Purpose:
To create a sentiment analyzing machine learning model for movies and TV shows.

Technologies Used:
Data Collection:
  -Selenium
  -JavaScript
  -HTML/CSS
  -Web Scraping/Automated Testing
Machine Learning Model:
  -Natural Language Processing
  -Python
  -Pandas/Numpy
  -NLTK/PUNKT
  -scikit-learn

Training Data:
10,000+ reviews were web scraped from Rotten Tomatoes using Selenium. These reviews were taken from a variety of movies from the last 10 years. For more information on the data, please look at the web scraper file.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
import nltk
#nltk.download("punkt") --> Run this the first time
from nltk import word_tokenize
```

The NLTK library is the natural language tool kit library of python. It is used for working with human language data (parsing, text-processing, tokenization, etc.) and NLP. The PUNKT module is related to tokenization (process of breaking words into individual words/tokens).

In [3]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDispla
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import (
    TfidfVectorizer,
    CountVectorizer,
    ENGLISH_STOP_WORDS,
)
```

sklearn stands for scikit-learn. It is a machine learning library for Python. It can also do data pre-processing, model evaluation, etc.

Here is an explanation of some of the modules imported from the sklearn library:

train_test_split:

This function is used to split a dataset into training and testing sets. It randomly divides the data into two subsets, typically one for training the model and one for evaluating its performance.

TfidfVectorizer:

This class is used for converting a collection of raw text documents into a matrix of TF-IDF (Term Frequency-Inverse Document Frequency) features. It's commonly used for text data preprocessing in natural language processing tasks.

CountVectorizer:

This class is used for converting a collection of text documents into a matrix of token counts. It essentialy creates a bag-of-words representation of the text data.

ENGLISH_STOP_WORDS:

This is a predefined set of English stop words provided by scikit-learn. Stop words are common words (e.g., "the," "and," "is") that are often removed from text data during preprocessing because they may not provide meaningful information for certain NLP tasks.

The main purpose of these modules is for:

Importing metrics for model evaluation (accuracy, confusion matrix, and visualization).
Importing the LogisticRegression class for building logistic regression models.
Importing train_test_split for splitting datasets into training and testing sets.
Importing text preprocessing tools (TfidfVectorizer, CountVectorizer, and stop words) for working with text data in NLP tasks.

In [4]:
```
df = pd.read_csv("reviewsRated.csv")
df.head() #Please look at the data and inspect that it is formatted correctly prior
```

Out[4]:

	review	score
0	No words to describe how AWESOME Avengers Infi...	5.0
1	This movie strings together every previous MCU...	5.0
2	Marvel definitely works hard on this movie, yo...	3.5
3	Não tem nem o que dizer, é uma obra de arte, o...	5.0
4	This is one of the best in the MCU Saga. It's ...	4.0

Now we are going to process that data and create a specific training Data Frame
Please note, that if you wish to only include english reviews in your data set, you can do that here.
For this model, we will consider any reviews 4 stars and above to be **positive**. And any reviews at 2 or less stars to be **negative**. Any reviews in between will not be included to avoid ambiguity during the training process.

In [5]:
```
processedDf = pd.DataFrame()
tempCounter = 0
for i in range (len(df["score"])):
    if(df.at[i,'score']>=4):
        processedDf.at[tempCounter,'review'] = df.at[i,'review']
        processedDf.at[tempCounter,'newScore'] = 1
        tempCounter+=1
    elif(df.at[i,'score'] <=2.0):
        processedDf.at[tempCounter,'newScore'] = 0
        processedDf.at[tempCounter,'review'] = df.at[i,'review']
        tempCounter+=1
processedDf

#Once again, check to see if the new data frame matches your expectations
```

Out[5]:

	review	newScore
0	No words to describe how AWESOME Avengers Infi...	1.0
1	This movie strings together every previous MCU...	1.0
2	Não tem nem o que dizer, é uma obra de arte, o...	1.0
3	This is one of the best in the MCU Saga. It's ...	1.0
4	The amazing visuals to stunning music and the ...	1.0
...	...	...
6621	Amazing film this started a new series	1.0
6622	They keep getting worse and worse every time	0.0
6623	Give me your ugly face lol!!!!!!! He said that...	1.0
6624	Watch it watch it watch it tutututruuut	1.0
6625	The best one out of all of them it's amazing	1.0

6626 rows × 2 columns

Now it is time to make the vectorizer so we can get numerical values to create predictions

In [6]:
```
word_tokens = [word_tokenize(review) for review in processedDf["review"]]
```

In [7]:
```
processedDf["n_words"] = [len(word_tokens[i]) for i in range(len(word_tokens))]
processedDf
```

Out[7]:

	review	newScore	n_words
0	No words to describe how AWESOME Avengers Infi...	1.0	21
1	This movie strings together every previous MCU...	1.0	30
2	Não tem nem o que dizer, é uma obra de arte, o...	1.0	18
3	This is one of the best in the MCU Saga. It's ...	1.0	36
4	The amazing visuals to stunning music and the ...	1.0	23
...	...	...	...
6621	Amazing film this started a new series	1.0	7
6622	They keep getting worse and worse every time	0.0	8
6623	Give me your ugly face lol!!!!!!! He said that...	1.0	18
6624	Watch it watch it watch it tutututruuut	1.0	7
6625	The best one out of all of them it's amazing	1.0	11

6626 rows × 3 columns

In [8]:
```
vect = TfidfVectorizer(
    stop_words = list(ENGLISH_STOP_WORDS),
    ngram_range=(1,2), #Will allow for individual or 2 consecutive words to be cons
    max_features=300, #Will pick 300 most frequent words, can be reduced if compute
    token_pattern=r"\b[^\d\W][^\d\W]+\b", #This makes sure that the words match a p
)
```

In [9]:
```
vect.fit(processedDf.review) #The vector learns from the provided data by tokenizin
                             # And then calculating "weightage" for the words based
                             #Now this is ready to transform other data
```

Out[9]:
```
TfidfVectorizer
TfidfVectorizer(max_features=300, ngram_range=(1, 2),
                stop_words=['back', 'down', 'over', 'as', 'without', 'f
ire',
                            'during', 'while', 'nor', 'whether', 'herea
fter',
                            'further', 'two', 'sincere', 'whom', 'someh
ow',
                            'we', 'would', 'keep', 'many', 'but', 'ha
d', 'my',
```

In [10]:
```
X = vect.transform(processedDf.review) #We sued the vector "fitted" above to transf
```

In [11]:
```
df_transformed = pd.DataFrame(data=X.toarray(), columns=vect.get_feature_names_out(
```

In [12]:
```
tempDf = processedDf.drop(["review","n_words"], axis=1) #Create a new data frame th
transformed = pd.concat([tempDf,df_transformed], axis=1) #This will combine the pos
                                                          #And the transformed dataf
transformed
```

Out[12]:

	newScore	absolute	absolutely	acting	action	actor	actors	actually	amazing
0	1.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000
1	1.0	0.0	0.0	0.0	0.260554	0.0	0.0	0.0	0.000000
2	1.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000
3	1.0	0.0	0.0	0.0	0.269246	0.0	0.0	0.0	0.000000
4	1.0	0.0	0.0	0.0	0.352584	0.0	0.0	0.0	0.400936
...	...	...	...	...	...	...	...	...	...
6621	1.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.446857
6622	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000
6623	1.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000
6624	1.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000
6625	1.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.767156

6626 rows × 301 columns

In [13]:
```
y = transformed["newScore"]
x = transformed.drop("newScore",axis=1)

X_train,X_test, y_train,y_test, = train_test_split(
    x,
    y,
    test_size=0.25, #75% of data is used for training and 25% is used for testing i
    random_state = 426, #A random seed that has been set, so others can reproduce t
)
```

In [14]:
```
    log_reg = LogisticRegression().fit(X_train,y_train)
```

In [15]:
```
y_predicted = log_reg.predict(X_test)
```

In [16]:
```
print("Accuracy: ", accuracy_score(y_test,y_predicted))
print(confusion_matrix(y_test,y_predicted)/len(y_test))
```

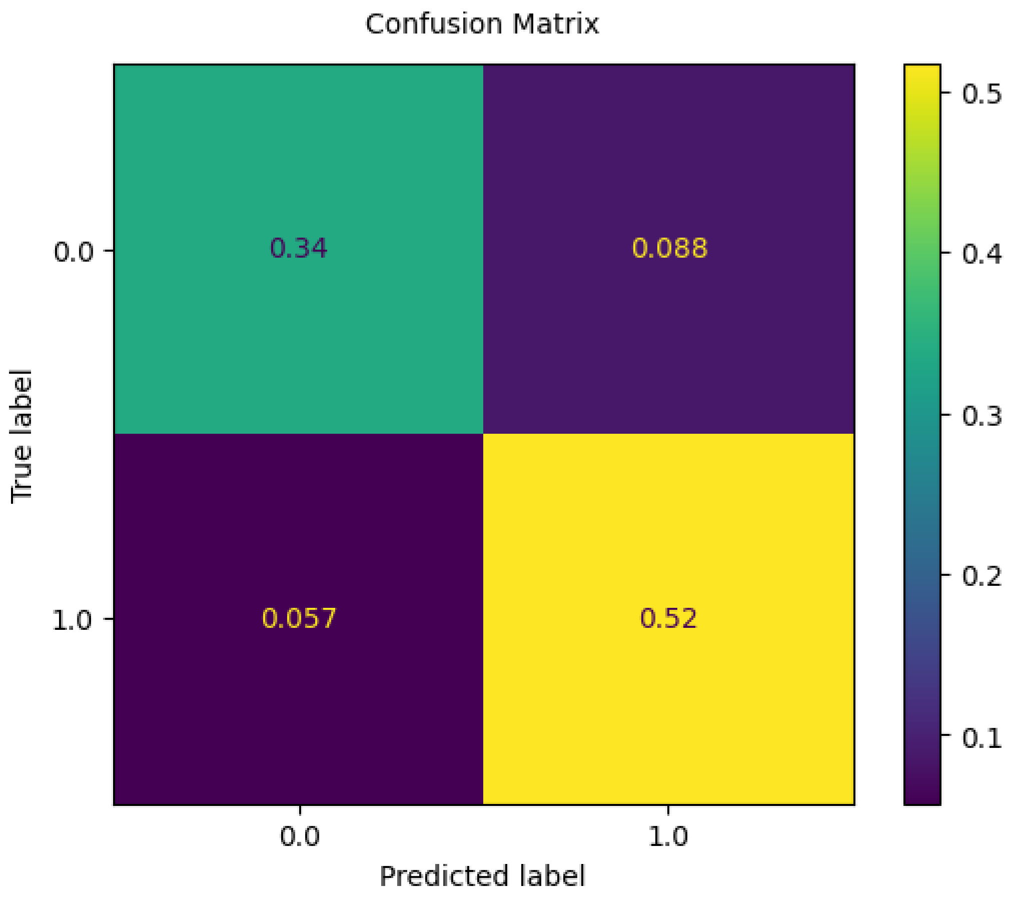
```
Accuracy:  0.8551599275799638
[[0.33796017 0.08750754]
 [0.05733253 0.51719976]]
```

We can display these results more visually.

Here is how to read the confusion matrix below:

- The top left quadrant is true negatives
- The top right quadrant is false positives (review is negative, but model judged it as positive)
- The bottom left quadrant is false negatives (review is positive, but model judged it as negative)
- The bottom right quadrant is true positives

In [17]:
```
ConfusionMatrixDisplay.from_estimator(log_reg,X_test,y_test,normalize="all")
plt.title("Confusion Matrix", y=1.02,size=10)
plt.show()
```



Here we can see that there is 34% true negatives and 52% true positives.
Given the testing and training data, this model has an 86% accuracy rate at determining if a review is either positive or negative.
With some adjustments, this model can be used to determine the "live" public sentiment about a movie (by feeding the model data through automated scrapers on social media).
It can also be modified to determine sentiment on other trending topics (such as product releases, controversial news, etc.)