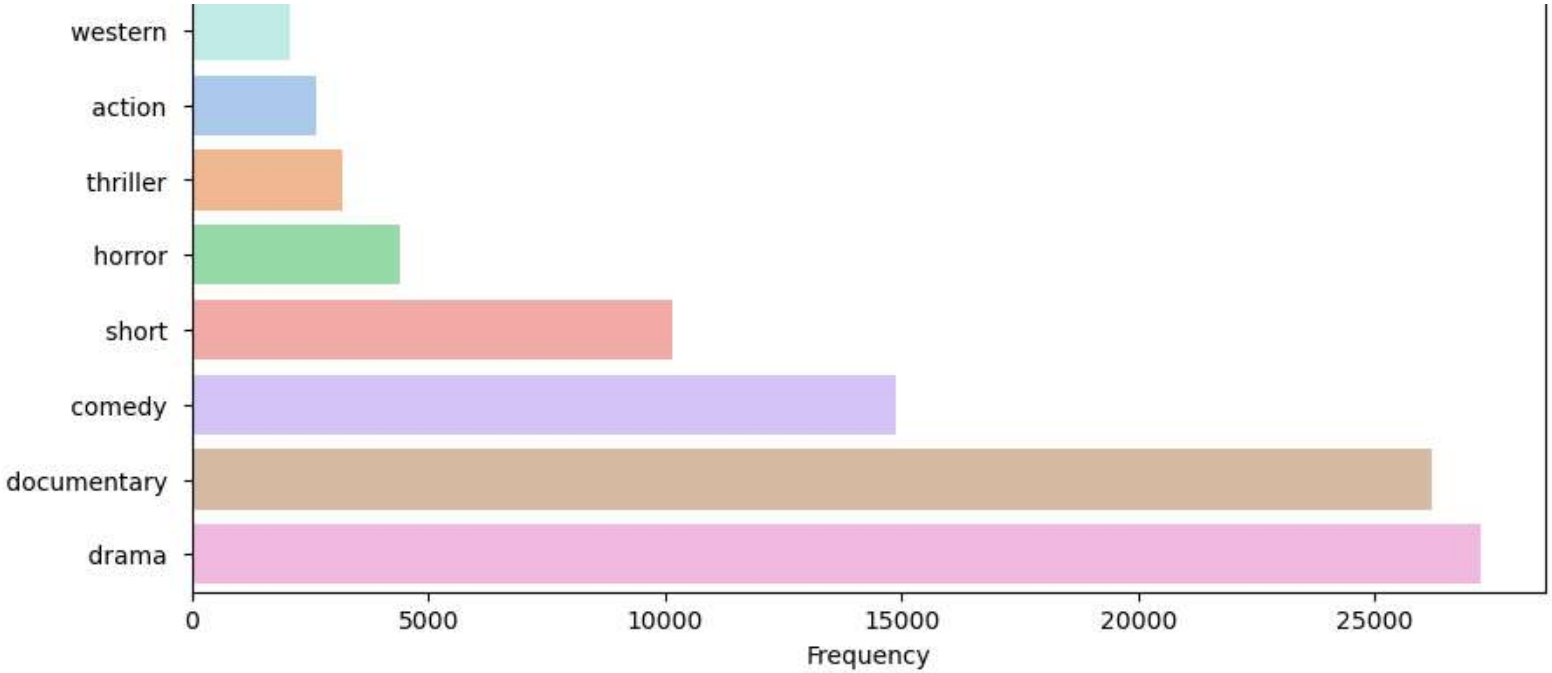
Out[24]:


Genre Distribution
24.2%
25.1%
13.7%
9.36%
4.07%
0.244%
0.334%
0.357%
0.448%
0.488%
0.51%
0.588%
0.595%
0.721%
0.796%
0.919%
0.932%
1.09%
1.19%
1.24%
1.35%
1.43%
1.45%
1.63%
1.9%
2.42%
2.93%
drama
documentary
comedy
short
horror
thriller
action

In [ ]:

```
# Count the frequency of each genre
genre_counts = df["Genre"].value_counts()

# Sort genres based on frequency
sorted_genres = genre_counts.sort_values(ascending=True)

# Set the color palette
colors = sns.color_palette("pastel")

# Create a horizontal bar chart with Seaborn for a stylish visualization
plt.figure(figsize=(10, 15))
sns.barplot(x=sorted_genres.values, y=sorted_genres.index, palette=colors)

plt.title("Genre Distribution")
plt.xlabel("Frequency")
plt.ylabel("Genres")
plt.show()
```

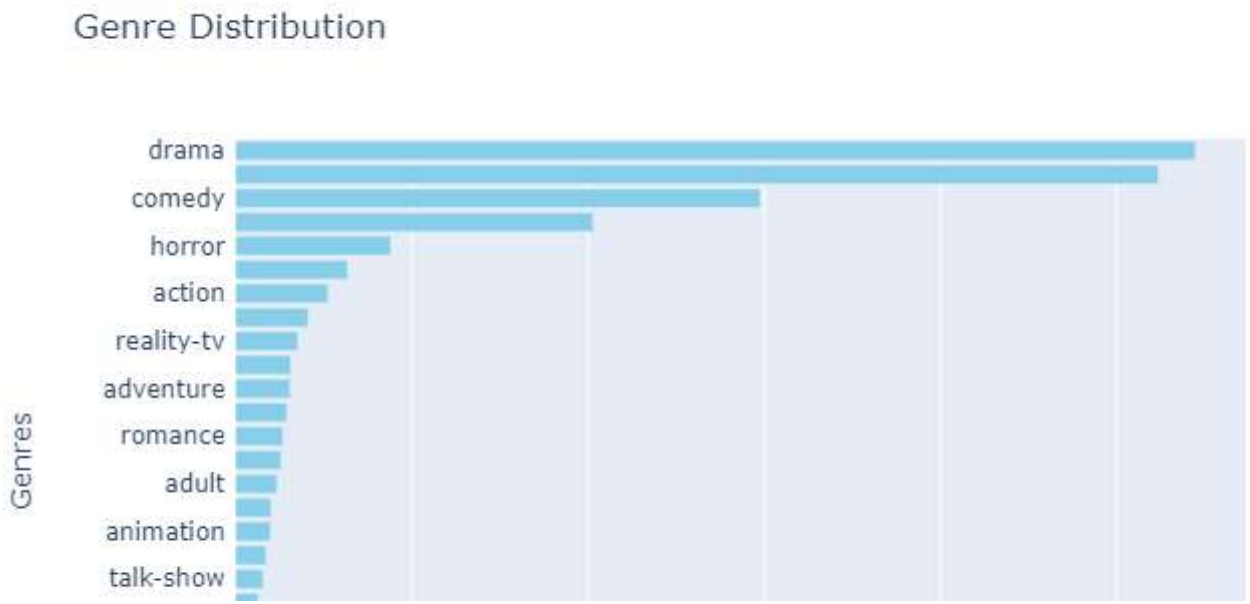

Genre Distribution
Genres
war
news
game-show
history
biography
musical
mystery
fantasy
talk-show
sport
animation
crime
adult
sci-fi
romance
music
adventure
family
reality-tv


western
action
thriller
horror
short
comedy
documentary
drama
0
5000
10000
15000
20000
25000
Frequency

In [25]:

```
# Count the frequency of each genre
genre_counts = df["Genre"].value_counts()

# Sort genres based on frequency
sorted_genres = genre_counts.sort_values(ascending=True)

# Create a horizontal bar chart using Plotly Express
fig = px.bar(y=sorted_genres.index, x=sorted_genres.values, orientation='h',
             labels={'x': 'Frequency', 'y': 'Genres'},
             title='Genre Distribution',
             color_discrete_sequence=['skyblue'])

fig.update_layout(showlegend=False, yaxis={'categoryorder':'total ascending'})

# Show the interactive bar chart
#fig.show()

# Save the bar chart as a static image (PNG format)
pio.write_image(fig, 'genre_distribution_bar_chart.png', format='png')

# Display the static image in Jupyter Notebook
Image('genre_distribution_bar_chart.png')
```

Out[25]:


Genre Distribution
Genres
drama
comedy
horror
action
reality-tv
adventure
romance
adult
animation
talk-show

In [ ]:

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Get unique genres from the dataset
unique_genres = df['Genre'].unique()

# Set up the plot grid
num_rows = len(unique_genres) // 2 + len(unique_genres) % 2
num_cols = 2
plt.figure(figsize=(15, 5 * num_rows))

# Generate word clouds for each genre
for i, genre in enumerate(unique_genres, 1):
    plt.subplot(num_rows, num_cols, i)
    genre_description = " ".join(list(df[df["Genre"] == genre]['Description']))
    wordcloud = WordCloud(max_words=400, width=900, height=400, background_color='white').generate(genre_d
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f'Word Cloud for {genre} Genre', fontsize=18)
    plt.axis('off')

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

4. Feature Enhancement and Refinement

- **Dimensionality Reduction:** Removing irrelevant columns reduces the dataset's dimensionality, preventing the curse of dimensionality and sparsity issues in high-dimensional spaces.
- **Computational Efficiency:** Smaller datasets with fewer features require less memory and computational resources, speeding up training and prediction processes.
- **Model Interpretability:** Models with fewer features are easier to interpret, making it simpler to understand the impact of essential variables on predictions.
- **Avoiding Overfitting:** Irrelevant features can introduce noise, leading to overfitting. Removing them reduces the risk of the model learning noise patterns.
- **Enhanced Model Performance:** Focusing on relevant features ensures the model is trained on meaningful information, potentially improving predictive accuracy.

In [ ]:
```
## remove id column from head
data = df.drop(["Title","id"] , axis = 1) # will drop column
data.head()
```

Out[83]:

	Genre	Description	Despcription_clean	Title Length	Description Length
0	drama	Listening in to a conversation between his do...	listening conversation doctor parent 10yearold...	30	546
1	thriller	A brother and sister with a past incestuous r...	brother sister past incestuous relationship cu...	14	184
2	adult	As the bus empties the students for their fie...	bus empty student field trip museum natural hi...	34	650
3	drama	To help their unemployed father make ends mee...	help unemployed father make end meet edith twi...	23	1082
4	drama	The film's title refers not only to the un-re...	film title refers unrecovered body ground zero...	24	625

5. Model Selection and Fine-Tuning

- In this section we will try:
- **Model Selection:** Choose appropriate text vectorization techniques like CountVectorizer and TF-IDF.
- **Text Vectorization:** Convert textual data into numerical vectors using CountVectorizer (token counts) and TF-IDF (word importance weights).
- **Fine-Tuning:** Experiment with different parameters and configurations to optimize the vectorization process.
- **Accuracy Evaluation:** Assess the accuracy of models generated with different vectorization techniques to determine the most effective method for the specific dataset.

Importing necessary libraries for Model Selection and Fine-Tuning

In [ ]:

```
## import necessary library for
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report,confusion_matrix,ConfusionMatrixDisplay
```

Converting Genre into Numerical form

In [ ]:
```
#Convert sentiment labels to numerical values for modeling
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
data['Genre_encoded'] = label_encoder.fit_transform(data['Genre'])
data['Genre_encoded']

class_names= list(label_encoder.classes_)
class_names
```

Out[85]:
```
[' action ',
 ' adult ',
 ' adventure ',
 ' animation ',
 ' biography ',
 ' comedy ',
 ' crime ',
 ' documentary ',
 ' drama ',
 ' family ',
 ' fantasy ',
 ' game-show ',
 ' history ',
 ' horror ',
 ' music ',
 ' musical ',
 ' mystery ',
 ' news ',
 ' reality-tv ',
 ' romance ',
 ' sci-fi ',
 ' short ',
 ' sport ',
 ' talk-show ',
 ' thriller ',
 ' war ',
 ' western ']
```

In [ ]:
```
data.head()
```

Out[86]:

	Genre	Description	Despcription_clean	Title Length	Description Length	Genre_encoded
0	drama	Listening in to a conversation between his do...	listening conversation doctor parent 10yearold...	30	546	8
1	thriller	A brother and sister with a past incestuous r...	brother sister past incestuous relationship cu...	14	184	24
2	adult	As the bus empties the students for their fie...	bus empty student field trip museum natural hi...	34	650	1
3	drama	To help their unemployed father make ends mee...	help unemployed father make end meet edith twi...	23	1082	8
4	drama	The film's title refers not only to the un-re...	film title refers unrecovered body ground zero...	24	625	8

Split the data to test and train

In [ ]:
```
## Split the data
x = data["Despcription_clean"]
y = data["Genre"]
```

In [ ]:
```
x
```

Out[88]:
```
0        listening conversation doctor parent 10yearold...
1        brother sister past incestuous relationship cu...
2        bus empty student field trip museum natural hi...
3        help unemployed father make end meet edith twi...
4        film title refers unrecovered body ground zero...
                               ...
54195    covering multiple genre tale light dark anthol...
54196    alice cora munro attempt find father british o...
54197    movie 169 year making oliver twist artful dodg...
54198    popular mysterious rock dj mike mallard askew ...
54199    curitiba city movement rhythm different pulsat...
Name: Despcription_clean, Length: 108414, dtype: object
```

In [ ]:
```
y
```

Out[89]:
```
0            drama
1         thriller
2            adult
3            drama
4            drama
              ...
54195       horror
54196      western
54197        adult
54198        drama
54199        drama
Name: Genre, Length: 108414, dtype: object
```

In [ ]:
```
x_train ,x_test ,y_train ,y_test = train_test_split(x ,y ,test_size = 0.5)
```

Training the Model Using CountVectorizer Technique

In [ ]:
```
vectorize = CountVectorizer()
x_train1 = vectorize.fit_transform(x_train)
x_test1 =  vectorize.transform(x_test)
```

In [ ]:
```
x_train1
```

Out[92]:
```
<54207x139581 sparse matrix of type '<class 'numpy.int64'>'
        with 2647691 stored elements in Compressed Sparse Row format>
```

In [ ]:
```
x_test1
```

Out[93]:
```
<54207x139581 sparse matrix of type '<class 'numpy.int64'>'
        with 2557989 stored elements in Compressed Sparse Row format>
```

1. **CountVectorizer Initialization:** `CountVectorizer()` is an instance of the CountVectorizer class from scikit-learn. It's used to convert a collection of text documents to a matrix of token counts.
2. **Fitting and Transforming Training Data:** `x_train1 = vectorize.fit_transform(x_train)` fits the CountVectorizer on the training data (`x_train`) and transforms it into a sparse matrix. Each row in the matrix represents a document, and each column represents a unique word in the entire corpus. The values in the matrix represent the count of the corresponding word in the

respective document.

3. **Transforming Test Data:** `x_test1 = vectorize.transform(x_test)` uses the same CountVectorizer instance to transform the test data (`x_test`) into a sparse matrix. The transformation ensures consistency in the features used by the machine learning model for both training and testing.
4. **Sparse Matrix Representation:** The resulting matrices (`x_train1` and `x_test1`) are in sparse matrix format, meaning that most of the elements in the matrix are zero. This format is memory-efficient for large datasets because it only stores the non-zero elements. In your specific case, the matrices are of size `54207x139581` , indicating 54,207 documents in the dataset and 139,581 unique words considered for vectorization.

In summary, CountVectorizer converts text data into numerical format suitable for machine learning algorithms by creating a sparse matrix where rows represent documents, columns represent words, and the values represent word frequencies in the documents. This numeric representation allows machine learning models to process and learn from textual data.

Multinomial Naive Bayes classifier

In [ ]:

```
mnb = MultinomialNB()
mnb.fit(x_train1, y_train)

# Evaluate the model
train_accuracy = mnb.score(x_train1, y_train)
test_accuracy = mnb.score(x_test1, y_test)

# Print model accuracy
print(f"Training Accuracy: {train_accuracy:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")

# Make predictions and display classification report
y_pred = mnb.predict(x_test1)
class_report = classification_report(y_test, y_pred)

print("Classification Report:\n", class_report)
```

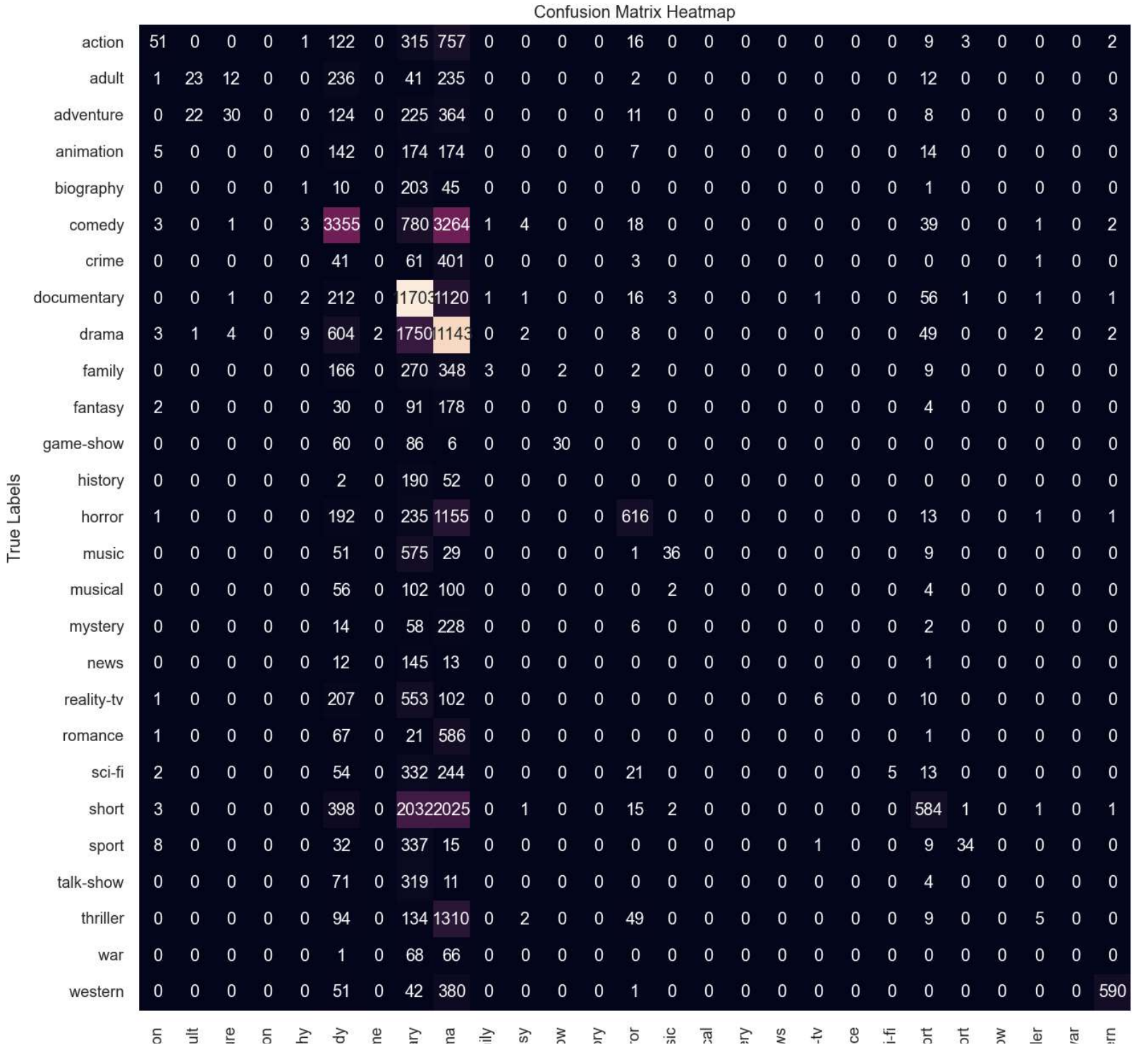
```
Training Accuracy: 0.63
Test Accuracy: 0.52
Classification Report:
               precision    recall  f1-score   support

      action       0.63      0.04      0.08      1276
       adult       0.50      0.04      0.08       562
   adventure       0.62      0.04      0.07       787
   animation       0.00      0.00      0.00       516
   biography       0.06      0.00      0.01       260
      comedy       0.52      0.45      0.48      7471
       crime       0.00      0.00      0.00       507
 documentary       0.56      0.89      0.69     13119
       drama       0.46      0.82      0.59     13579
      family       0.60      0.00      0.01       800
     fantasy       0.00      0.00      0.00       314
   game-show       0.94      0.16      0.28       182
     history       0.00      0.00      0.00       244
      horror       0.77      0.28      0.41      2214
       music       0.84      0.05      0.10       701
     musical       0.00      0.00      0.00       264
     mystery       0.00      0.00      0.00       308
        news       0.00      0.00      0.00       171
  reality-tv       0.75      0.01      0.01       879
     romance       0.00      0.00      0.00       676
      sci-fi       1.00      0.01      0.01       671
       short       0.68      0.12      0.20      5063
       sport       0.87      0.08      0.14       436
   talk-show       0.00      0.00      0.00       405
    thriller       0.42      0.00      0.01      1603
         war       0.00      0.00      0.00       135
     western       0.98      0.55      0.71      1064

    accuracy                           0.52     54207
   macro avg       0.41      0.13      0.14     54207
weighted avg       0.53      0.52      0.44     54207
```

- **Model Type:** Multinomial Naive Bayes
- **Training Accuracy:** 63%
- **Testing Accuracy:** 52%

- **Best Predictions:** Genres like "game-show" (precision: 94%) and "western" (precision: 98%)
- **Challenges:** Model struggles with genres like "animation" (precision: 0%) and "sci-fi" (precision: 1%), indicating difficulty in distinguishing these categories. Overall accuracy is moderate, indicating the potential for improvements in genre predictions.

In [ ]:

```
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(conf_matrix, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap
True Labels
action 51 0 0 0 1 122 0 315 757 0 0 0 0 16 0 0 0 0 0 0 0 9 3 0 0 0 2
adult 1 23 12 0 0 236 0 41 235 0 0 0 0 2 0 0 0 0 0 0 0 12 0 0 0 0 0
adventure 0 22 30 0 0 124 0 225 364 0 0 0 0 11 0 0 0 0 0 0 0 8 0 0 0 0 3
animation 5 0 0 0 0 142 0 174 174 0 0 0 0 7 0 0 0 0 0 0 0 14 0 0 0 0 0
biography 0 0 0 0 1 10 0 203 45 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
comedy 3 0 1 0 3 3355 0 780 3264 1 4 0 0 18 0 0 0 0 0 0 0 39 0 0 1 0 2
crime 0 0 0 0 0 41 0 61 401 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 1 0 0
documentary 0 0 1 0 2 212 0 1703 1120 1 1 0 0 16 3 0 0 0 1 0 0 56 1 0 1 0 1
drama 3 1 4 0 9 604 2 1750 1143 0 2 0 0 8 0 0 0 0 0 0 0 49 0 0 2 0 2
family 0 0 0 0 0 166 0 270 348 3 0 2 0 2 0 0 0 0 0 0 0 9 0 0 0 0 0
fantasy 2 0 0 0 0 30 0 91 178 0 0 0 0 9 0 0 0 0 0 0 0 4 0 0 0 0 0
game-show 0 0 0 0 0 60 0 86 6 0 0 30 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
history 0 0 0 0 0 2 0 190 52 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
horror 1 0 0 0 0 192 0 235 1155 0 0 0 0 616 0 0 0 0 0 0 0 13 0 0 1 0 1
music 0 0 0 0 0 51 0 575 29 0 0 0 0 1 36 0 0 0 0 0 0 9 0 0 0 0 0
musical 0 0 0 0 0 56 0 102 100 0 0 0 0 0 2 0 0 0 0 0 0 4 0 0 0 0 0
mystery 0 0 0 0 0 14 0 58 228 0 0 0 0 6 0 0 0 0 0 0 0 2 0 0 0 0 0
news 0 0 0 0 0 12 0 145 13 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
reality-tv 1 0 0 0 0 207 0 553 102 0 0 0 0 0 0 0 0 0 6 0 0 10 0 0 0 0 0
romance 1 0 0 0 0 67 0 21 586 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
sci-fi 2 0 0 0 0 54 0 332 244 0 0 0 0 21 0 0 0 0 0 0 5 13 0 0 0 0 0
short 3 0 0 0 0 398 0 2032 2025 0 1 0 0 15 2 0 0 0 0 0 0 584 1 0 1 0 1
sport 8 0 0 0 0 32 0 337 15 0 0 0 0 0 0 0 0 0 1 0 0 9 34 0 0 0 0
talk-show 0 0 0 0 0 71 0 319 11 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0
thriller 0 0 0 0 0 94 0 134 1310 0 2 0 0 49 0 0 0 0 0 0 0 9 0 0 5 0 0
war 0 0 0 0 0 1 0 68 66 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
western 0 0 0 0 0 51 0 42 380 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 590

acti
ad
adventu
animati
biograp
come
crin
documenta
dran
fam
fanta
game-sh
histo
horr
mus
music
myste
new
reality-
roman
sci
sh
spo
talk-sh
thrill
w
weste
Predicted Labels

LogisticRegression

In [ ]:

```
model = LogisticRegression()
model.fit(x_train1 ,y_train)
print("Model Score on Training data",model.score(x_train1 ,y_train))
print("Model Score on Testing data",model.score(x_test1 ,y_test))
y_pred = model.predict(x_test1)
print(classification_report(y_pred ,y_test))
```

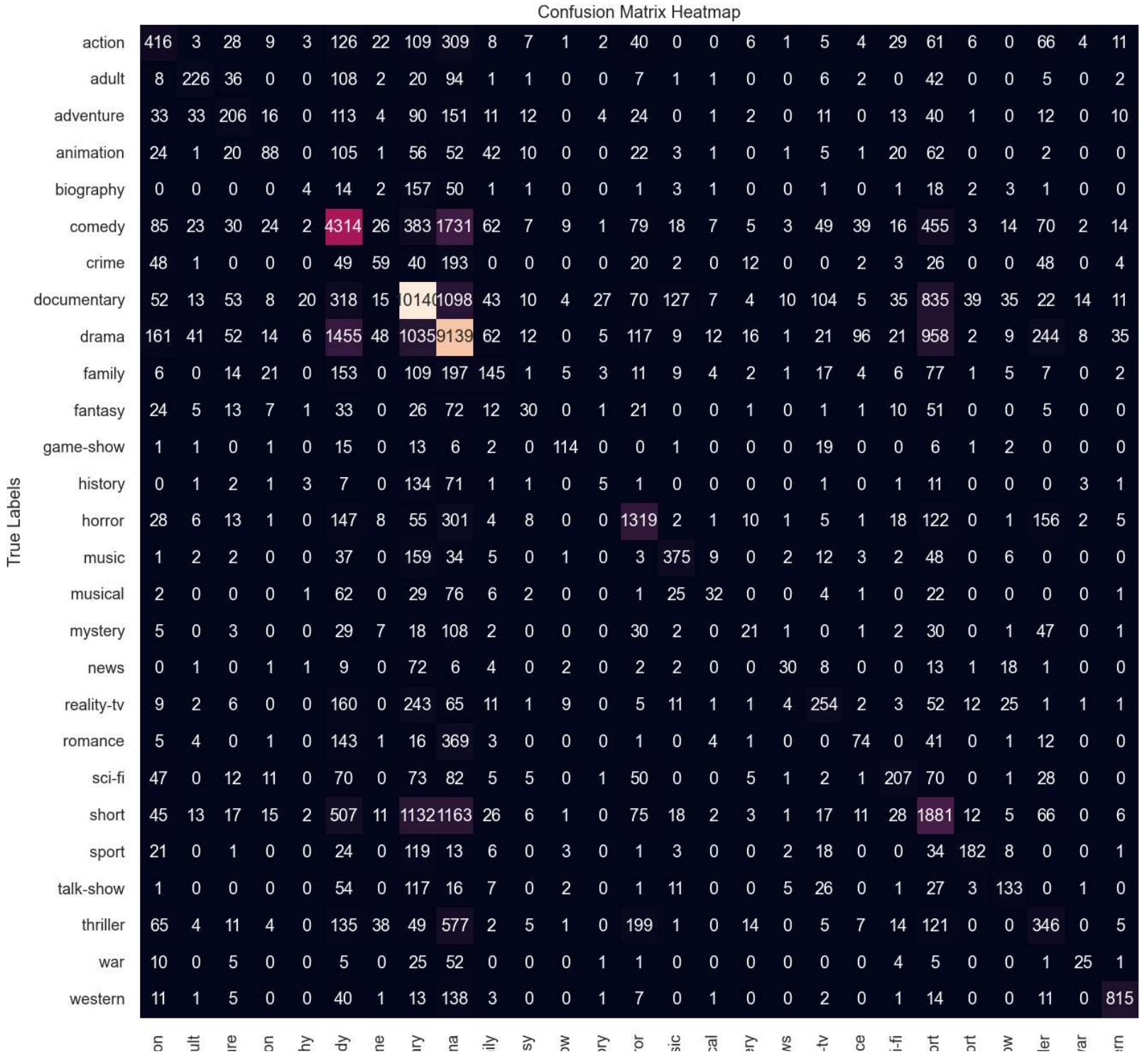
```
Model Score on Training data 0.9925101924105743
Model Score on Testing data 0.5641337834597008
              precision    recall  f1-score   support

      action       0.33      0.38      0.35      1108
       adult       0.40      0.59      0.48       381
   adventure       0.26      0.39      0.31       529
   animation       0.17      0.40      0.24       222
   biography       0.02      0.09      0.03        43
      comedy       0.58      0.52      0.55      8232
       crime       0.12      0.24      0.16       245
 documentary       0.77      0.70      0.74     14432
       drama       0.67      0.57      0.61     16163
      family       0.18      0.31      0.23       474
     fantasy       0.10      0.25      0.14       119
   game-show       0.63      0.75      0.68       152
     history       0.02      0.10      0.03        51
      horror       0.60      0.63      0.61      2108
       music       0.53      0.60      0.57       623
     musical       0.12      0.38      0.18        84
     mystery       0.07      0.20      0.10       103
        news       0.18      0.47      0.26        64
  reality-tv       0.29      0.43      0.35       593
     romance       0.11      0.29      0.16       255
      sci-fi       0.31      0.48      0.37       435
       short       0.37      0.37      0.37      5122
       sport       0.42      0.69      0.52       265
   talk-show       0.33      0.50      0.40       267
    thriller       0.22      0.30      0.25      1151
         war       0.19      0.42      0.26        60
     western       0.77      0.88      0.82       926

    accuracy                           0.56     54207
   macro avg       0.32      0.44      0.36     54207
weighted avg       0.60      0.56      0.58     54207
```

- **Model Type:** Logistic Regression
- **Training Accuracy:** 99.25%
- **Testing Accuracy:** 56.41%
- **Best Predictions:** Genres like "game-show" (precision: 63%) and "western" (precision: 77%)

- **Areas for Improvement:** Overall accuracy indicates the potential for improvements in genre predictions, especially for niche categories like "biography" (precision: 2%) and "history" (precision: 2%).

In [ ]:

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap
True Labels
action 416 3 28 9 3 126 22 109 309 8 7 1 2 40 0 0 6 1 5 4 29 61 6 0 66 4 11
adult 8 226 36 0 0 108 2 20 94 1 1 0 0 7 1 1 0 0 6 2 0 42 0 0 5 0 2
adventure 33 33 206 16 0 113 4 90 151 11 12 0 4 24 0 1 2 0 11 0 13 40 1 0 12 0 10
animation 24 1 20 88 0 105 1 56 52 42 10 0 0 22 3 1 0 1 5 1 20 62 0 0 2 0 0
biography 0 0 0 0 4 14 2 157 50 1 1 0 0 1 3 1 0 0 1 0 1 18 2 3 1 0 0
comedy 85 23 30 24 2 4314 26 383 1731 62 7 9 1 79 18 7 5 3 49 39 16 455 3 14 70 2 14
crime 48 1 0 0 0 49 59 40 193 0 0 0 0 20 2 0 12 0 0 2 3 26 0 0 48 0 4
documentary 52 13 53 8 20 318 15 1014 1098 43 10 4 27 70 127 7 4 10 104 5 35 835 39 35 22 14 11
drama 161 41 52 14 6 1455 48 1035 9139 62 12 0 5 117 9 12 16 1 21 96 21 958 2 9 244 8 35
family 6 0 14 21 0 153 0 109 197 145 1 5 3 11 9 4 2 1 17 4 6 77 1 5 7 0 2
fantasy 24 5 13 7 1 33 0 26 72 12 30 0 1 21 0 0 1 0 1 1 10 51 0 0 5 0 0
game-show 1 1 0 1 0 15 0 13 6 2 0 114 0 0 1 0 0 0 19 0 0 6 1 2 0 0 0
history 0 1 2 1 3 7 0 134 71 1 1 0 5 1 0 0 0 0 1 0 1 11 0 0 0 3 1
horror 28 6 13 1 0 147 8 55 301 4 8 0 0 1319 2 1 10 1 5 1 18 122 0 1 156 2 5
music 1 2 2 0 0 37 0 159 34 5 0 1 0 3 375 9 0 2 12 3 2 48 0 6 0 0 0
musical 2 0 0 0 1 62 0 29 76 6 2 0 0 1 25 32 0 0 4 1 0 22 0 0 0 0 1
mystery 5 0 3 0 0 29 7 18 108 2 0 0 0 30 2 0 21 1 0 1 2 30 0 1 47 0 1
news 0 1 0 1 1 9 0 72 6 4 0 2 0 2 2 0 0 30 8 0 0 13 1 18 1 0 0
reality-tv 9 2 6 0 0 160 0 243 65 11 1 9 0 5 11 1 1 4 254 2 3 52 12 25 1 1 1
romance 5 4 0 1 0 143 1 16 369 3 0 0 0 1 0 4 1 0 0 74 0 41 0 1 12 0 0
sci-fi 47 0 12 11 0 70 0 73 82 5 5 0 1 50 0 0 5 1 2 1 207 70 0 1 28 0 0
short 45 13 17 15 2 507 11 1132 1163 26 6 1 0 75 18 2 3 1 17 11 28 1881 12 5 66 0 6
sport 21 0 1 0 0 24 0 119 13 6 0 3 0 1 3 0 0 2 18 0 0 34 182 8 0 0 1
talk-show 1 0 0 0 0 54 0 117 16 7 0 2 0 1 11 0 0 5 26 0 1 27 3 133 0 1 0
thriller 65 4 11 4 0 135 38 49 577 2 5 1 0 199 1 0 14 0 5 7 14 121 0 0 346 0 5
war 10 0 5 0 0 5 0 25 52 0 0 0 1 1 0 0 0 0 0 0 4 5 0 0 1 25 1
western 11 1 5 0 0 40 1 13 138 3 0 0 1 7 0 1 0 0 2 0 1 14 0 0 11 0 815

acti
ad
adventu
animati
biograp
come
crin
documenta
dran
fam
fanta
game-sh
histo
horr
mus
music
myste
new
reality-
roman
sci
sh
spo
talk-sh
thrill
w
weste
Predicted Labels

Support Vector Machine (SVC)

In [ ]:

```
svm = LinearSVC()
svm.fit(x_train1 ,y_train)
print("Model Score on Training data",svm.score(x_train1 ,y_train))
print("Model Score on Testing data",svm.score(x_test1 ,y_test))
y_pred = svm.predict(x_test1)
print(classification_report(y_pred ,y_test))
```

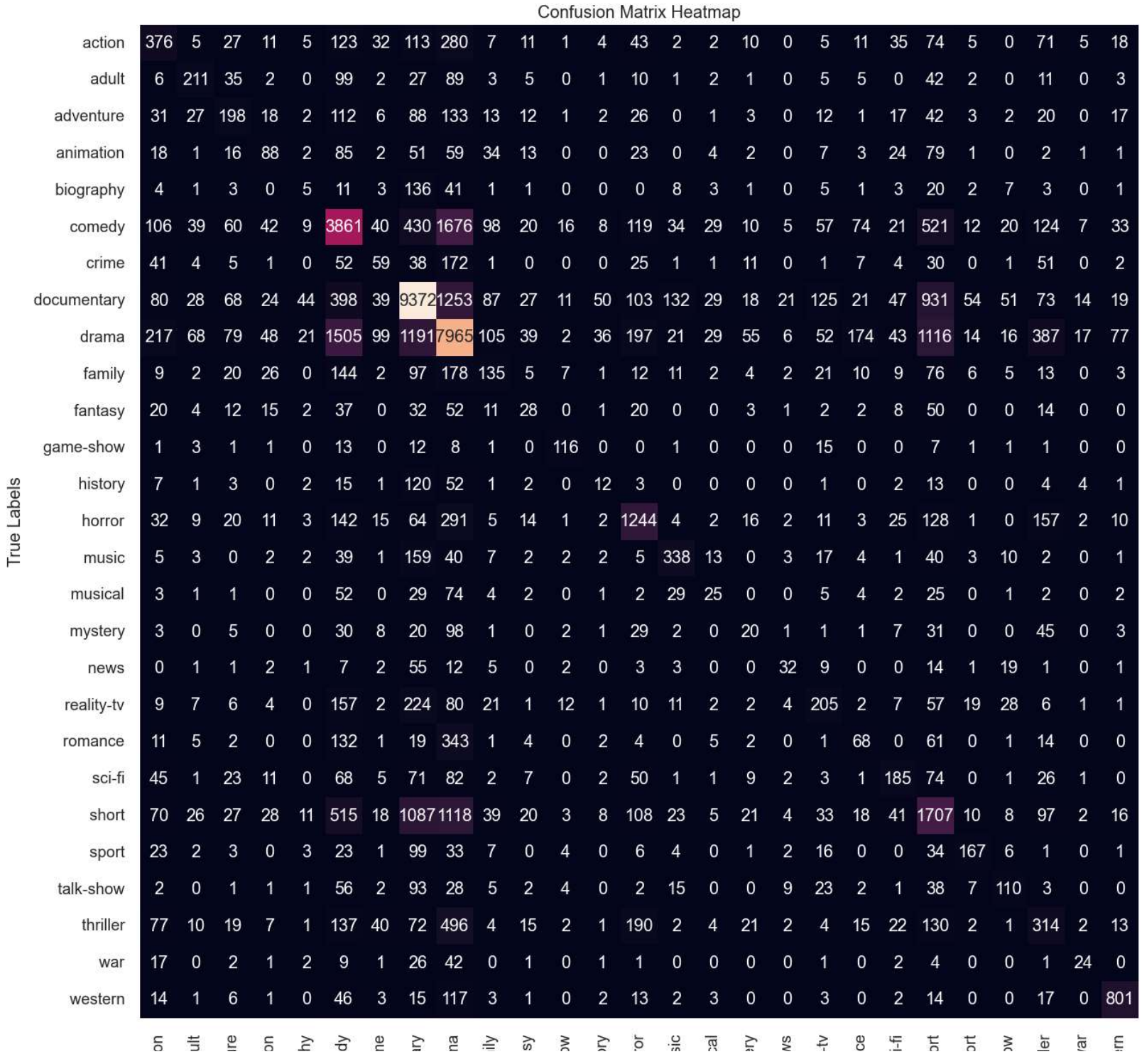
```
Model Score on Training data 0.9995941483572233
Model Score on Testing data 0.5103768885937241
              precision    recall  f1-score   support

      action       0.29      0.31      0.30      1227
       adult       0.38      0.46      0.41       460
   adventure       0.25      0.31      0.28       643
   animation       0.17      0.26      0.20       344
   biography       0.02      0.04      0.03       116
      comedy       0.52      0.49      0.50      7868
       crime       0.12      0.15      0.13       384
 documentary       0.71      0.68      0.70     13740
       drama       0.59      0.54      0.56     14812
      family       0.17      0.22      0.19       601
     fantasy       0.09      0.12      0.10       232
   game-show       0.64      0.62      0.63       186
     history       0.05      0.09      0.06       138
      horror       0.56      0.55      0.56      2248
       music       0.48      0.52      0.50       645
     musical       0.09      0.15      0.12       162
     mystery       0.06      0.10      0.08       210
        news       0.19      0.33      0.24        96
  reality-tv       0.23      0.32      0.27       640
     romance       0.10      0.16      0.12       427
      sci-fi       0.28      0.36      0.31       508
       short       0.34      0.32      0.33      5358
       sport       0.38      0.54      0.45       310
   talk-show       0.27      0.38      0.32       288
    thriller       0.20      0.22      0.21      1460
         war       0.18      0.30      0.22        80
     western       0.75      0.78      0.77      1024

    accuracy                           0.51     54207
   macro avg       0.30      0.35      0.32     54207
weighted avg       0.53      0.51      0.52     54207
```

- **Model Type:** Support Vector Machine (LinearSVC)
- **Training Accuracy:** 99.96%
- **Testing Accuracy:** 51.04%
- **Best Predictions:** Genres like "game-show" (precision: 64%) and "western" (precision: 75%)

- **Challenges:** Model struggles with genres like "action" (precision: 29%) and "family" (precision: 17%). It has difficulty in accurately predicting these categories, indicating a need for improvement, possibly through more data or feature engineering. Overall accuracy is moderate, suggesting room for enhancement in genre predictions.

In [ ]:

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap
True Labels
action 376 5 27 11 5 123 32 113 280 7 11 1 4 43 2 2 10 0 5 11 35 74 5 0 71 5 18
adult 6 211 35 2 0 99 2 27 89 3 5 0 1 10 1 2 1 0 5 5 0 42 2 0 11 0 3
adventure 31 27 198 18 2 112 6 88 133 13 12 1 2 26 0 1 3 0 12 1 17 42 3 2 20 0 17
animation 18 1 16 88 2 85 2 51 59 34 13 0 0 23 0 4 2 0 7 3 24 79 1 0 2 1 1
biography 4 1 3 0 5 11 3 136 41 1 1 0 0 0 8 3 1 0 5 1 3 20 2 7 3 0 1
comedy 106 39 60 42 9 3861 40 430 1676 98 20 16 8 119 34 29 10 5 57 74 21 521 12 20 124 7 33
crime 41 4 5 1 0 52 59 38 172 1 0 0 0 25 1 1 11 0 1 7 4 30 0 1 51 0 2
documentary 80 28 68 24 44 398 39 9372 1253 87 27 11 50 103 132 29 18 21 125 21 47 931 54 51 73 14 19
drama 217 68 79 48 21 1505 99 1191 7965 105 39 2 36 197 21 29 55 6 52 174 43 1116 14 16 387 17 77
family 9 2 20 26 0 144 2 97 178 135 5 7 1 12 11 2 4 2 21 10 9 76 6 5 13 0 3
fantasy 20 4 12 15 2 37 0 32 52 11 28 0 1 20 0 0 3 1 2 2 8 50 0 0 14 0 0
game-show 1 3 1 1 0 13 0 12 8 1 0 116 0 0 1 0 0 0 15 0 0 7 1 1 1 0 0
history 7 1 3 0 2 15 1 120 52 1 2 0 12 3 0 0 0 0 1 0 2 13 0 0 4 4 1
horror 32 9 20 11 3 142 15 64 291 5 14 1 2 1244 4 2 16 2 11 3 25 128 1 0 157 2 10
music 5 3 0 2 2 39 1 159 40 7 2 2 2 5 338 13 0 3 17 4 1 40 3 10 2 0 1
musical 3 1 1 0 0 52 0 29 74 4 2 0 1 2 29 25 0 0 5 4 2 25 0 1 2 0 2
mystery 3 0 5 0 0 30 8 20 98 1 0 2 1 29 2 0 20 1 1 1 7 31 0 0 45 0 3
news 0 1 1 2 1 7 2 55 12 5 0 2 0 3 3 0 0 32 9 0 0 14 1 19 1 0 1
reality-tv 9 7 6 4 0 157 2 224 80 21 1 12 1 10 11 2 2 4 205 2 7 57 19 28 6 1 1
romance 11 5 2 0 0 132 1 19 343 1 4 0 2 4 0 5 2 0 1 68 0 61 0 1 14 0 0
sci-fi 45 1 23 11 0 68 5 71 82 2 7 0 2 50 1 1 9 2 3 1 185 74 0 1 26 1 0
short 70 26 27 28 11 515 18 1087 1118 39 20 3 8 108 23 5 21 4 33 18 41 1707 10 8 97 2 16
sport 23 2 3 0 3 23 1 99 33 7 0 4 0 6 4 0 1 2 16 0 0 34 167 6 1 0 1
talk-show 2 0 1 1 1 56 2 93 28 5 2 4 0 2 15 0 0 9 23 2 1 38 7 110 3 0 0
thriller 77 10 19 7 1 137 40 72 496 4 15 2 1 190 2 4 21 2 4 15 22 130 2 1 314 2 13
war 17 0 2 1 2 9 1 26 42 0 1 0 1 1 0 0 0 0 1 0 2 4 0 0 1 24 0
western 14 1 6 1 0 46 3 15 117 3 1 0 2 13 2 3 0 0 3 0 2 14 0 0 17 0 801

acti
ad
adventu
animati
biograp
come
crin
documenta
dran
fam
fanta
game-sh
histo
horr
mus
music
myste
new
reality-
roman
sci
sh
spo
talk-sh
thrill
w
weste
Predicted Labels

RandomForestClassifier

In [ ]:

```
# Create a Random Forest model
random_forest = RandomForestClassifier()

random_forest.fit(x_train1, y_train)
print("Random Forest - Train Score:",random_forest.score(x_train1, y_train))
print("Random Forest - Test Score:", random_forest.score(x_test1, y_test))

y_pred = random_forest.predict(x_test1)
print(classification_report(y_pred ,y_test))
```

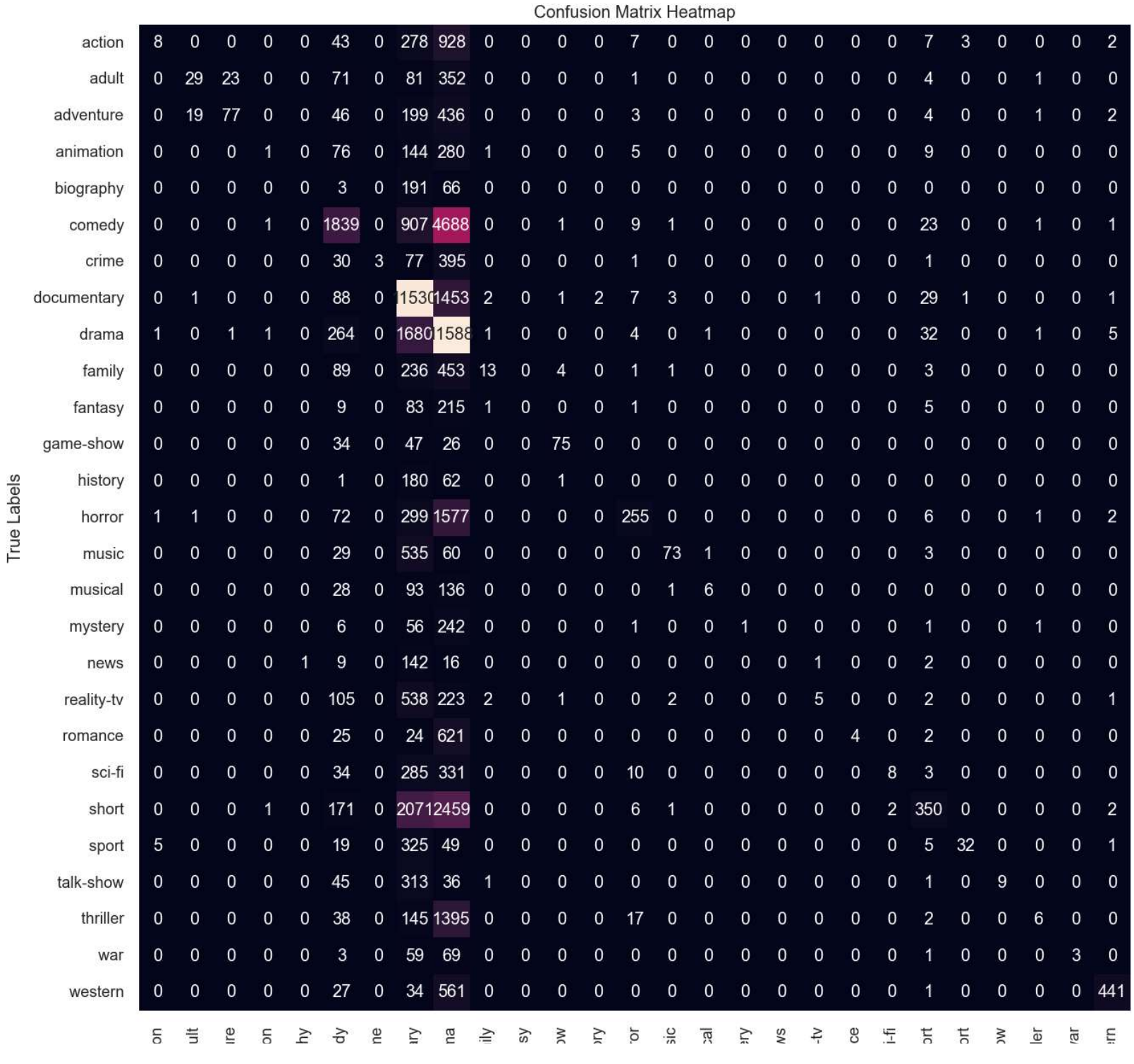
```
Random Forest - Train Score: 0.9996494917630564
Random Forest - Test Score: 0.48621026804656226
              precision    recall  f1-score   support

      action       0.01      0.53      0.01        15
       adult       0.05      0.58      0.09        50
   adventure       0.10      0.76      0.17       101
   animation       0.00      0.25      0.00         4
   biography       0.00      0.00      0.00         1
      comedy       0.25      0.57      0.34      3204
       crime       0.01      1.00      0.01         3
 documentary       0.88      0.56      0.68     20552
       drama       0.85      0.40      0.55     28717
      family       0.02      0.62      0.03        21
     fantasy       0.00      0.00      0.00         0
   game-show       0.41      0.90      0.57        83
     history       0.00      0.00      0.00         2
      horror       0.12      0.78      0.20       328
       music       0.10      0.89      0.19        82
     musical       0.02      0.75      0.04         8
     mystery       0.00      1.00      0.01         1
        news       0.00      0.00      0.00         0
  reality-tv       0.01      0.71      0.01         7
     romance       0.01      1.00      0.01         4
      sci-fi       0.01      0.80      0.02        10
       short       0.07      0.71      0.13       496
       sport       0.07      0.89      0.14        36
   talk-show       0.02      1.00      0.04         9
    thriller       0.00      0.50      0.01        12
         war       0.02      1.00      0.04         3
     western       0.41      0.96      0.58       458

    accuracy                           0.49     54207
   macro avg       0.13      0.64      0.14     54207
weighted avg       0.81      0.49      0.58     54207
```

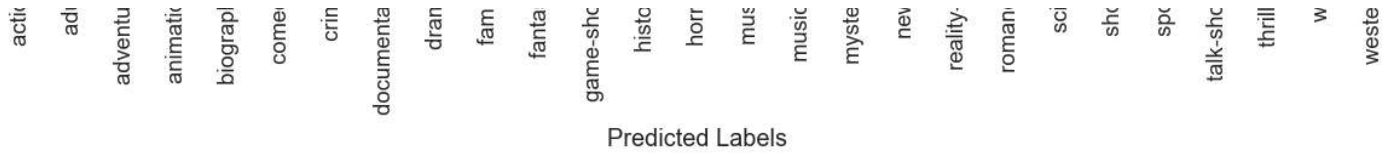
- **Model Type:** Random Forest Classifier
- **Training Accuracy:** 99.96%
- **Testing Accuracy:** 48.62%
- **Best Predictions:** Genres like "game-show" (precision: 41%) and "western" (precision: 41%)

- **Challenges:** Model struggles with genres like "action" (precision: 1%) and "family" (precision: 2%), indicating difficulty in distinguishing these categories. Overall accuracy is low, suggesting the need for model refinement or feature engineering to improve genre predictions.

In [ ]:

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap
True Labels
action 8 0 0 0 0 43 0 278 928 0 0 0 0 7 0 0 0 0 0 0 0 7 3 0 0 0 2
adult 0 29 23 0 0 71 0 81 352 0 0 0 0 1 0 0 0 0 0 0 0 4 0 0 1 0 0
adventure 0 19 77 0 0 46 0 199 436 0 0 0 0 3 0 0 0 0 0 0 0 4 0 0 1 0 2
animation 0 0 0 1 0 76 0 144 280 1 0 0 0 5 0 0 0 0 0 0 0 9 0 0 0 0 0
biography 0 0 0 0 0 3 0 191 66 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
comedy 0 0 0 1 0 1839 0 907 4688 0 0 1 0 9 1 0 0 0 0 0 0 23 0 0 1 0 1
crime 0 0 0 0 0 30 3 77 395 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0
documentary 0 1 0 0 0 88 0 1530 1453 2 0 1 2 7 3 0 0 0 1 0 0 29 1 0 0 0 1
drama 1 0 1 1 0 264 0 1680 1588 1 0 0 0 4 0 1 0 0 0 0 0 32 0 0 1 0 5
family 0 0 0 0 0 89 0 236 453 13 0 4 0 1 1 0 0 0 0 0 0 3 0 0 0 0 0
fantasy 0 0 0 0 0 9 0 83 215 1 0 0 0 1 0 0 0 0 0 0 0 5 0 0 0 0 0
game-show 0 0 0 0 0 34 0 47 26 0 0 75 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
history 0 0 0 0 0 1 0 180 62 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
horror 1 1 0 0 0 72 0 299 1577 0 0 0 0 255 0 0 0 0 0 0 0 6 0 0 1 0 2
music 0 0 0 0 0 29 0 535 60 0 0 0 0 0 73 1 0 0 0 0 0 3 0 0 0 0 0
musical 0 0 0 0 0 28 0 93 136 0 0 0 0 0 1 6 0 0 0 0 0 0 0 0 0 0 0
mystery 0 0 0 0 0 6 0 56 242 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0
news 0 0 0 0 1 9 0 142 16 0 0 0 0 0 0 0 0 0 1 0 0 2 0 0 0 0 0
reality-tv 0 0 0 0 0 105 0 538 223 2 0 1 0 0 2 0 0 0 5 0 0 2 0 0 0 0 1
romance 0 0 0 0 0 25 0 24 621 0 0 0 0 0 0 0 0 0 0 4 0 2 0 0 0 0 0
sci-fi 0 0 0 0 0 34 0 285 331 0 0 0 0 10 0 0 0 0 0 0 8 3 0 0 0 0 0
short 0 0 0 1 0 171 0 2071 2459 0 0 0 0 6 1 0 0 0 0 0 2 350 0 0 0 0 2
sport 5 0 0 0 0 19 0 325 49 0 0 0 0 0 0 0 0 0 0 0 0 5 32 0 0 0 1
talk-show 0 0 0 0 0 45 0 313 36 1 0 0 0 0 0 0 0 0 0 0 0 1 0 9 0 0 0
thriller 0 0 0 0 0 38 0 145 1395 0 0 0 0 17 0 0 0 0 0 0 0 2 0 0 6 0 0
war 0 0 0 0 0 3 0 59 69 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 3 0
western 0 0 0 0 0 27 0 34 561 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 441



Training the Model Using TF-IDF Vectorization Technique

In [ ]:
```
## Using TfidfVectorizer technique
vectorizer = TfidfVectorizer()
x_train2 = vectorize.fit_transform(x_train)
x_test2 =  vectorize.transform(x_test)
```

In [ ]:
```
x_train2
```

Out[119]:
```
<54207x139581 sparse matrix of type '<class 'numpy.int64'>'
        with 2647691 stored elements in Compressed Sparse Row format>
```

In [ ]:
```
x_test2
```

Out[120]:
```
<54207x139581 sparse matrix of type '<class 'numpy.int64'>'
        with 2557989 stored elements in Compressed Sparse Row format>
```

In [ ]:
```

```

Multinomial Naive Bayes classifier

In [ ]:

```
mnb = MultinomialNB()
mnb.fit(x_train2 ,y_train)
print("Model Score on Training data",mnb.score(x_train2 ,y_train))
print("Model Score on Testing data",mnb.score(x_test2 ,y_test))
y_pred = mnb.predict(x_test2)
print(classification_report(y_pred ,y_test))
```

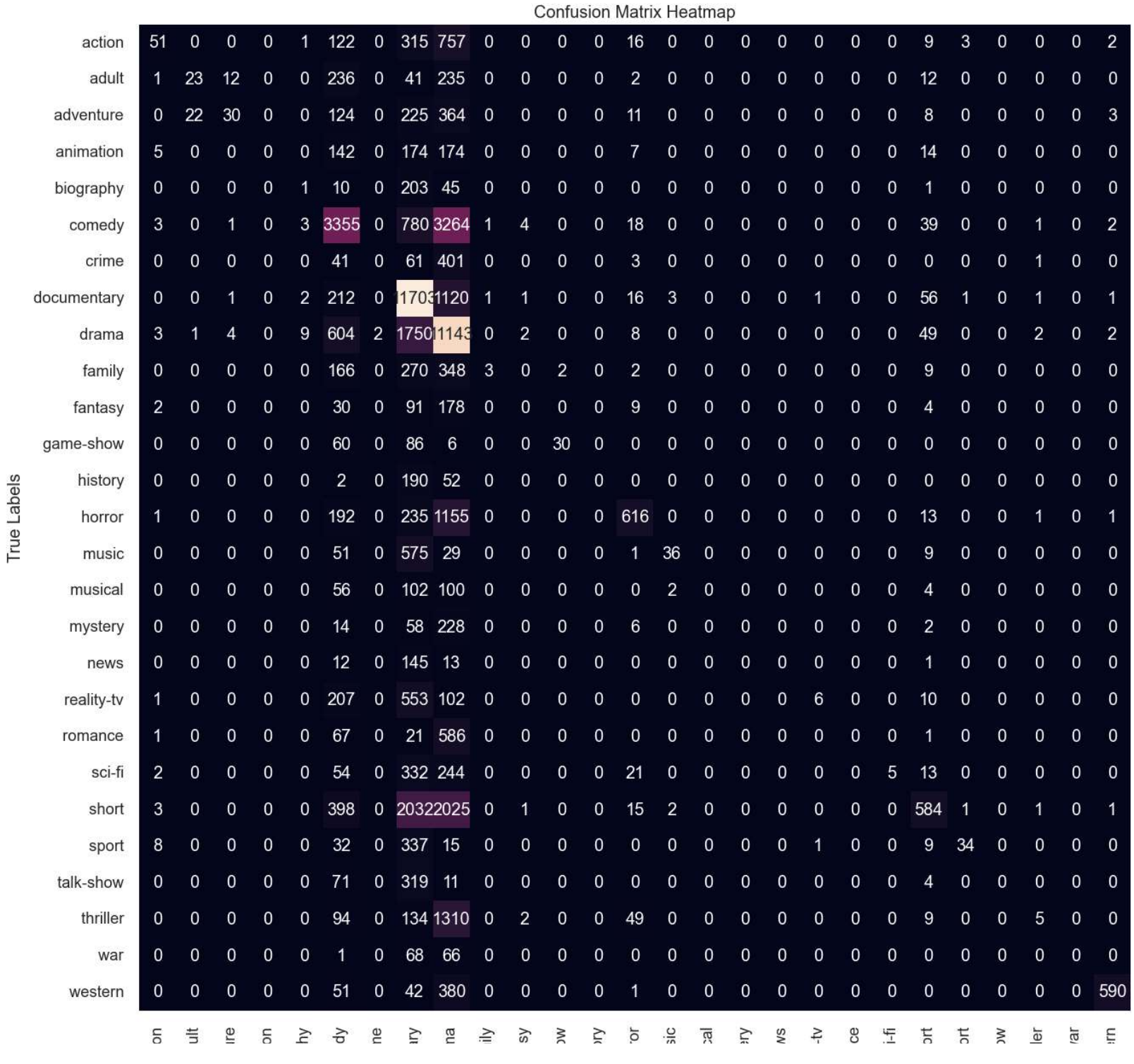
```
Model Score on Training data 0.6288855682845389
Model Score on Training data 0.5205047318611987
              precision    recall  f1-score   support

      action       0.04      0.63      0.08        81
       adult       0.04      0.50      0.08        46
   adventure       0.04      0.62      0.07        48
   animation       0.00      0.00      0.00         0
   biography       0.00      0.06      0.01        16
      comedy       0.45      0.52      0.48      6404
       crime       0.00      0.00      0.00         2
 documentary       0.89      0.56      0.69     20842
       drama       0.82      0.46      0.59     24351
      family       0.00      0.60      0.01         5
     fantasy       0.00      0.00      0.00        10
   game-show       0.16      0.94      0.28        32
     history       0.00      0.00      0.00         0
      horror       0.28      0.77      0.41       801
       music       0.05      0.84      0.10        43
     musical       0.00      0.00      0.00         0
     mystery       0.00      0.00      0.00         0
        news       0.00      0.00      0.00         0
  reality-tv       0.01      0.75      0.01         8
     romance       0.00      0.00      0.00         0
      sci-fi       0.01      1.00      0.01         5
       short       0.12      0.68      0.20       860
       sport       0.08      0.87      0.14        39
   talk-show       0.00      0.00      0.00         0
    thriller       0.00      0.42      0.01        12
         war       0.00      0.00      0.00         0
     western       0.55      0.98      0.71       602

    accuracy                           0.52     54207
   macro avg       0.13      0.41      0.14     54207
weighted avg       0.78      0.52      0.60     54207
```

- **Model Type:** Multinomial Naive Bayes Classifier
- **Training Accuracy:** 62.89%
- **Testing Accuracy:** 52.05%
- **Best Predictions:** Genres like "game-show" (precision: 16%) and "western" (precision: 55%)

- **Challenges:** Model struggles with genres like "action" (precision: 4%) and "family" (precision: 0%). It has difficulty in accurately predicting these categories, indicating a need for improvement, possibly through more data or feature engineering. Overall accuracy is moderate, suggesting room for enhancement in genre predictions.

In [ ]:

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))  # Adjust the figure size as needed
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap
True Labels
action 51 0 0 0 1 122 0 315 757 0 0 0 0 16 0 0 0 0 0 0 0 9 3 0 0 0 2
adult 1 23 12 0 0 236 0 41 235 0 0 0 0 2 0 0 0 0 0 0 0 12 0 0 0 0 0
adventure 0 22 30 0 0 124 0 225 364 0 0 0 0 11 0 0 0 0 0 0 0 8 0 0 0 0 3
animation 5 0 0 0 0 142 0 174 174 0 0 0 0 7 0 0 0 0 0 0 0 14 0 0 0 0 0
biography 0 0 0 0 1 10 0 203 45 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
comedy 3 0 1 0 3 3355 0 780 3264 1 4 0 0 18 0 0 0 0 0 0 0 39 0 0 1 0 2
crime 0 0 0 0 0 41 0 61 401 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 1 0 0
documentary 0 0 1 0 2 212 0 1703 1120 1 1 0 0 16 3 0 0 0 1 0 0 56 1 0 1 0 1
drama 3 1 4 0 9 604 2 1750 1143 0 2 0 0 8 0 0 0 0 0 0 0 49 0 0 2 0 2
family 0 0 0 0 0 166 0 270 348 3 0 2 0 2 0 0 0 0 0 0 0 9 0 0 0 0 0
fantasy 2 0 0 0 0 30 0 91 178 0 0 0 0 9 0 0 0 0 0 0 0 4 0 0 0 0 0
game-show 0 0 0 0 0 60 0 86 6 0 0 30 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
history 0 0 0 0 0 2 0 190 52 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
horror 1 0 0 0 0 192 0 235 1155 0 0 0 0 616 0 0 0 0 0 0 0 13 0 0 1 0 1
music 0 0 0 0 0 51 0 575 29 0 0 0 0 1 36 0 0 0 0 0 0 9 0 0 0 0 0
musical 0 0 0 0 0 56 0 102 100 0 0 0 0 0 2 0 0 0 0 0 0 4 0 0 0 0 0
mystery 0 0 0 0 0 14 0 58 228 0 0 0 0 6 0 0 0 0 0 0 0 2 0 0 0 0 0
news 0 0 0 0 0 12 0 145 13 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
reality-tv 1 0 0 0 0 207 0 553 102 0 0 0 0 0 0 0 0 0 6 0 0 10 0 0 0 0 0
romance 1 0 0 0 0 67 0 21 586 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
sci-fi 2 0 0 0 0 54 0 332 244 0 0 0 0 21 0 0 0 0 0 0 5 13 0 0 0 0 0
short 3 0 0 0 0 398 0 2032 2025 0 1 0 0 15 2 0 0 0 0 0 0 584 1 0 1 0 1
sport 8 0 0 0 0 32 0 337 15 0 0 0 0 0 0 0 0 0 1 0 0 9 34 0 0 0 0
talk-show 0 0 0 0 0 71 0 319 11 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0
thriller 0 0 0 0 0 94 0 134 1310 0 2 0 0 49 0 0 0 0 0 0 0 9 0 0 5 0 0
war 0 0 0 0 0 1 0 68 66 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
western 0 0 0 0 0 51 0 42 380 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 590

acti
ad
adventu
animati
biograp
come
crin
documenta
dran
fam
fanta
game-sh
histo
horr
mus
music
myste
new
reality-
roman
sci
sh
spo
talk-sh
thrill
w
weste
Predicted Labels

LogisticRegression

In [ ]:

```
model = LogisticRegression()
model.fit(x_train2 ,y_train)
print("Model Score on Training data",model.score(x_train2 ,y_train))
print("Model Score on Testing data",model.score(x_test2 ,y_test))
y_pred = model.predict(x_test2)
print(classification_report(y_pred ,y_test))
```

```
Model Score on Training data 0.9925101924105743
Model Score on Testing data 0.5641337834597008
              precision    recall  f1-score   support

      action       0.33      0.38      0.35      1108
       adult       0.40      0.59      0.48       381
   adventure       0.26      0.39      0.31       529
   animation       0.17      0.40      0.24       222
   biography       0.02      0.09      0.03        43
      comedy       0.58      0.52      0.55      8232
       crime       0.12      0.24      0.16       245
 documentary       0.77      0.70      0.74     14432
       drama       0.67      0.57      0.61     16163
      family       0.18      0.31      0.23       474
     fantasy       0.10      0.25      0.14       119
   game-show       0.63      0.75      0.68       152
     history       0.02      0.10      0.03        51
      horror       0.60      0.63      0.61      2108
       music       0.53      0.60      0.57       623
     musical       0.12      0.38      0.18        84
     mystery       0.07      0.20      0.10       103
        news       0.18      0.47      0.26        64
  reality-tv       0.29      0.43      0.35       593
     romance       0.11      0.29      0.16       255
      sci-fi       0.31      0.48      0.37       435
       short       0.37      0.37      0.37      5122
       sport       0.42      0.69      0.52       265
   talk-show       0.33      0.50      0.40       267
    thriller       0.22      0.30      0.25      1151
         war       0.19      0.42      0.26        60
     western       0.77      0.88      0.82       926

    accuracy                           0.56     54207
   macro avg       0.32      0.44      0.36     54207
weighted avg       0.60      0.56      0.58     54207
```

- **Model Type:** Logistic Regression
- **Training Accuracy:** 99.25%
- **Testing Accuracy:** 56.41%
- **Best Predictions:** Genres like "game-show" (precision: 63%) and "western" (precision: 77%)

- **Challenges:** Model struggles with genres like "action" (precision: 33%) and "family" (precision: 18%). It has difficulty in accurately predicting these categories, indicating a need for improvement, possibly through more data or feature engineering. Overall accuracy is moderate, suggesting room for enhancement in genre predictions.

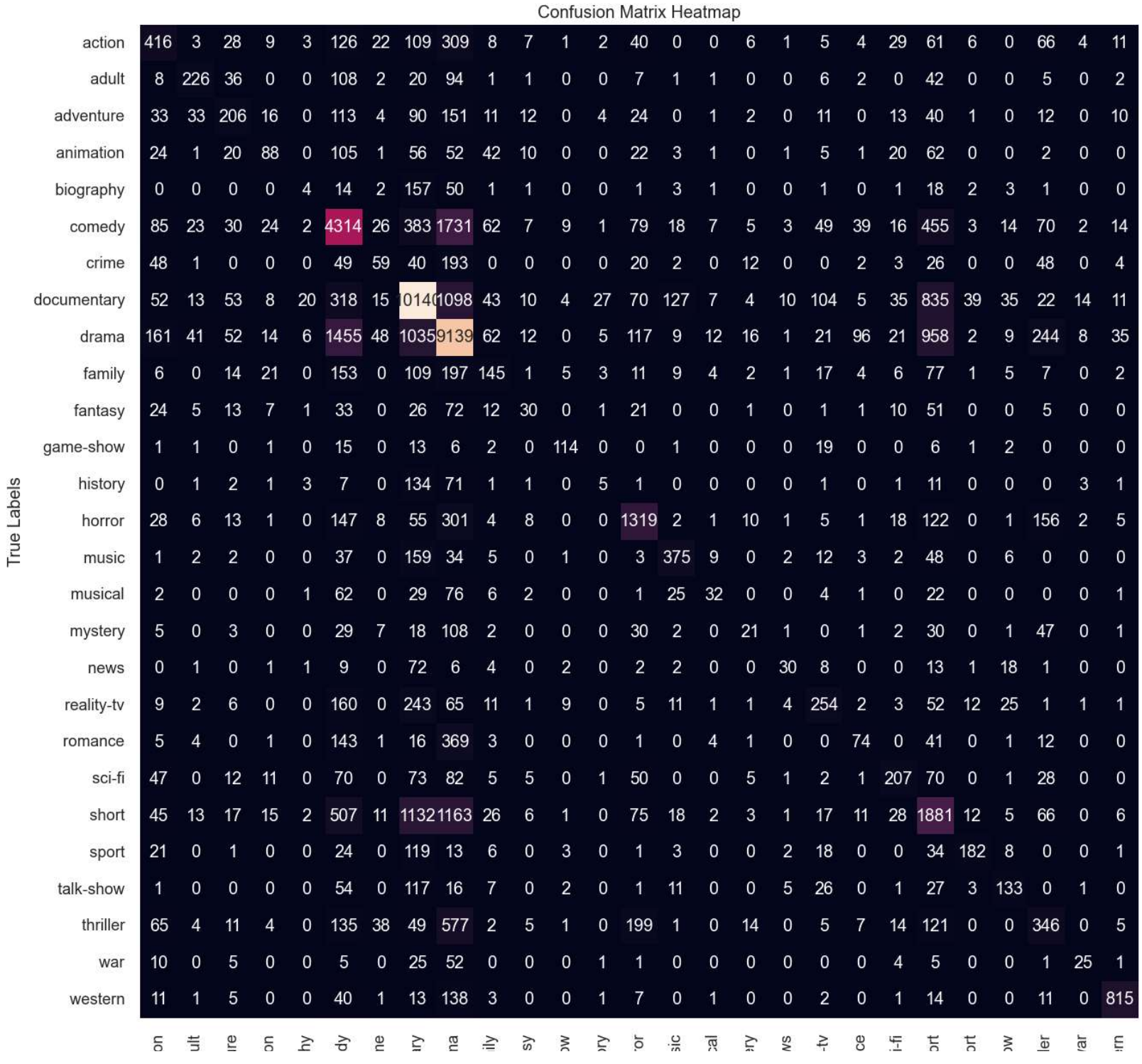
In [ ]:

```
import joblib
# Save the model to a file
joblib.dump((Logistic_Regression, vectorizer), 'logistic_regression_model.pkl')

print("Model and vectorizer saved successfully.")
```

```
Model and vectorizer saved successfully.
```

In [ ]:

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap
True Labels
action 416 3 28 9 3 126 22 109 309 8 7 1 2 40 0 0 6 1 5 4 29 61 6 0 66 4 11
adult 8 226 36 0 0 108 2 20 94 1 1 0 0 7 1 1 0 0 6 2 0 42 0 0 5 0 2
adventure 33 33 206 16 0 113 4 90 151 11 12 0 4 24 0 1 2 0 11 0 13 40 1 0 12 0 10
animation 24 1 20 88 0 105 1 56 52 42 10 0 0 22 3 1 0 1 5 1 20 62 0 0 2 0 0
biography 0 0 0 0 4 14 2 157 50 1 1 0 0 1 3 1 0 0 1 0 1 18 2 3 1 0 0
comedy 85 23 30 24 2 4314 26 383 1731 62 7 9 1 79 18 7 5 3 49 39 16 455 3 14 70 2 14
crime 48 1 0 0 0 49 59 40 193 0 0 0 0 20 2 0 12 0 0 2 3 26 0 0 48 0 4
documentary 52 13 53 8 20 318 15 1014 1098 43 10 4 27 70 127 7 4 10 104 5 35 835 39 35 22 14 11
drama 161 41 52 14 6 1455 48 1035 9139 62 12 0 5 117 9 12 16 1 21 96 21 958 2 9 244 8 35
family 6 0 14 21 0 153 0 109 197 145 1 5 3 11 9 4 2 1 17 4 6 77 1 5 7 0 2
fantasy 24 5 13 7 1 33 0 26 72 12 30 0 1 21 0 0 1 0 1 1 10 51 0 0 5 0 0
game-show 1 1 0 1 0 15 0 13 6 2 0 114 0 0 1 0 0 0 19 0 0 6 1 2 0 0 0
history 0 1 2 1 3 7 0 134 71 1 1 0 5 1 0 0 0 0 1 0 1 11 0 0 0 3 1
horror 28 6 13 1 0 147 8 55 301 4 8 0 0 1319 2 1 10 1 5 1 18 122 0 1 156 2 5
music 1 2 2 0 0 37 0 159 34 5 0 1 0 3 375 9 0 2 12 3 2 48 0 6 0 0 0
musical 2 0 0 0 1 62 0 29 76 6 2 0 0 1 25 32 0 0 4 1 0 22 0 0 0 0 1
mystery 5 0 3 0 0 29 7 18 108 2 0 0 0 30 2 0 21 1 0 1 2 30 0 1 47 0 1
news 0 1 0 1 1 9 0 72 6 4 0 2 0 2 2 0 0 30 8 0 0 13 1 18 1 0 0
reality-tv 9 2 6 0 0 160 0 243 65 11 1 9 0 5 11 1 1 4 254 2 3 52 12 25 1 1 1
romance 5 4 0 1 0 143 1 16 369 3 0 0 0 1 0 4 1 0 0 74 0 41 0 1 12 0 0
sci-fi 47 0 12 11 0 70 0 73 82 5 5 0 1 50 0 0 5 1 2 1 207 70 0 1 28 0 0
short 45 13 17 15 2 507 11 1132 1163 26 6 1 0 75 18 2 3 1 17 11 28 1881 12 5 66 0 6
sport 21 0 1 0 0 24 0 119 13 6 0 3 0 1 3 0 0 2 18 0 0 34 182 8 0 0 1
talk-show 1 0 0 0 0 54 0 117 16 7 0 2 0 1 11 0 0 5 26 0 1 27 3 133 0 1 0
thriller 65 4 11 4 0 135 38 49 577 2 5 1 0 199 1 0 14 0 5 7 14 121 0 0 346 0 5
war 10 0 5 0 0 5 0 25 52 0 0 0 1 1 0 0 0 0 0 0 4 5 0 0 1 25 1
western 11 1 5 0 0 40 1 13 138 3 0 0 1 7 0 1 0 0 2 0 1 14 0 0 11 0 815

acti
ad
adventu
animati
biograp
come
crin
documenta
dran
fam
fanta
game-sh
histo
horr
mus
music
myste
new
reality-
roman
sci
sh
spo
talk-sh
thrill
w
weste
Predicted Labels

Support Vector Machine (SVC)

In [ ]:

```
svm = LinearSVC()
svm.fit(x_train2 ,y_train)
print("Model Score on Training data",svm.score(x_train2 ,y_train))
print("Model Score on Testing data",svm.score(x_test2 ,y_test))
y_pred = svm.predict(x_test2)
print(classification_report(y_pred ,y_test))
```

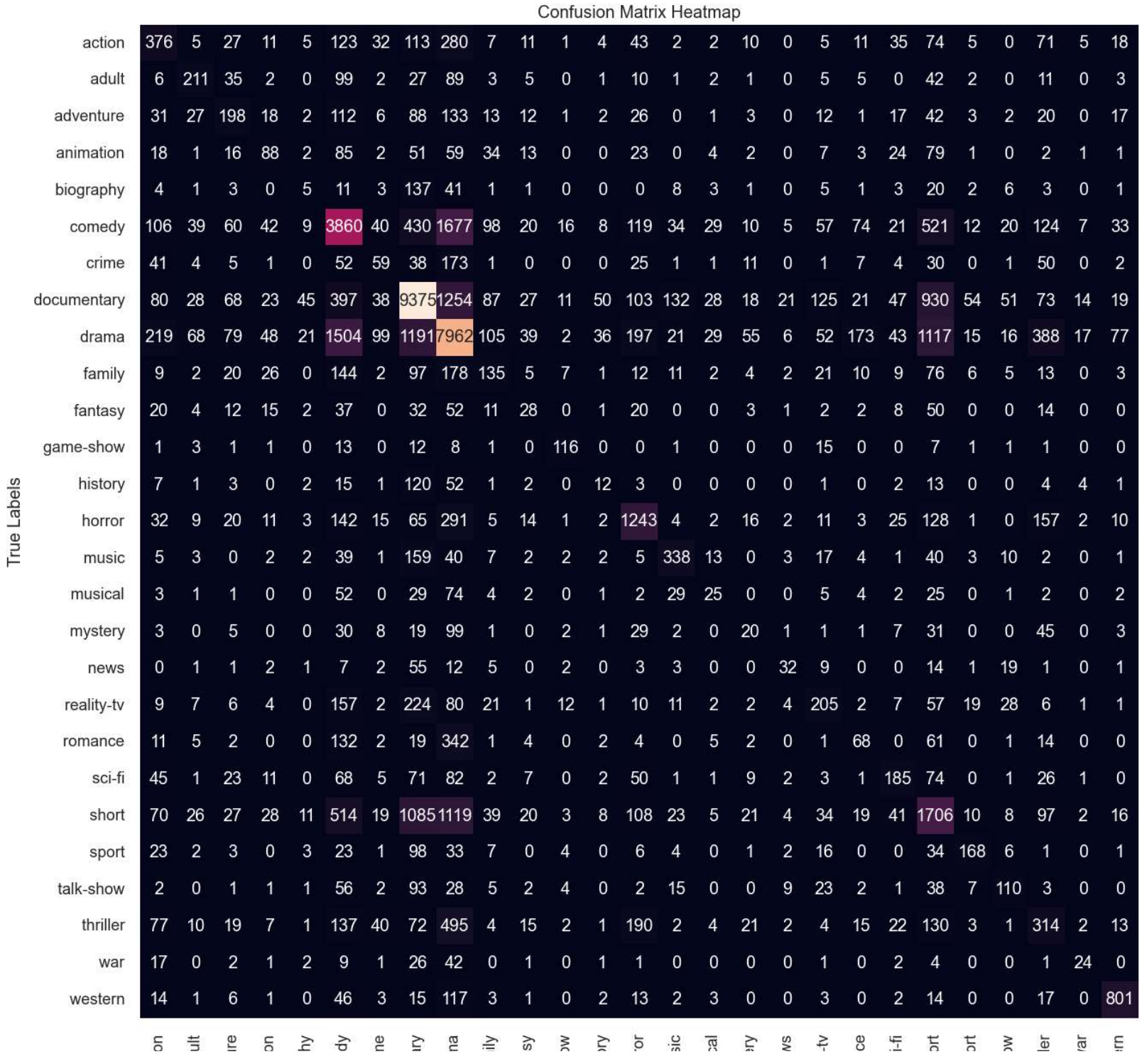
```
Model Score on Training data 0.9996125961591676
Model Score on Testing data 0.5103399929898352
              precision    recall  f1-score   support

      action       0.29      0.31      0.30      1229
       adult       0.38      0.46      0.41       460
   adventure       0.25      0.31      0.28       643
   animation       0.17      0.26      0.20       343
   biography       0.02      0.04      0.03       117
      comedy       0.52      0.49      0.50      7864
       crime       0.12      0.15      0.13       385
 documentary       0.71      0.68      0.70     13741
       drama       0.59      0.54      0.56     14812
      family       0.17      0.22      0.19       601
     fantasy       0.09      0.12      0.10       232
   game-show       0.64      0.62      0.63       186
     history       0.05      0.09      0.06       138
      horror       0.56      0.55      0.56      2247
       music       0.48      0.52      0.50       645
     musical       0.09      0.16      0.12       161
     mystery       0.06      0.10      0.08       210
        news       0.19      0.33      0.24        96
  reality-tv       0.23      0.32      0.27       641
     romance       0.10      0.16      0.12       427
      sci-fi       0.28      0.36      0.31       508
       short       0.34      0.32      0.33      5357
       sport       0.39      0.54      0.45       313
   talk-show       0.27      0.38      0.32       287
    thriller       0.20      0.22      0.21      1460
         war       0.18      0.30      0.22        80
     western       0.75      0.78      0.77      1024

    accuracy                           0.51     54207
   macro avg       0.30      0.35      0.32     54207
weighted avg       0.53      0.51      0.52     54207
```

- **Model Type:** Support Vector Machine (LinearSVC)
- **Training Accuracy:** 99.96%
- **Testing Accuracy:** 51.03%
- **Best Predictions:** Genres like "game-show" (precision: 64%) and "western" (precision: 75%)

- **Challenges:** Model struggles with genres like "action" (precision: 29%) and "family" (precision: 17%). It has difficulty in accurately predicting these categories, indicating a need for improvement, possibly through more data or feature engineering. Overall accuracy is moderate, suggesting room for enhancement in genre predictions.

In [ ]:

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap
True Labels
action 376 5 27 11 5 123 32 113 280 7 11 1 4 43 2 2 10 0 5 11 35 74 5 0 71 5 18
adult 6 211 35 2 0 99 2 27 89 3 5 0 1 10 1 2 1 0 5 5 0 42 2 0 11 0 3
adventure 31 27 198 18 2 112 6 88 133 13 12 1 2 26 0 1 3 0 12 1 17 42 3 2 20 0 17
animation 18 1 16 88 2 85 2 51 59 34 13 0 0 23 0 4 2 0 7 3 24 79 1 0 2 1 1
biography 4 1 3 0 5 11 3 137 41 1 1 0 0 0 8 3 1 0 5 1 3 20 2 6 3 0 1
comedy 106 39 60 42 9 3860 40 430 1677 98 20 16 8 119 34 29 10 5 57 74 21 521 12 20 124 7 33
crime 41 4 5 1 0 52 59 38 173 1 0 0 0 25 1 1 11 0 1 7 4 30 0 1 50 0 2
documentary 80 28 68 23 45 397 38 9375 1254 87 27 11 50 103 132 28 18 21 125 21 47 930 54 51 73 14 19
drama 219 68 79 48 21 1504 99 1191 7962 105 39 2 36 197 21 29 55 6 52 173 43 1117 15 16 388 17 77
family 9 2 20 26 0 144 2 97 178 135 5 7 1 12 11 2 4 2 21 10 9 76 6 5 13 0 3
fantasy 20 4 12 15 2 37 0 32 52 11 28 0 1 20 0 0 3 1 2 2 8 50 0 0 14 0 0
game-show 1 3 1 1 0 13 0 12 8 1 0 116 0 0 1 0 0 0 15 0 0 7 1 1 1 0 0
history 7 1 3 0 2 15 1 120 52 1 2 0 12 3 0 0 0 0 1 0 2 13 0 0 4 4 1
horror 32 9 20 11 3 142 15 65 291 5 14 1 2 1243 4 2 16 2 11 3 25 128 1 0 157 2 10
music 5 3 0 2 2 39 1 159 40 7 2 2 2 5 338 13 0 3 17 4 1 40 3 10 2 0 1
musical 3 1 1 0 0 52 0 29 74 4 2 0 1 2 29 25 0 0 5 4 2 25 0 1 2 0 2
mystery 3 0 5 0 0 30 8 19 99 1 0 2 1 29 2 0 20 1 1 1 7 31 0 0 45 0 3
news 0 1 1 2 1 7 2 55 12 5 0 2 0 3 3 0 0 32 9 0 0 14 1 19 1 0 1
reality-tv 9 7 6 4 0 157 2 224 80 21 1 12 1 10 11 2 2 4 205 2 7 57 19 28 6 1 1
romance 11 5 2 0 0 132 2 19 342 1 4 0 2 4 0 5 2 0 1 68 0 61 0 1 14 0 0
sci-fi 45 1 23 11 0 68 5 71 82 2 7 0 2 50 1 1 9 2 3 1 185 74 0 1 26 1 0
short 70 26 27 28 11 514 19 1085 1119 39 20 3 8 108 23 5 21 4 34 19 41 1706 10 8 97 2 16
sport 23 2 3 0 3 23 1 98 33 7 0 4 0 6 4 0 1 2 16 0 0 34 168 6 1 0 1
talk-show 2 0 1 1 1 56 2 93 28 5 2 4 0 2 15 0 0 9 23 2 1 38 7 110 3 0 0
thriller 77 10 19 7 1 137 40 72 495 4 15 2 1 190 2 4 21 2 4 15 22 130 3 1 314 2 13
war 17 0 2 1 2 9 1 26 42 0 1 0 1 1 0 0 0 0 1 0 2 4 0 0 1 24 0
western 14 1 6 1 0 46 3 15 117 3 1 0 2 13 2 3 0 0 3 0 2 14 0 0 17 0 801

acti
ad
adventu
animati
biograp
come
crin
documenta
dran
fam
fanta
game-sh
histo
horr
mus
music
myste
new
reality-
roman
sci
sh
spo
talk-sh
thrill
w
weste
Predicted Labels

RandomForestClassifier

In [ ]:

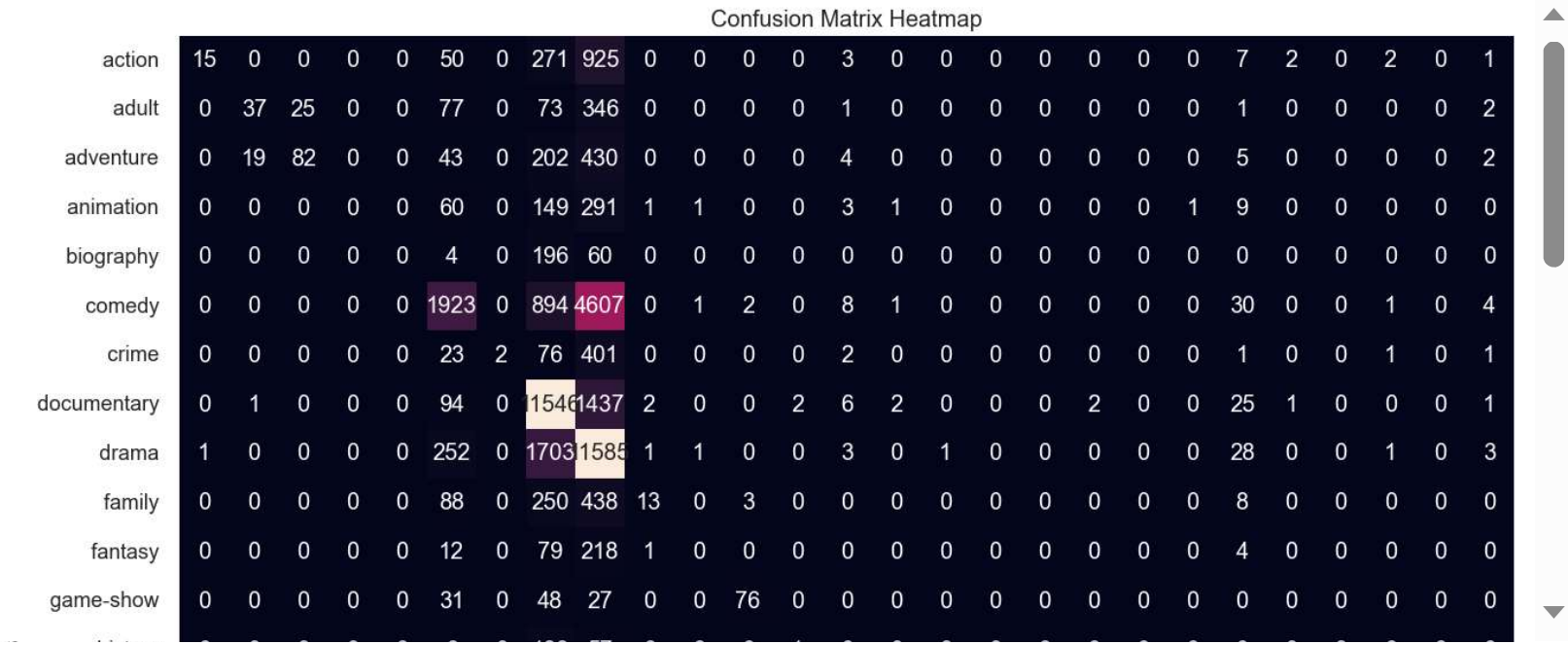
```
# Create a Random Forest model
random_forest = RandomForestClassifier()


random_forest.fit(x_train2, y_train)


print("Random Forest - Train Score:",random_forest.score(x_train2, y_train))
print("Random Forest - Test Score:", random_forest.score(x_test2, y_test))

y_pred = random_forest.predict(x_test2)
print(classification_report(y_pred ,y_test))
```

```
Random Forest - Train Score: 0.9996494917630564
Random Forest - Test Score: 0.48707731473794896
              precision    recall  f1-score   support

      action       0.01      0.68      0.02        22
       adult       0.07      0.64      0.12        58
   adventure       0.10      0.77      0.18       107
   animation       0.00      0.00      0.00         1
   biography       0.00      0.00      0.00         1
      comedy       0.26      0.58      0.36      3304
       crime       0.00      1.00      0.01         2
 documentary       0.88      0.56      0.69     20574
       drama       0.85      0.40      0.55     28631
      family       0.02      0.59      0.03        22
     fantasy       0.00      0.00      0.00         3
   game-show       0.42      0.90      0.57        84
     history       0.00      0.33      0.01         3
      horror       0.10      0.78      0.18       283
       music       0.12      0.91      0.22        94
     musical       0.02      0.75      0.04         8
     mystery       0.00      1.00      0.01         1
        news       0.00      0.00      0.00         0
  reality-tv       0.00      0.57      0.01         7
     romance       0.01      1.00      0.01         4
      sci-fi       0.01      0.71      0.01         7
       short       0.07      0.68      0.12       497
       sport       0.06      0.90      0.12        31
   talk-show       0.02      0.90      0.04        10
    thriller       0.01      0.47      0.01        19
         war       0.02      1.00      0.04         3
     western       0.39      0.95      0.55       431

    accuracy                           0.49     54207
   macro avg       0.13      0.63      0.14     54207
weighted avg       0.81      0.49      0.58     54207
```

- **Model Type:** Random Forest Classifier
- **Training Accuracy:** 99.96%
- **Testing Accuracy:** 48.71%
- **Best Predictions:** Genres like "game-show" (precision: 42%) and "western" (precision: 39%)

- **Challenges:** Similar to other models, this Random Forest model struggles with genres like "action" (precision: 1%) and "family" (precision: 2%). It also has difficulty predicting categories like "thriller" (precision: 1%) and "animation" (precision: 0%). The overall accuracy is relatively low, indicating the need for improvements in genre predictions, possibly through more data or feature engineering.

In [ ]:
```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```



Conclusion:

After rigorous evaluation of various models trained using both CountVectorizer and TF-IDF Vectorization techniques, the Logistic Regression model stands out as the most promising choice:

Logistic Regression Model:

- **CountVectorizer:** Achieved a high training accuracy of 99.25% and a testing accuracy of 56.41%. Successfully predicted genres like "game-show" and "western" with precision rates of 63% and 77%, respectively. Faces challenges in predicting niche categories such as "action" and "family."
- **TF-IDF Vectorization:** Maintained comparable performance with a training accuracy of 99.25% and a testing accuracy of 56.41%. Similar challenges in predicting genres like "action" and "family" were observed, indicating room for improvement.

Conclusion: Considering both techniques, the Logistic Regression model consistently demonstrates the best overall performance. While challenges persist in predicting specific genres, the Logistic Regression model shows the highest testing accuracy and reasonable precision for key genres. Further enhancements through feature engineering and additional data could refine its predictions, making it the most promising choice for genre classification tasks.

Thank You! 😊🙌

We appreciate your time and attention!

In [ ]: