IMDB Movies Review

May 04, 2023

Introduction:

The text mining application I am investigating is focused on analyzing movie reviews from the IMDB website. I will be scraping data from IMDB to gather a large corpus of movie reviews, and then perform various text mining techniques to gain insights and extract valuable information from the text data.

- Large and Diverse Dataset: IMDB contains a vast amount of movie reviews written by users from all over the world. This dataset provides a rich source of textual data with diverse opinions, sentiments, and expressions, making it ideal for exploring various text mining techniques.
- Sentiment Analysis: Analyzing sentiments in movie reviews is a common and important task in text mining. By performing sentiment analysis on the basis of movie ratings, we can identify the overall sentiment (positive, negative, or neutral) associated with different movies. This analysis can provide valuable insights into audience preferences and help in understanding the factors that contribute to the success or failure of a movie.
- Feature Extraction: Text mining techniques allow us to extract meaningful features from the movie reviews, such as key topics, themes, or aspects that are frequently mentioned by reviewers. This information can be useful for understanding the aspects of a movie that receive the most attention and influence the audience's perception.
- Predictive Analysis: By combining text mining with machine learning algorithms, we can build predictive models to classify movie reviews into different categories (e.g., positive or negative). This can be beneficial for automatically categorizing reviews and identifying patterns or trends in the data.

Overall, this project offers an opportunity to apply various text mining techniques to a large and diverse dataset of movie reviews, allowing us to gain insights, perform sentiment analysis, extract meaningful features, and potentially build predictive models.

Proposal:

- Objective:

  The objective of this project is to perform text mining analysis on movie reviews from the IMDB website. By scraping data, conducting sentiment analysis, and extracting meaningful features, we aim to gain insights into audience opinions, identify sentiment patterns based on movie ratings, and understand the factors influencing movie success.

- Methedology:
  1. Data Scraping: Develop a web scraping program to collect movie reviews from IMDB, extracting movie titles, ratings, and review text.
  2. Sentiment Analysis: Apply sentiment analysis techniques to classify reviews into positive, negative, or neutral categories based on associated ratings.
  3. Feature Extraction: Implement methods to extract key features and topics from the reviews, such as plot, acting, cinematography, etc.
  4. Predictive Analysis: Build a predictive model to classify movie reviews based on sentiment using machine learning algorithms.
- Deliverables:
  1. Data scraping program to collect movie reviews.
  2. Sentiment analysis results and sentiment trends based on movie ratings.
  3. Extracted features and topics from the reviews.
  4. Predictive model for sentiment classification.

What we are testing:

The purpose of the experiments in this project is to analyze IMDB movie reviews using text mining techniques. We aim to test the effectiveness of data scraping in collecting a large corpus of movie reviews from the IMDB website, including relevant information such as movie titles, ratings, and review text. Additionally, we will evaluate the performance of sentiment analysis techniques in accurately classifying movie reviews into positive, negative, or neutral sentiment categories based on associated ratings. The overall goal is to gain insights into audience

opinions, sentiment trends, and the factors influencing movie success through comprehensive text mining analysis of the IMDB movie reviews dataset.

Dataset Explanation:

1. **Movie**: This variable represents the title or name of the movie. It serves as an identifier for each film in your dataset.
2. Year: This variable indicates the year in which the movie was released. It provides information about the temporal aspect of the films.
3. **Time_minute**: This variable denotes the duration of the movie in minutes. It specifies the length of the film, which can vary from movie to movie.
4. **IMDB Rating**: This variable represents the rating given to the movie on the IMDB website. IMDB ratings are typically based on user reviews and range from 1 to 10, with higher ratings indicating better reception.
5. **Metascore**: Metascore is a weighted average of critic reviews for a movie, calculated by the review aggregation website Metacritic. It provides a numerical score (typically from 0 to 100) that reflects the critical reception of the film.
6. **Vote**: This variable represents the number of votes or ratings that the movie has received on the IMDB website. It indicates the level of user engagement or popularity of the film.
7. **Gross Earnings**: Gross earning refers to the total revenue generated by the movie at the box office. It represents the financial success of the film and is usually measured in monetary units (e.g., US dollars).
8. **Description**: This variable contains a brief summary or description of the movie. It provides a textual overview of the film's plot, genre, and other relevant details.

Methodology:

1. Web Scraping:

   For this project, a dataset of 1000 top movies and 1000 movies from the bottom has been scraped, capturing various variables related to each film. The variables include movie title, release year, duration in minutes, IMDB rating, Metascore, number of votes, gross earnings, and a brief description.I have used **Beautiful soap** for the Web Scraping.

   The provided code is a Python script that scrapes data from IMDb for both the top 1000 movies and the bottom 1000 movies. It collects information such as movie titles, release years, duration, IMDb ratings, Metascores, number of votes, gross earnings, and movie descriptions.

   Here's a brief summary of the code:

   - The code defines two URLs for the top and bottom movies respectively.
   - It initializes empty lists to store the scraped data for each variable.
   - It iterates through the URLs and retrieves the contents of each URL using the requests library.
   - BeautifulSoup is used to parse the HTML and extract relevant information from the movie_div section of the HTML.
   - The script then iterates through each movie container, extracting the movie's title, year, duration, IMDb rating, Metascore, number of votes, gross earnings, and movie description.
   - The scraped data is appended to their respective lists.
   - A random sleep interval is added between requests to mimic human-like browsing behavior.
   - The code repeats this process for each page of movies.

```
# URLs for top and bottom movies
urls = ['https://www.imdb.com/search/title/?groups=top_1000&start=', 'https://www.imdb.com/search/title/?
titles = []
years = []
time = []
imdb_ratings = []
metascores = []
votes = []
us_gross = []
descriptions = []

for url in urls:
    for page in pages:
        # Getting the contents from each URL
        page = requests.get(url + str(page), headers=headers)
        soup = BeautifulSoup(page.text, 'html.parser')

        # Aiming the part of the HTML we want to get the information from
        movie_div = soup.find_all('div', class_='lister-item mode-advanced')

        # Controlling the loop's rate by pausing the execution for a specified amount of time
        # Waiting time between requests for a number between 2-10 seconds
        sleep(randint(2, 10))
```

```
        for container in movie_div:
            # Scraping the movie's name
            name = container.h3.a.text
            titles.append(name)

            # Scraping the movie's year
            year = container.h3.find('span', class_='lister-item-year').text
            years.append(year)

            # Scraping the movie's length
            runtime = container.find('span', class_='runtime').text if container.find('span', class_='rur
            time.append(runtime)

            # Scraping the rating
            imdb = float(container.strong.text)
            imdb_ratings.append(imdb)

            # Scraping the metascore
            m_score = container.find('span', class_='metascore').text.strip() if container.find('span', c
            metascores.append(m_score)

            # Scraping votes and gross earnings
            nv = container.find_all('span', attrs={'name': 'nv'})
            vote = nv[0]['data-value']
            votes.append(vote)
            grosses = nv[1]['data-value'] if len(nv) > 1 else '-'
            us_gross.append(grosses)
```

Exploratory Data Analysis:

Movies discription:

```
movies['discription'].head()
```

```
0    \nStill reeling from the loss of Gamora, Peter...
1    \nA group of intergalactic criminals must pull...
2    \nJohn Wick uncovers a path to defeating The H...
3    \nThe Guardians struggle to keep together as a...
4    \nA noble family becomes embroiled in a war fo...
Name: discription, dtype: object
```

Cleaning Dataset:

To extract the year,time_minute columns digits

```
# Cleaning 'year' column

movies['year'] = movies['year'].str.extract('(\d+)').astype(int)

movies.head(3)
```

```
# Cleaning 'time_minute' column
movies['time_minute'] = movies['time_minute'].str.extract('(\d+)').astype(int)
movies.head(3)
```

	movie	year	time_minute	imdb_rating	metascore	vote	gross_earning	
0	Guardians of the Galaxy Vol. 3	2023	150	8.3	64	109240	140	\nSt
1	Guardians of the Galaxy	2014	121	8.0	76	1210811	333,176,600	
2	John Wick: Chapter 4	2023	169	8.2	78	147604	200	\nJo

We have seen that some grossing values are not good extracted that's why we are replacing then with the Nan.

```
[75] # Replace 3-digit values with NaN
     df.loc[df['gross_earning'].apply(lambda x:  len(str(x)) < 8), 'gross_earning'] = np.nan
```

Dataset Insights:

```
movies.describe()
```

	year	time_minute	imdb_rating	metascore	vote	gross_earning
count	2000.000000	2000.000000	2000.000000	1663.000000	2.000000e+03	1.456000e+03
mean	1999.790000	111.927500	6.322100	56.742032	1.785591e+05	5.534253e+07
std	20.192468	26.134436	1.750435	25.528733	3.043137e+05	9.288657e+07
min	1920.000000	45.000000	1.200000	1.000000	1.033200e+04	1.001190e+05
25%	1993.000000	93.000000	4.900000	34.000000	2.606900e+04	6.707589e+06
50%	2006.000000	104.000000	7.200000	57.000000	5.510800e+04	2.344062e+07
75%	2014.000000	125.000000	7.900000	80.000000	1.714642e+05	5.864940e+07
max	2023.000000	321.000000	9.300000	100.000000	2.739736e+06	9.366622e+08

Top gross earning movies

```
# Sort the DataFrame by 'gross_earning' in descending order
sorted_df = movies.sort_values(by='gross_earning', ascending=False)

# Print the movies with top grossing
top_movies = sorted_df.head()
print(top_movies)
```

```
                                            movie  year  time_minute  \
106  Star Wars: Episode VII - The Force Awakens  2015          138
15                             Avengers: Endgame  2019          181
16                         Spider-Man: No Way Home  2021          148
59                                          Avatar  2009          162
9                                 Top Gun: Maverick  2022          130

     imdb_rating  metascore      vote  gross_earning  \
106          7.8       80.0    946168     936662225.0
15           8.4       78.0   1175788     858373000.0
16           8.2       71.0    793066     804747988.0
59           7.9       83.0   1340536     760507625.0
9            8.3       78.0    583486     718732821.0
```

Top Rated movies

```
[110] # Sort the DataFrame by 'Top Rating' in ascending order
      sorted_df = movies.sort_values(by='imdb_rating',ascending=False)

      # Print the movies with bottom grossing
      bottom_movies = sorted_df.head()
      print(bottom_movies)
```

```
                        movie  year  time_minute  imdb_rating  metascore  \
7   The Shawshank Redemption  1994          142          9.3       82.0
10              The Godfather  1972          175          9.2      100.0
90               12 Angry Men  1957           96          9.0       97.0
53           Schindler's List  1993          195          9.0       95.0
17            The Dark Knight  2008          152          9.0       84.0
```

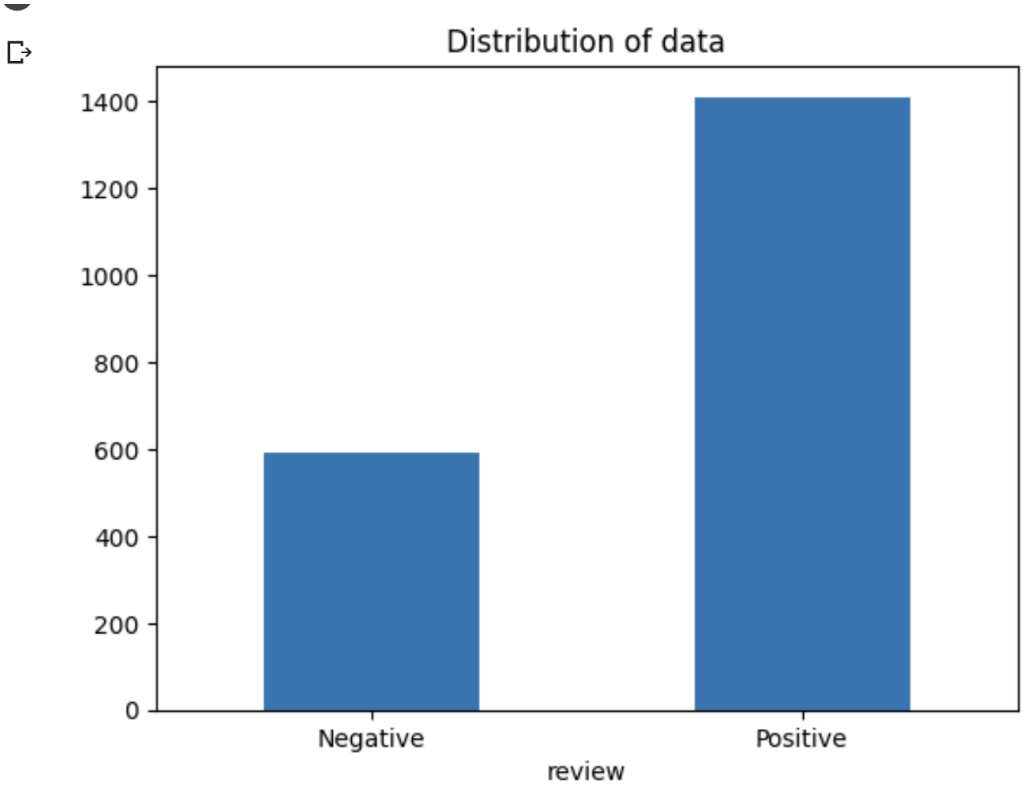
Sentiment Analysis:

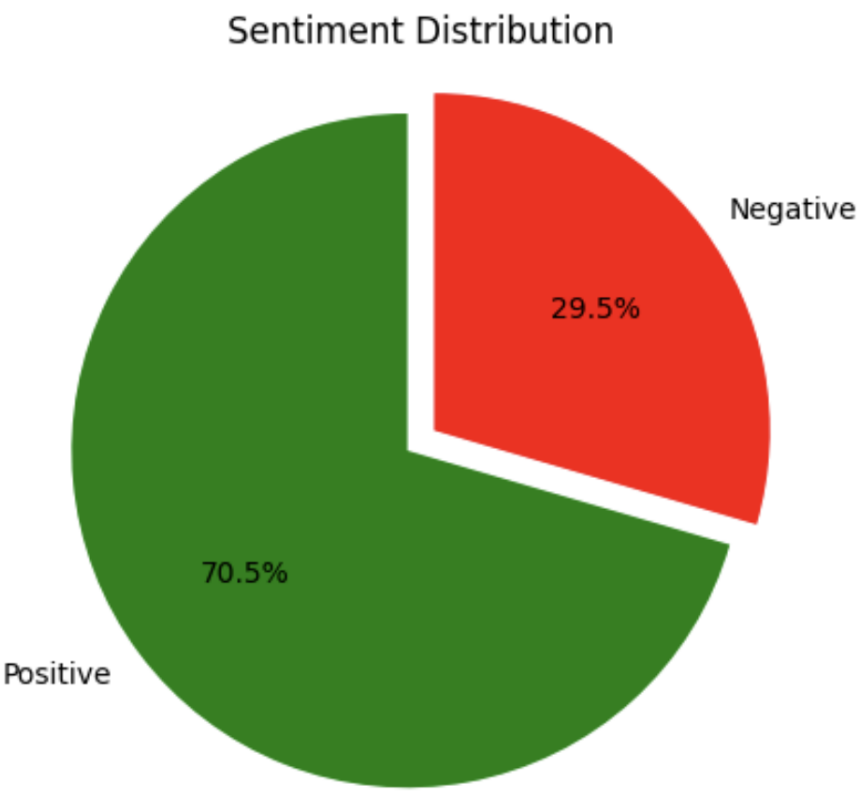
adding a 'review' column to the 'movies' dataset based on the 'imdb_rating' column. It categorizes the reviews as either 'positive' or 'negative' depending on whether the IMDb rating is greater than 5 or not.

```
17] # Create the 'review' column based on the 'Rating' column
    movies['review'] = movies['imdb_rating'].apply(lambda x: 'positive' if x > 5 else 'negative')
    movies.head()
```

	movie	year	time_minute	imdb_rating	metascore	vote	gross_earning	discription	review
0	Guardians of the Galaxy Vol. 3	2023	150	8.3	64.0	109240	140	\nStill reeling from the loss of Gamora, Peter...	positive
1	Guardians of the Galaxy	2014	121	8.0	76.0	1210811	333,176,600	\nA group of intergalactic criminals	positive

Sentiments Distribution:





Removing Stop words:

Stop words are commonly used words in a language that are considered to have little or no significance in determining the meaning of a text. Examples of stop words in English include "a," "an," "the," "in," "on," "is," etc.

```
# Function to remove stopwords from a text
def remove_stopwords(text):
    tokens = word_tokenize(text)  # Tokenize the text
    filtered_tokens = [token for token in tokens if token.lower() not in stop_words]  # Remove stopwords
    filtered_text = ' '.join(filtered_tokens)  # Reconstruct the text
    return filtered_text

# Apply the remove_stopwords function to the 'description' column
movies['discription'] = movies['discription'].apply(remove_stopwords)
```

Removing Punctuations:

Removing punctuation marks from text is a common preprocessing step in natural language processing tasks. Punctuation marks include characters such as periods, commas, question marks, exclamation marks, and quotation marks. Removing punctuation can help simplify text and reduce noise that may not contribute to the meaning of the text.

```
import string
# Function to remove punctuation from a text
def remove_punctuation(text):
    translator = str.maketrans('', '', string.punctuation)  # Create a translation table for punctuation
    text_without_punct = text.translate(translator)  # Remove punctuation using the translation table
    return text_without_punct
```

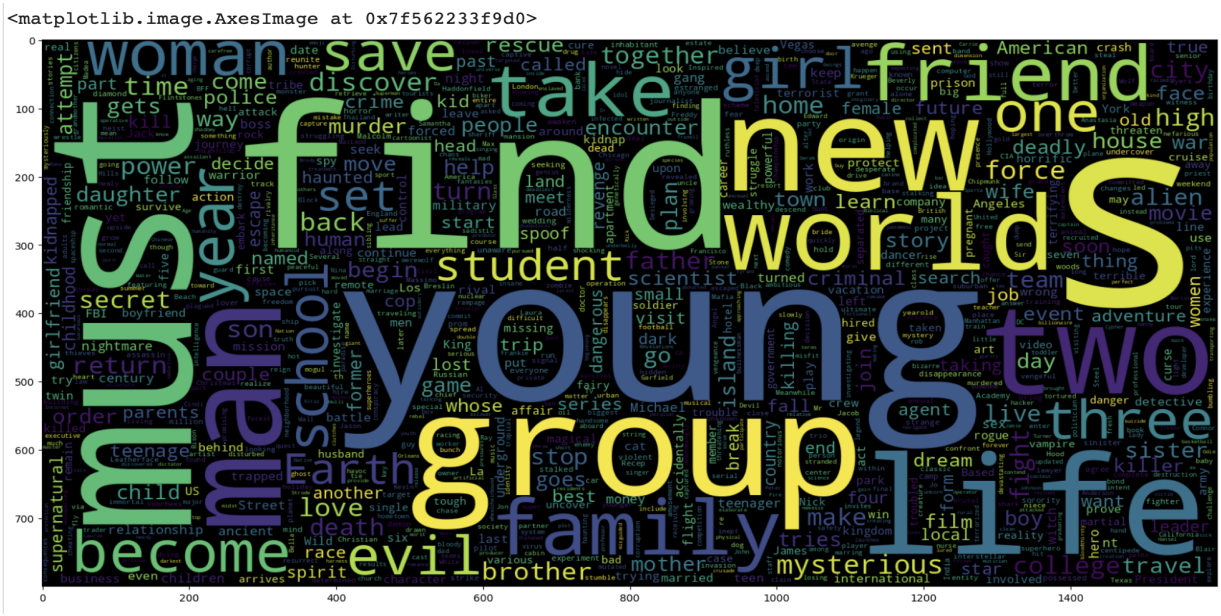
Lemmatization:

Lemmatization is the process of reducing words to their base or root form, known as a lemma. It involves identifying the canonical form of a word, considering its intended meaning and context. For example, let's consider the word "running." The lemma for this word, obtained through lemmatization, would be "run." Similarly, for words like "better," "best," and "good," the lemmatization process would map them to their lemma "good."

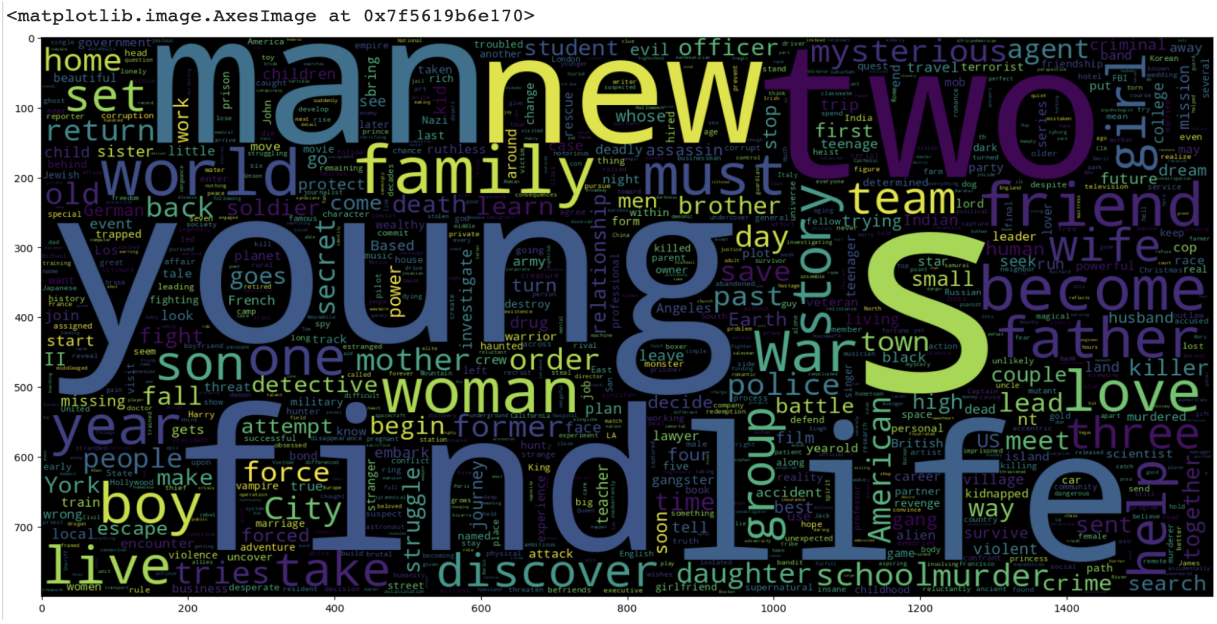
```
lm = nltk.WordNetLemmatizer()
nltk.download('wordnet')
def lemmatizer_on_text(data):
    text = [lm.lemmatize(word) for word in data]
    return data
movies['discription'] = movies['discription'].apply(lambda x: lemmatizer_on_text(x))
movies['discription'].head()
```

Word Clouds:

- **Most used words in negative feedback movies:**



- **Most used words in positive feedback movies:**



Tfid-Vectorizer:

Converting a collection of raw documents into a matrix of TF-IDF (Term Frequency-Inverse Document Frequency) features.

Train test Split:

```
# Separating the 80% data for training data and 20% for testing data
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, random_state =26105111)
```

Machine Learning Algorithms:

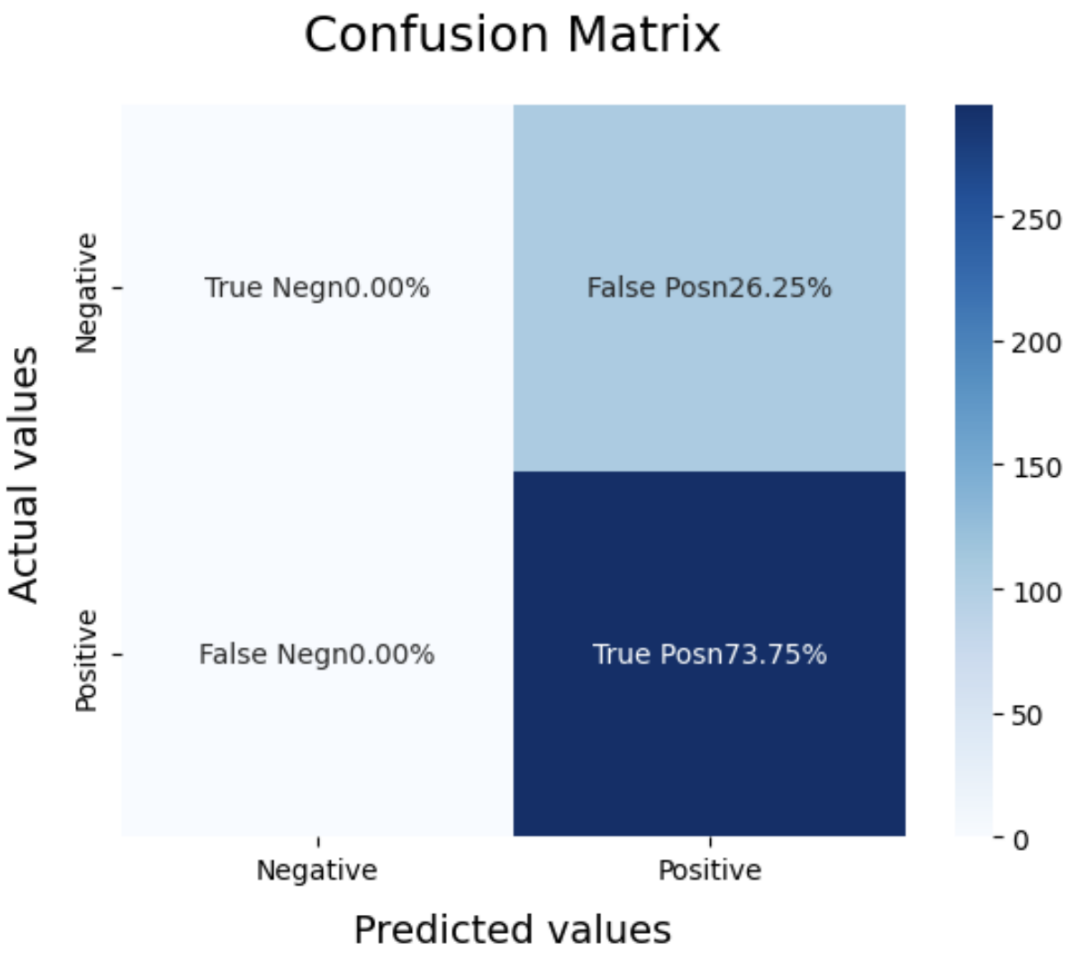
- **Naive Bayes**

  Naive Bayes is a classification algorithm based on Bayes' theorem, which is a probabilistic approach for making predictions. The "naive" assumption in Naive Bayes refers to the assumption of independence between features, meaning that the presence or absence of a particular feature does not depend on the presence or absence of any other feature.

```
BNBmodel = BernoulliNB()
BNBmodel.fit(X_train, y_train)
model_Evaluate(BNBmodel)
y_pred1 = BNBmodel.predict(X_test)
```

Evaluation:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	105
1	0.74	1.00	0.85	295
accuracy			0.74	400
macro avg	0.37	0.50	0.42	400
weighted avg	0.54	0.74	0.63	400



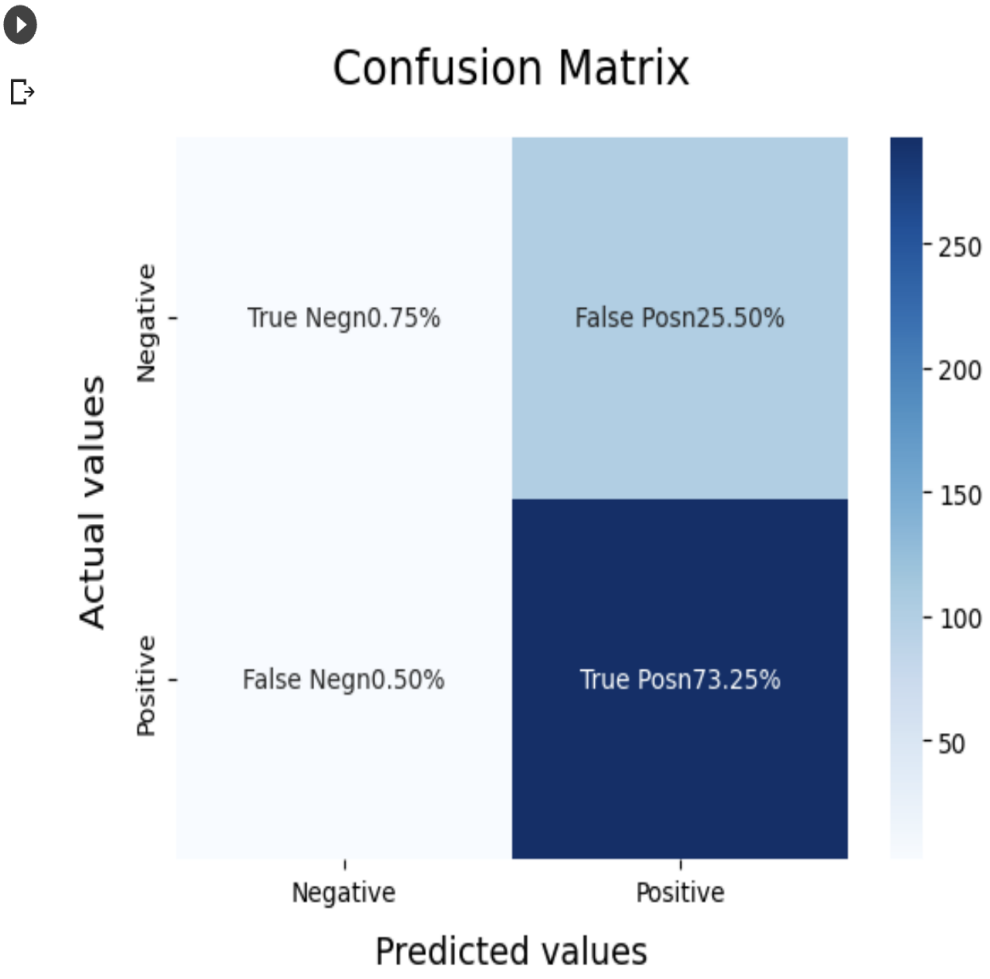
- **Logistic Regression:**

Logistic Regression is a statistical modeling technique used for binary classification tasks. It predicts the probability of an instance belonging to a particular class. Despite its name, logistic regression is a classification algorithm rather than a regression algorithm. In logistic regression, the dependent variable is binary or categorical, and the independent variables can be continuous or categorical. The goal is to find a relationship between the independent variables and the probability of the binary outcome.

```
[37] LRmodel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)
     LRmodel.fit(X_train, y_train)
     model_Evaluate(LRmodel)
     y_pred3 = LRmodel.predict(X_test)
```

Evaluation:

	precision	recall	f1-score	support
0	0.60	0.03	0.05	105
1	0.74	0.99	0.85	295
accuracy			0.74	400
macro avg	0.67	0.51	0.45	400
weighted avg	0.70	0.74	0.64	400


Confusion Matrix
Actual values
Negative
Positive
True Negn0.75%
False Posn25.50%
False Negn0.50%
True Posn73.25%
250
200
150
100
50
Negative
Positive
Predicted values