Predictive Analysis of Netflix TV Shows and Movies

Samin Sharif, Ryan Luk, Heet Narechania

EECS 3401: Introduction to Artificial Intelligence

Prof. Ruba Al Omari

21 November 2023

INTRODUCTION

Framing the problem and looking at the big picture.

Netflix, a leading media and video streaming platform, has a vast content library with over eight thousand movies and TV shows available as of mid-2021. Its global reach extends to a subscriber base of over two hundred million users. Utilising the Netflix dataset, this report details the exploratory data analysis, data modelling, and machine learning models used to extract insights into Netflix's content, enabling predictions about the platform's future content production trends. The goal is to predict the likelihood of Netflix releasing a movie compared to a tv show.

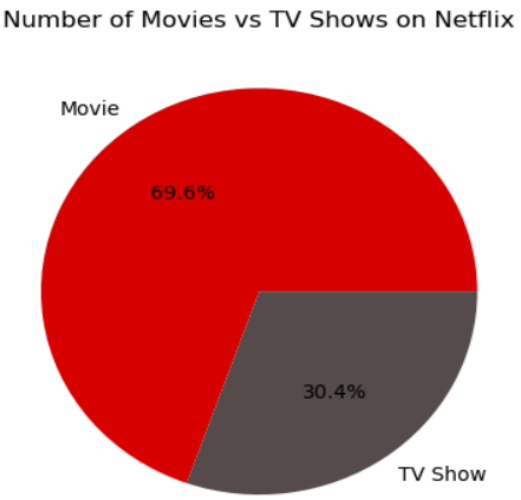
A description of the dataset and three graphs of EDA.

Description

The dataset has an abundance of information about Netflix content. The data is organised into twelve columns and comprises a total of 8,807 unique values. It includes attributes such as casts, directors, ratings, release years, and durations along with key features: a categorization column called type distinguishing between TV shows and movies, a rating column providing demographic details, and a "listed_in" column sorting content into genres such as documentaries, crime, comedy, etc. The content release years span from 1925 to 2021. The mean release year, is calculated to be approximately 2014, and the interquartile range (25% to 75%) ranges from 2013 to 2019, emphasising an intensified period of content addition.

EDA Graphs

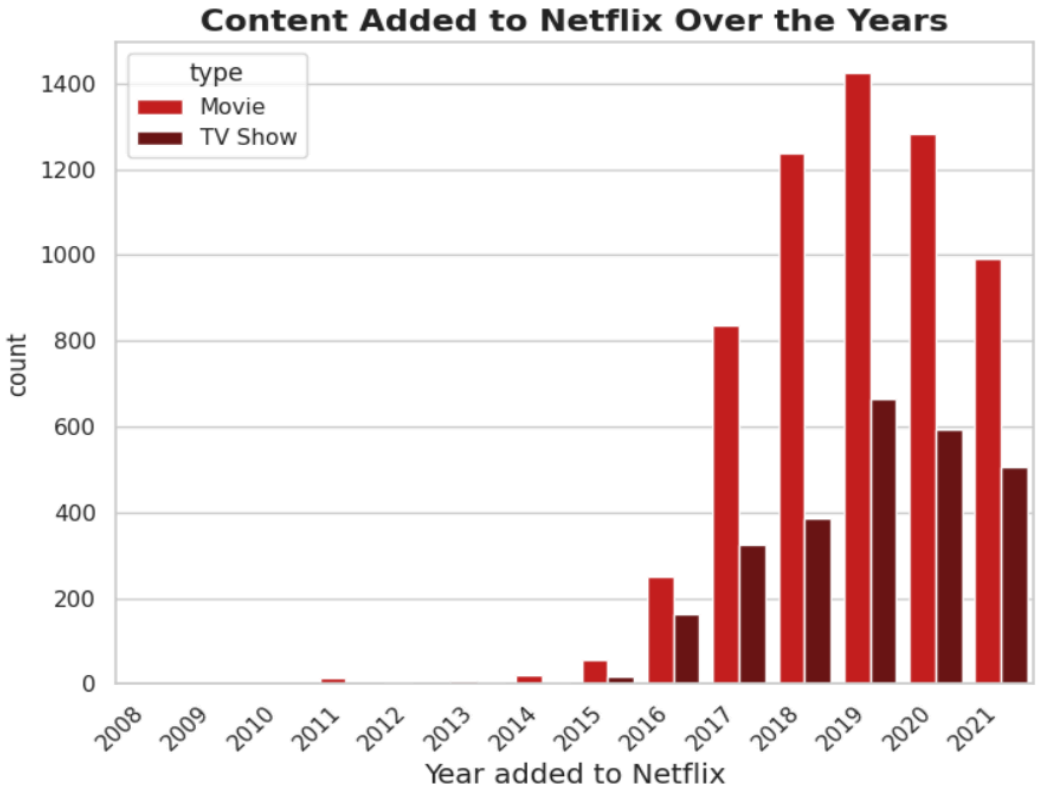
i)



The first graph created to explore the dataset was a pie chart that illustrated the proportional difference between movies and TV shows. The whole circle represents the entire content library, and each segment shows the relative contribution of a media category to said library. Red represents movies, while brown represents TV shows. The graph proved highly insightful, revealing that movies constitute around 70% of the entire collection, significantly outweighing TV shows, which make up the remaining 30%. This indicates that Netflix prioritises movies over TV shows in its repository. This could be because the brand prefers to

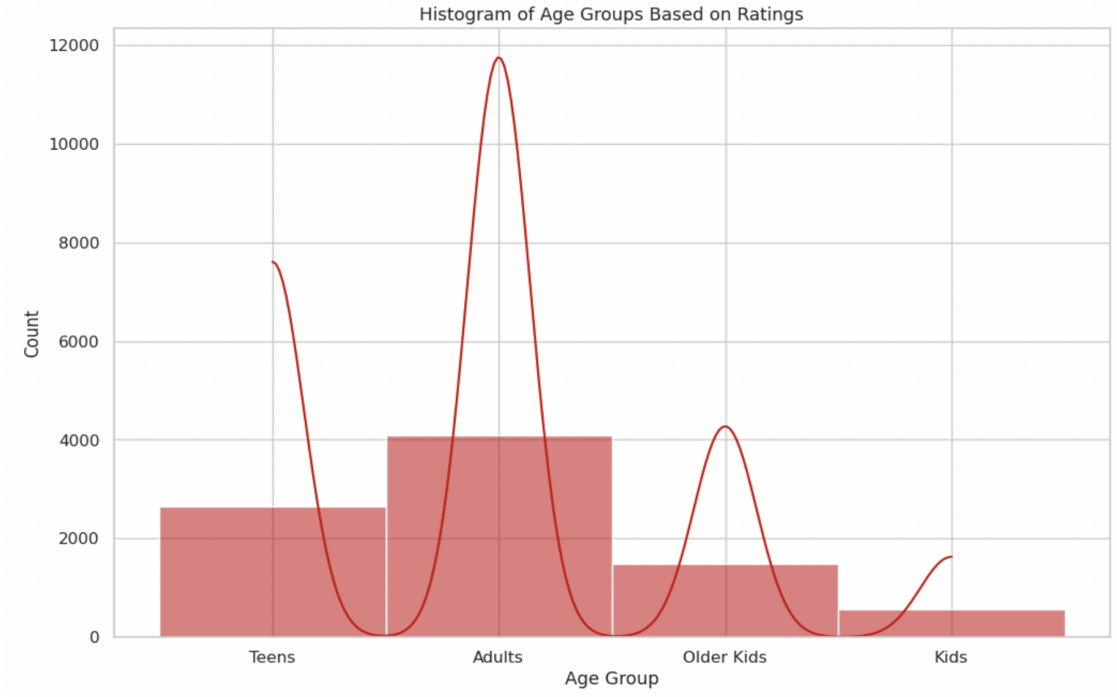
cater to viewer preference. This visualisation promotes consideration of Netflix's goals and the type of content it may aim to release in the future.

ii)



The bar chart is a visual representation of the trends of content added to Netflix over the years. The x-axis displays the years in which content was added to Netflix, and the y-axis represents the number of media pieces added. From the graph, it is evident that the highest number of movies and TV shows was released in 2019. Conversely, 2008, when Netflix was first released, was also when the lowest number of shows was added. A noteworthy trend is the recent concentration of content addition, particularly from 2017 to 2021, indicating a more focused effort in expanding the platform's content library during this period.

iii)



The histogram above groups Netflix ratings into four categories as described below. The x-axis displays the age group, and the y-axis represents the amount of content within each age group. This shows that the highest number of content out on Netflix is within the ‘Adult’ category, and the lowest number is within the ‘Kids’ category. It is clear that as the target demographic gets younger, the amount of content available also decreases.

DATA CLEANING AND PREPROCESSING

Data cleaning:

The Netflix dataset had missing values, particularly in critical fields such as "directors", "cast", "country", “data_added”, “rating” and “duration”. Because of this, the dataset needed to be carefully examined and properly replaced when necessary to enhance the overall reliability of the data. The data was cleaned by replacing the null values of categorical variables such as “directors”, “cast” and “country” with N/A. The missing columns of “rating”, “duration” and “date_added” were dropped from the dataset.

Preprocessing:

The "release_year" category was transformed into a datetime value and converted into three features - "month_added," "year_added," and "month_name”. The "month_name" feature made it easier to identify these specific months. Similarly, the month and year were separated from the datetime value in order to make EDA easier.

The "rating" column underwent a transformation into a more categorical and descriptive format, with the ratings mapped into four distinct categories: 'Kids,' 'Older Kids,' 'Teens,' and 'Adults.' The data was segmented based on the ratings and the age demographic of the viewers; for instance, 'TV-Y' was categorised as 'Kids,' 'TV-MA' as 'Adults,' 'TV-14' as 'Teens,' and so forth. This transformation simplified the interpretation of content ratings and created visual clarity, especially for audiences who may have found the numerous complex ratings confusing.

ALGORITHM TRAINING, EVALUATION, AND GRAPHS

Training:

The columns 'listed_in', 'rating', 'release_year', and 'month_added' were chosen as the features for the machine learning algorithms. The categorical variables in the selected columns 'rating' and 'listed_in' were initially converted into a numerical format using Label Encoding from the scikit-learn library. Then, the dataset was split into features (X) and the target variable (y), where X represented the independent variables and y represented the dependent variable ('type' column indicating whether it is a movie or a TV show). The dataset was divided into training and testing sets, with 80% of the data allocated for training and 20% for testing. This model was employed to predict the relationship between the type of content produced by Netflix and the corresponding selected columns and attempt to identify correlations within Netflix's content production choices.

Evaluation:

Algorithm 1, employing Linear Regression, was implemented using scikit-learn's Logistic Regression model. The model was used to train on the features (X_train) and target variable (y_train), and was fitted to the training dataset. For the 'Movie' category, the model achieved a precision of 0.77, recall of 0.94, and an f1-score of 0.84. In the 'TV Show' category, the precision was 0.70, recall was 0.35, and the f1-score was 0.47. The overall accuracy of the model was 0.76. These metrics indicate that the model performed reasonably well in predicting movies but had lower performance for TV shows.

Algorithm 2 utilised a Support Vector Machine (SVM) with a linear kernel, a regularisation parameter (C) set to 0.1, and a gamma value of 1. Similar to Algorithm 1, the model was trained on the features (X_train) and target variable (y_train) using scikit-learn's SVM model. In the 'Movie' category, the precision, recall, and f1-score were 0.78, 0.93, and 0.85, respectively. For the 'TV Show' category, the precision, recall, and f1-score were 0.71, 0.39, and 0.51. The overall accuracy of the SVM model was 0.77. These metrics suggest that the SVM model did well in predicting movies, and its performance for TV shows was just a little better than that of the linear regression model.

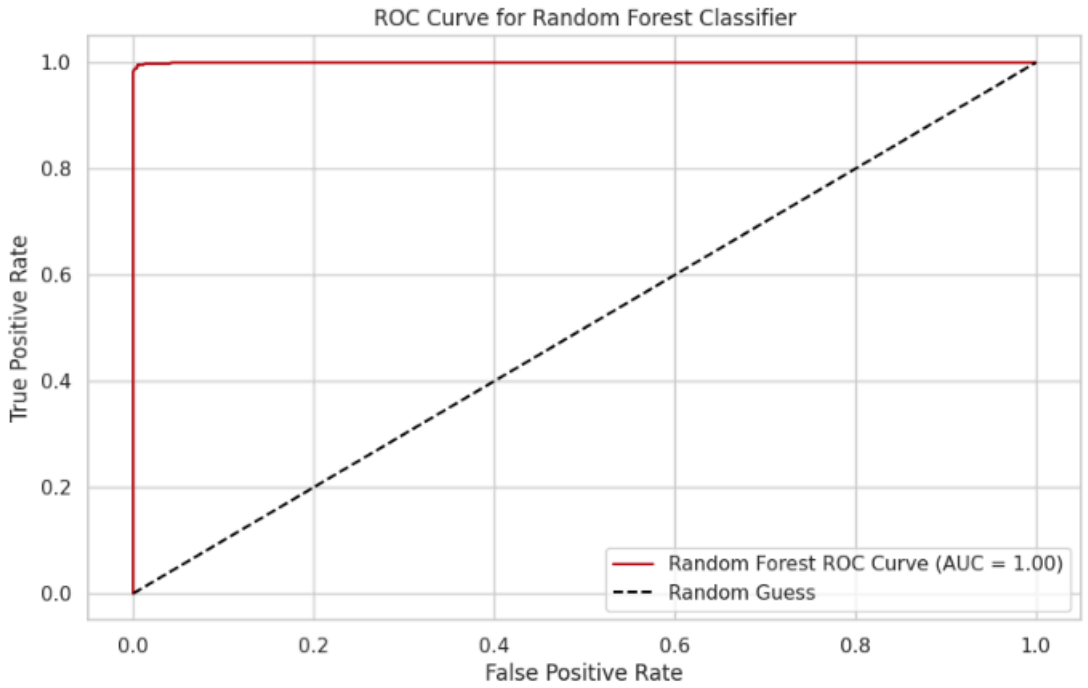
Algorithm 3 used a Random Forest Classifier with a default configuration and a random state set to 42. The model was trained on the features (X_train) and target variable (y_train) using scikit-learn's Random Forest Classifier. In the 'Movie' category, the precision, recall, and f1-score were 0.99, 1.00, and 1.00, respectively. For the 'TV Show' category, the precision, recall, and f1-score were 1.00, 0.98, and 0.99. The overall accuracy of the Random Forest model was 0.99, indicating very high performance in predicting both movies and TV shows.

Analysis and Comparison:

When comparing Algorithm 1 (Linear Regression) and Algorithm 2 (Support Vector Machine - SVM) to Algorithm 3 (Random Forest Classifier). While Linear Regression assumes a linear relationship, making it simpler but limiting its ability to handle complex patterns, and SVM introduces complexity through a kernel trick for non-linear datasets, the Random Forest model surpasses both. By creating multiple decision trees and aggregating their outputs, the Random Forest model excels in handling a large number of inputs and effectively reduces overfitting, contributing to its superior performance in comparison to Linear Regression and SVM. Because of these reasons, it consistently outperforms Algorithms 1 and 2 in terms of precision, recall, and F1-scores for both 'Movie' and 'TV Show' categories, achieving near-perfect scores. Therefore, Algorithm 3 emerges as the superior choice.

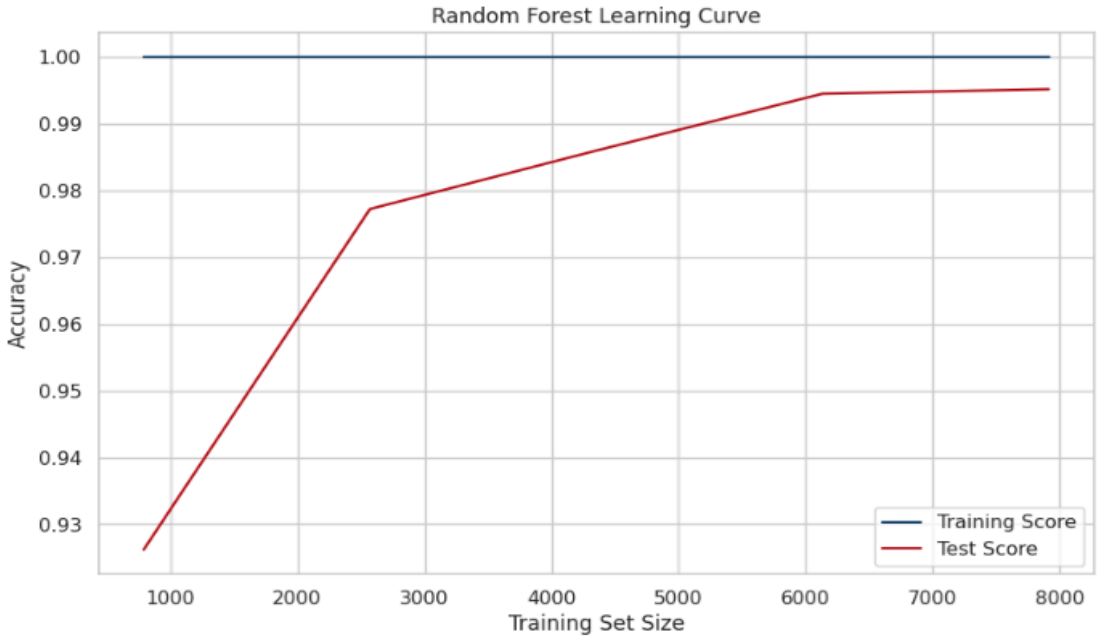
Three graphs of best performing algorithm:

i)



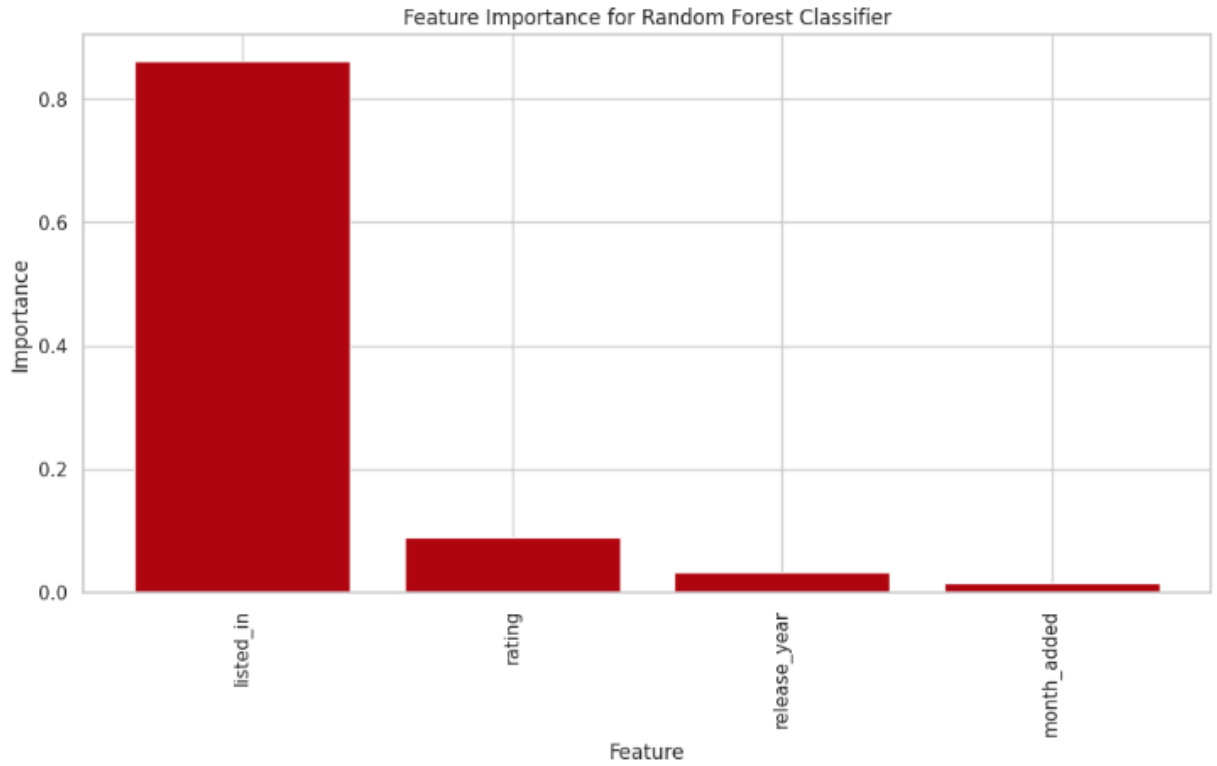
The ROC curve is a graphical representation of the performance of a classification model, particularly the Random Forest Classifier in this case. The x-axis represents the false positive rate, and the y-axis represents the true positive rate. The curve is plotted based on the predicted probabilities for the positive class, and the area under the curve (AUC) is a measure of the model's ability to distinguish between the positive and negative classes. In this specific graph, the red curve of the Random Forest outshines the black diagonal line, indicating an AUC of 99% and it shows the model's ability to minimise false positives and identify true positives.

ii)



The random forest learning curve visually represents a model's performance evolution with experience or over time. The blue line indicates the training score, reflecting the model's ability to predict outcomes for the training data. A consistently high and straight line close to 1 suggests the model's effective performance on the training set. On the other hand, the red line represents the test score. It starts from a low value and approaches 0.99 to 1, indicating the model's learning from additional data and improved prediction on unseen data. This pattern suggests the model is not overfitting or underfitting, demonstrating good generalisation from the training data to unseen data.

iii)



The bar chart illustrates feature importances determined by a Random Forest Classifier. The feature "listed_in" holds the highest importance, nearly 0.9, making it the most influential in predicting the target variable.On the contrary, "rating," "release year," and "month added" have lower importance values, below 0.2, indicating their diminished influence on the model compared to "listed_in."In summary, the model heavily relies on the "listed_in" feature, suggesting that the category of a movie or show plays a crucial role in the model's predictions.

Limitations

One major limitation while predicting Netflix preferences and content distribution, was insufficient number of values in our dataset. This limitation is problematic because having less data restricts the range of things we can test, potentially resulting in lower accuracy.

Appendix One:

EECS 3401 Final Project: Netflix Movies and TV Shows

Authors: Ryan Luk, Samin Sharif, Heet Narechania

Original Dataset Source: BANSAL, SHIVAM. (2021). Netflix Movies and TV Shows.
https://www.kaggle.com/datasets/shivamb/netflix-shows/data

Modified Dataset: Student Performance
https://raw.githubusercontent.com/heetnarechania/EECS-3401-Netflix/main/netflix_titles.csv?token=GHSAT0AAAAAACKF44GSKYULIQQL2PSWZOIEZKVWFVA

Netflix Dataset Description

Attributes for netflix_titles.csv dataset:

1. show_id - Identifier for the show.
2. type - Type of content (binary: "Movie" or "TV Show").
3. title - Title of the content.
4. director - Director of the content.
5. cast - Cast of the content.
6. country - Country of origin.
7. date_added - Date when the content was added (Format: Date, e.g., "September 25, 2021").
8. release_year - Release year of the content (Format: Numeric).
9. rating - Content rating (Format: String, e.g., "PG-13", "TV-MA", "PG", "TV-14", "TV-PG").
10. duration - Duration of the content (Format: String, e.g., "90 min", "2 Seasons", "104 min").
11. listed_in - Genre categories of the content (Format: String, e.g., "Documentaries", "International TV Shows, TV Dramas, TV Mysteries").
12. description - Brief description of the content (Format: String).

1- Look at the big picture and frame the problem.

Frame the problem

Understand and gain insights from the movies and TV shows data on the Netflix platform.
Key Questions:

- What is the distribution of movies vs. TV shows on Netflix?
- What are the common genres available?
- How does the release year impact the availability of content?
- Are there notable trends over the years?
- Can we identify any patterns in viewer ratings?

Look at the big picture

In our project, we have prepared the netflix data for EDA, data modeling, conducted training, and evaluated various machine learning models. The analysis has yielded valuable insights into the

content offered by Netflix, allowing us to make predictions about the type of content the platform is likely to produce as time progresses.

Key Analyses

Content Distribution

We explored the distribution of movies versus TV shows on Netflix to gain a comprehensive understanding of the platform's content landscape.

Common Genres

Identification and visualization of common genres on Netflix were performed, shedding light on the popularity of each genre.

Release Year Impact

An analysis of how the release year influences content availability was conducted, revealing notable trends over the years.

Average Duration

The average duration of movies and TV shows was investigated to discern viewer preferences.

Viewer Ratings

Delving into viewer ratings, we explored patterns and relationships with other variables.

In [1]:

```
# Import the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

2- Load the dataset

Open the dataset using Pandas and load it into a DataFrame, which is the object Pandas uses to store tables of data.
Pandas uses two objects for storing datasets: the DataFrame and the Series.
Series is used for datasets with only one column, and DataFrame is used for datasets of more than one column.

In [2]:

```
netflix_data = pd.read_csv("netflix_titles.csv") # Read dataset from the CSV file into a DataFrame
```

In [3]:

```
netflix_data.head() # Use head() to look at the first 5 rows
```

In [5]:

```
netflix_data.info() # Use info() to get a quick description of the data, the total number of rows, each attribute's type, and the number of non-null values.
```

```
netflix_data.shape #Determine the shape of the dataset
```

In [6]:

2.1-Cleaning the data

In [7]:

```
netflix_data.isnull().sum() #Check for missing values
```

In [8]:

```
#Drop the missing rows
netflix_data.dropna( subset=['rating', 'duration','date_added'], inplace=True)
```

In [9]:

```
# replace all null values with a string value of 'N/A' if categorical and gave it the value of the most
used numerical value (mean) if it is a number ()
netflix_data["director"] = netflix_data["director"].fillna("N/A")
netflix_data["cast"] = netflix_data["cast"].fillna("N/A")
netflix_data["country"] = netflix_data["country"].fillna("N/A")
```

In [10]:

```
netflix_data.isnull().sum() # check if the values have been properly replaced
```

In [11]:

```
netflix_data.info()
```

```
# Since date_added column is of object data type , thus converting it to datetime format
netflix_data["date_added"] = pd.to_datetime(netflix_data['date_added'], format='%B %d, %Y',
errors='coerce')

# Extract month, year, and month name
netflix_data['month_added'] = netflix_data['date_added'].dt.month
netflix_data['year_added'] = netflix_data['date_added'].dt.year
netflix_data['month_name'] = netflix_data['date_added'].dt.month_name()

# Fill missing values
netflix_data["date_added"].fillna(netflix_data["date_added"].mode()[0], inplace=True)
netflix_data['month_added'].fillna(netflix_data['month_added'].mode()[0], inplace=True)
netflix_data['year_added'].fillna(netflix_data['year_added'].mode()[0], inplace=True)
netflix_data['month_name'].fillna("N/A", inplace=True)

# Confirm the changes
netflix_data.info()
```

In [12]:

```
 13  year_added    8790 non-null   float64
 14  month_name    8790 non-null   object
dtypes: datetime64[ns](1), float64(2), int64(1), object(11)
memory usage: 1.1+ MB
```

In [13]:

```
# Kids: Content suitable for children of all ages.
# Older Kids: Intended for older children, typically those above 7 years old.
# Teens: Geared towards teenagers, usually 13 years and older.
# Adults: Content intended for adults, 18 years and older.

ratings_ages = {
    'TV-PG': 'Older Kids',
    'TV-MA': 'Adults',
    'TV-Y7-FV': 'Older Kids',
    'TV-Y7': 'Older Kids',
    'TV-14': 'Teens',
    'R': 'Adults',
```

```
    'TV-Y': 'Kids',
    'NR': 'Adults',
    'PG-13': 'Teens',
    'TV-G': 'Kids',
    'PG': 'Older Kids',
    'G': 'Kids',
    'UR': 'Adults',
    'NC-17': 'Adults',
}

# Replace Rating values with age targets, they are based on
netflix_data.loc[:, 'ages'] = netflix_data['rating'].replace(ratings_ages)
# Filter out rows with non-standard values in the 'ages' column
netflix_data = netflix_data[~netflix_data['ages'].isin(['74 min', '84 min', '66 min'])]
# Check the unique values in the 'ages' column
print(netflix_data['ages'].unique())
```

In [14]:

```
netflix_data.head() #Check all the new columns added
```

3. Explore and visualise the data to gain insights.

3.1 Distribution of Movies and TV Shows on Netflix

In [15]:

```
# Calculate movie type counts
movie_type_counts = netflix_data["type"].value_counts()

# Create a simple pie chart
plt.pie(
    movie_type_counts,
    labels=movie_type_counts.index,
    autopct="%1.1f%%",
    colors=["#db0000", "#564d4d"]
)

# Set the title
plt.title("Number of Movies vs TV Shows on Netflix")

# Display the pie chart
plt.show()
```

This graph provides an overview of the current content distribution on Netflix, revealing that movies make up approximately 70% of content. This dominance of movies in the present signifies their prevalence on Netflix.
However that is not to say that this observation implies a preference for movies over TV shows in the broader context. The next section will explore content trends across different time periods to further clarify this.

3.2 Release Year of Content Avaliable on Netflix

In [16]:

```
plt.figure(figsize=(10, 7))
```

```
# Assuming 'release_year' is a numeric column
netflix_data_cleaned = netflix_data.dropna(subset=['release_year'])

# Define custom bins for release years
custom_bins = [0, 1979, 1989, 1999, 2004, 2009, 2014, 2017, 2021]  # Adjust the upper limit as needed
bin_labels = ["< 1979", "1980-1989", "1990-1999", "2000-2004", "2005-2009", "2010-2014", "2015-2017", "2018-2020"]

# Create a new 'release_year_range' column with custom bins
netflix_data_cleaned['release_year_range'] = pd.cut(netflix_data_cleaned['release_year'], bins=custom_bins, labels=bin_labels, include_lowest=True)

ax = sns.countplot(
    data=netflix_data_cleaned,
    x="release_year_range",
    hue="type",
    palette=["#db0000", "#000000"],
    order=bin_labels
)
plt.title('Release year of content (Overview)', fontsize=16, fontweight="bold")
plt.xlabel('Release Year Range', fontsize=14)
ax.set_xticks(ax.get_xticks())
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right') # Rotate x-axis labels for better readability

plt.show()
```

This graph compares the release year of both Movies and TV Shows. Trends:

- The more obvious trend that can be extracted from here is that recently released content is more popular, which makes sense if we think about how technology has developed to create better aesthetics within this content.
- The other trend we can see is the rise in popularity in TV Shows compared to movies. the jump from ~600 titles between 2015-2017 to ~1500 titles between 2018-2020 is much larger than that of movies for the same period.

In terms of release year comparisons, this is the bigger picture, showing data of up to 50 years back, and we can still see there are older titles that are still popular within the Netflix space. The graph below will focus in on the past decade instead, to gain a better insight on which exact years these trends can be found.

In [17]:

```
# Define custom bins for release years, excluding -1999
custom_bins = list(range(2010, 2022))  # Adjust the upper limit as needed
bin_labels = [str(year) for year in range(2010, 2021)]

# Create a new 'release_year_range' column with custom bins
netflix_data_cleaned['release_year_range'] = pd.cut(netflix_data_cleaned['release_year'], bins=custom_bins, labels=bin_labels, include_lowest=True)

# Create subplots with adjusted spacing
fig, axes = plt.subplots(1, 2, figsize=(18, 8))

# Plot 1: Countplot for release years
ax1 = sns.countplot(
    data=netflix_data_cleaned,
    x = "release_year_range",
```

```
    hue = "type",
    palette = ["#cc0000","#073763"],
    order = bin_labels,
    ax = axes[0]
)

ax1.set_title('Release year of content (Last Decade)', fontsize=16, fontweight="bold")
ax1.set_xlabel('Release Year', fontsize=14) # x-axis title
plt.xticks(rotation=45, ha='right')
ax1.set_ylabel('Count', fontsize=14) # y-axis title

# Plot 2: Countplot for release years with hue for content type
ax2 = sns.countplot(
    data = netflix_data_cleaned,
    x = "release_year_range",
    hue = "type",
    palette = ["#cc0000","#073763"],
    order = bin_labels,
    ax = axes[1],
)

ax2.set_title('Release year of Movies vs TV Shows (Last Decade)', fontsize=16,
fontweight="bold")
ax2.set_xlabel('Release Year', fontsize=14) # x-axis title
plt.xticks(rotation=45, ha='right')
ax2.set_ylabel('Count', fontsize=14) # y-axis title

# Adjust layout
plt.tight_layout()
plt.show()
```

Now that the content of the last decade has been shown, we see that the first trend that was mentioned doesn't seem to be supported within this context. Looking at the graph on the left, we can say that because of the COVID-19 pandemic, years 2019 and 2020 have seen a big drop in content output, which does make sense. However, it seems as though this trend had already started 2 years before (from 2017 onward). This interesting decline in content in general can potentially be explained by the graph on the right, which shows the same data, but split between movies and TV shows.

In the right graph, looking at content being released between 2016-2017, there was barely an increase in movies, and almost all of the total increase comes from TV shows. In 2017, the number of movies nearly doubled that of the number of TV Shows, and in just 3 years, the number of TV shows is higher than Movies in 2020. Now if we look at the trend of just TV Shows, it is in-line with the first trend mentioned previously, that most recent content is more popular, ignoring the drop due to the pandemic. All of this supports the second point, that TV Shows are slowly becoming more popular than movies within the Netflix space.

Because the data is on Netflix, meaning not all possible content is available on the platform, we cannot conclude anything about the entertainment industry in general just yet. The next graph will look at content Netflix has released over the years to draw more conclusions towards the entertainment industry as a whole.

3.3 Year of Content Added on Netflix

In [18]:

```
plt.figure(figsize=(12, 8))
```

```
sns.set(style='whitegrid') # Set Seaborn style
colors = ["#cc0000"] # Define a custom color palette

# Create the histogram
sns.histplot(data=netflix_data, x='ages', discrete=True, color=colors[0], binwidth=0, kde=True)
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.title('Histogram of Age Groups Based on Ratings')

# Show the plot
plt.show()
```

This graph illustrates that on Netflix, movies and TV shows primarily target "Adults" as their audience, as indicated by the highest rating of the content. Following adults, the next largest target demographic is "Teens", then "Older Kids", and finally "Kids", with less than 2000 pieces of content aimed at this group.

In [19]:

```
plt.figure(figsize=(8, 6))

netflix_data = netflix_data.dropna(subset=['year_added']) # drop N/A values that show as a column (with no values) in visual
netflix_data['year_added'] = netflix_data['year_added'].astype(int) # year_added changed to int from float

# create bar chart
ax = sns.countplot(
    data = netflix_data,
    x = "year_added",
    hue = "type", # show type (Movie, TV Show)
    palette = {'Movie': '#df0707', 'TV Show': '#780909'},
    order = netflix_data["year_added"].sort_values().unique()  # Sort in ascending order (left to right)
)

plt.title('Content Added to Netflix Over the Years', fontsize=16, fontweight="bold")
plt.xlabel('Year added to Netflix', fontsize=14)
plt.xticks(rotation=45, ha='right')

plt.show()
```

This graph shows trends in how many titles Netflix adds each year. Trends:

- Netflix consistently adds more Movies than TV shows every year
- Total Content being added each year has fallen since 2019 (Movies had more of a decrease than TV shows)

Since the data seems consistent, in that trends in both Movies and TV shows have followed each other, we can conclude that the analysis of release year graphs do show trends within the entertainment industry as a whole.

In [ ]:

```
genres_count = netflix_data['listed_in'].str.split(', ').explode().value_counts() # Extract and count unique genres
top_genres = genres_count.head(15) # Select top 15 genres
netflix_palette = ['#221f1f', '#b20710', '#e50914', '#f5f5f1', '#b2b2b2']
```

```
#plotting the top 15 genres
plt.figure(figsize=(12, 8))
sns.barplot(y=top_genres.index, x=top_genres.values, palette=netflix_palette)
#label of the graph
plt.xlabel('Count')
plt.ylabel('Genres')
plt.title('Top 15 Genres on Netflix')
plt.show()
```

4. Prepare the data for Machine Learning Algorithms

In [21]:

```
columns_to_drop = ['show_id', 'title', 'director', 'cast', 'date_added']
# Drop the specified columns
netflix_data.drop(columns=columns_to_drop, inplace=True)
netflix_data.info() #check the info
```

In [22]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Drop unnecessary columns
selected_columns = ['listed_in', 'rating', 'release_year', 'month_added']
data_subset = netflix_data[selected_columns].copy()

# Convert categorical variables to numerical format using Label Encoding
label_encoder = LabelEncoder()
data_subset['rating'] = label_encoder.fit_transform(data_subset['rating'])
data_subset['listed_in'] = label_encoder.fit_transform(data_subset['listed_in'])

# Splitted the data into features (X) and target variable (y)
X = data_subset
y = netflix_data['type']

#Split the dataset into a training dataset (80%) and testing dataset.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
data_subset.head()
```

Release Year: This numerical feature might have some correlation with the type of content.

Rating: The content rating is a categorical feature, and it can provide valuable information about the type of content.

Listed_in: The genre of the content influence's whether the content is a movie or a TV show.

Month Added: These could capture patterns related to when content is added

When considering the feature "Duration," which represents the length of content in minutes, it appeared to be a self-sufficient and impactful predictor for distinguishing between movies and TV shows. To explore additional features that could contribute significantly to accuracy, we identified "Listed_in" and "rating" as another prominent features capable of making accurate predictions.

5. Select a model and train it

Algorithm 1 : Linear Regression

In [23]:

```
from sklearn.linear_model import LogisticRegression
model_lg = LogisticRegression()
model_lg.fit(X_train, y_train)
```

In [24]:

```
from sklearn.metrics import classification_report, accuracy_score
y_pred = model_lg.predict(X_test)
# Display classification report
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

In [25]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred))
disp.plot() # Display the Confusion Matrix
```

Logistic Regression is often used for tasks involving classification. In the case of predicting content types, such as whether its a Movie or a TV Show the task can be simplified to two classes. Logistic Regression is particularly suitable, for these scenarios as it models the probability of an instance belonging to a class and makes predictions based on a decision boundary. Moreover Logistic Regression is known for its simplicity, interpretability and computational efficiency making it an ideal option, for experimentation and creating baseline models.

Algorithm 2 : SVM Model

In [26]:

```
from sklearn.svm import SVC
model_svm = SVC(kernel='linear', C=0.1, gamma=1)
model_svm.fit(X_train, y_train)
```

In [27]:

```
from sklearn.metrics import classification_report, accuracy_score
y_pred = model_svm.predict(X_test)
# Display classification report
print(classification_report(y_test, y_pred, zero_division=1))
print(accuracy_score(y_test, y_pred))
```

In [28]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred))
disp.plot()  # Display the Confusion Matrix
```

The choice of using Support Vector Machines (SVM) for the task of predicting content types (Movie or TV Show) is influenced by SVM's capability to handle binary classification tasks effectively. SVM aims to find a hyperplane that best separates the data points of different classes, making it suitable for scenarios where the goal is to distinguish between two categories.

In this case, using the linear kernel (kernel='linear') and setting hyperparameters like C and gamma, we applied an SVM model. However, the achieved accuracy of 77% may suggest that the data's inherent complexity or the chosen features might not be perfectly separable using a linear SVM.

We tried to use (kernel='poly') but it took really long time and then it didnt excute which is why we choosed linear and for the value of C = 0.1 is the which is neither underfit or overfit.

Algorithm 3 : Random Forest Classifier

In [29]:

```
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)
```

In [30]:

```
from sklearn.metrics import classification_report, accuracy_score
rf_predictions = rf_classifier.predict(X_test)
# Display classification report
print(classification_report(y_test, rf_predictions))
print(accuracy_score(y_test, rf_predictions))
```

In [31]:

```
# Confusion Matrix Plot
cm = confusion_matrix(y_test, rf_predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=rf_classifier.classes_)
plt.figure(figsize=(8, 6))
disp.plot(cmap='Reds', values_format='.0f')
plt.title('Confusion Matrix')
plt.show()
```

Random Forest was employed for the task of predicting content types (Movie or TV Show) due to its ability to handle complex relationships within the data and improve generalization. By constructing an ensemble of decision trees and aggregating their predictions, Random Forest reduces the risk of overfitting compared to a single decision tree. This is crucial for achieving robust performance on unseen data. Additionally, Random Forest provides insights into feature importance, allowing us to understand which attributes contribute significantly to the classification task. The algorithm's capacity to handle categorical data, maintain interpretability, and produce reliable predictions makes it well-suited for the classification of content types in the context of this dataset.

Conclusion: Best-Performing Algorithm: Random Forest classifier

In [32]:

```
# Graph 1 and small description
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import LabelEncoder

# Convert categorical labels to binary format
label_encoder = LabelEncoder()
y_test_binary = label_encoder.fit_transform(y_test)

# Get predicted probabilities for the positive class
y_scores_rf = rf_classifier.predict_proba(X_test)[:, 1]

# Calculate ROC curve
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test_binary, y_scores_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot ROC curve
plt.figure(figsize=(10, 6))
plt.plot(fpr_rf, tpr_rf, color='#b20710', label=f'Random Forest ROC Curve (AUC =
{roc_auc_rf:.2f})')
```

```
plt.plot([0, 1], [0, 1], color='black', linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Random Forest Classifier')
plt.legend()
plt.show()
```

The ROC curve evaluates the performance of a Random Forest classifier. It plots the true positive rate against the false positive rate at varying thresholds. The red Random Forest curve significantly outperforms the random guess (black diagonal) line with an AUC of 99%. This high AUC demonstrates the Random Forest model's effectiveness at minimizing false positives and detecting true positives.

In [33]:

```
# Graph 2 and small description
from sklearn.model_selection import learning_curve

# Generate learning curve
train_sizes, train_scores, test_scores = learning_curve(rf_classifier, X, y, cv=10)

# Plot learning curve
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, np.mean(train_scores, axis=1), label='Training Score', color = "#073763")
plt.plot(train_sizes, np.mean(test_scores, axis=1), label='Test Score', color = "#b20710")
plt.xlabel('Training Set Size')
plt.ylabel('Accuracy')
plt.title('Random Forest Learning Curve')
plt.legend()
plt.show()
```

The learning curve helps to shows the performance of a model over experience or time. The blue line represents the training score. If it remains straight and close to 1, it means that the Random Forest classifier is able to perfectly (or near perfectly) predict the outcomes for the training data. This is expected behavior as machine learning models are typically able to perform well on the data they were trained on.

The red line represents the test score. If it starts low and then climbs up as the training size increases, it means that the model is learning from the additional data and is getting better at predicting outcomes for unseen data. The fact that it stops between 0.99 and 1 indicates that the model is performing very well on the test data, almost as well as it is on the training data.

This is a good sign and suggests that the model is not overfitting (which would be indicated by a high training score but low test score) or underfitting (which would be indicated by both scores being low). The model seems to generalize well from the training data to unseen data.

In [34]:

```
# Graph 3 and small description

features = X.columns
importances = rf_classifier.feature_importances_

# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]

# Plot the feature importances
plt.figure(figsize=(12, 6))
```

```
plt.bar(range(X.shape[1]), importances[indices], align='center', color='#b20710')
plt.xticks(range(X.shape[1]), features[indices], rotation=90)
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Feature Importance for Random Forest Classifier')
plt.show()
```

The graph is a bar chart of feature importances as determined by a Random Forest Classifier. The feature "listed_in" has the highest importance, close to 0.9, which means that it is the most influential feature in predicting the target variable according to the model.

The features "rating", "release year", and "month added" all have importance values below 0.2, indicating that they have less influence on the model's predictions compared to "listed_in".

In summary, the model is indeed highly dependent on the "listed_in" feature for its predictions. This could mean that the category in which a movie or show is listed plays a significant role in the outcome the model is predicting.

In conclusion, the **Random Forest classifier** emerged as the best-performing algorithm for predicting content types (Movie or TV Show) in the Netflix dataset. With an accuracy of approximately 99%, it demonstrated robust performance, effectively handling the complexity of the dataset and minimizing the risk of overfitting through its ensemble of decision trees. The precision, recall, and F1-score metrics further underscore the classifier's ability to make accurate predictions for both Movie and TV Show categories. The interpretability of Random Forest, coupled with its feature importance analysis, enhances our understanding of the key attributes influencing content classification. Overall, Random Forest proves to be a reliable and effective choice for this particular classification task.

Appendix Two:

Netflix Dataset: **https://www.kaggle.com/datasets/shivamb/netflix-shows/data**

Github Link: https://github.com/heetnarechania/EECS-3401-Netflix/blob/main/Final-project-Complete.ipynb

Video Link 1 (Youtube) https://youtu.be/p2t69_RM8p0:

Video Link 2 (ClipCharm): https://clipchamp.com/watch/oQ8NvOrCMfF