Sentiment Analysis

Supervised by :

Dr.Eman Raslan

IBM Data Science

CLS ONL1_AIS3_S1e

Our Team Members

Ahmed Balta	Abanoub Reda	Ahmed Adel	Hussein Mohamed
Eman Osama	Rahma Osama	Mohamedden Othman	Mariam Mahmoud

Agenda

1-Project Overview

2-Data Exploration and Visualization

3-Data Cleaning and preprocessing

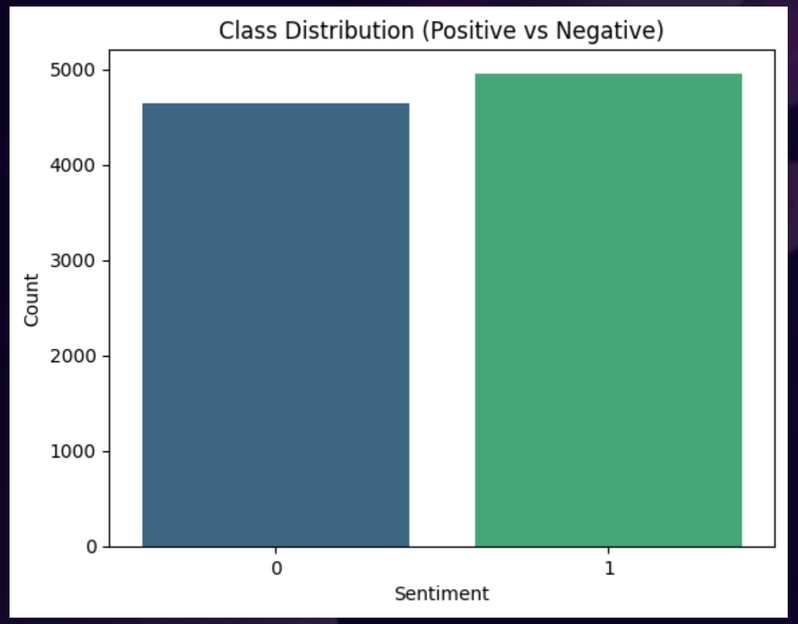
4-Build and Evaluate model

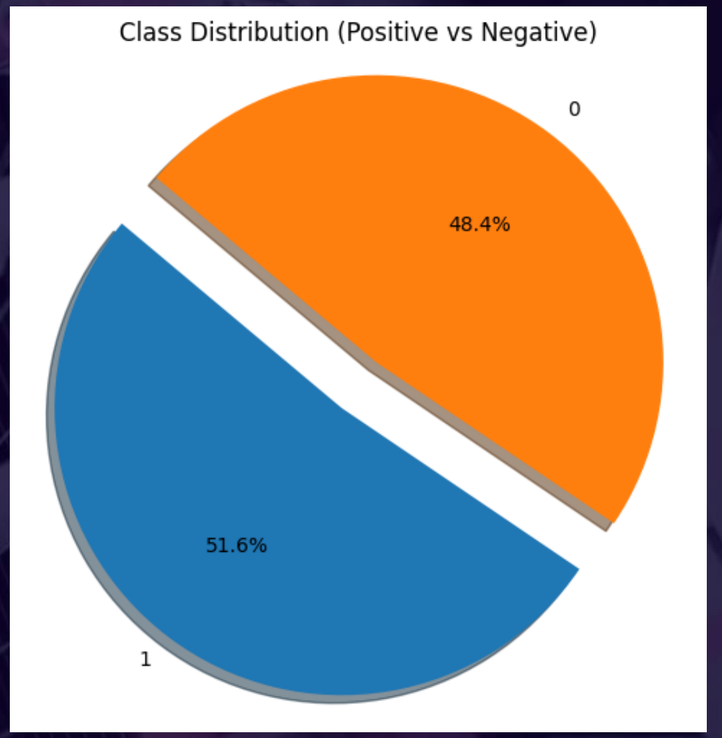
Project Overview

The primary goal of a sentiment analysis project is to determine the sentiment (positive or negative) expressed in text data such as social media posts, reviews, feedback, or other types of unstructured text. The project can be used for business insights, customer feedback analysis, product reviews, etc.

Data Exploration and Visualization

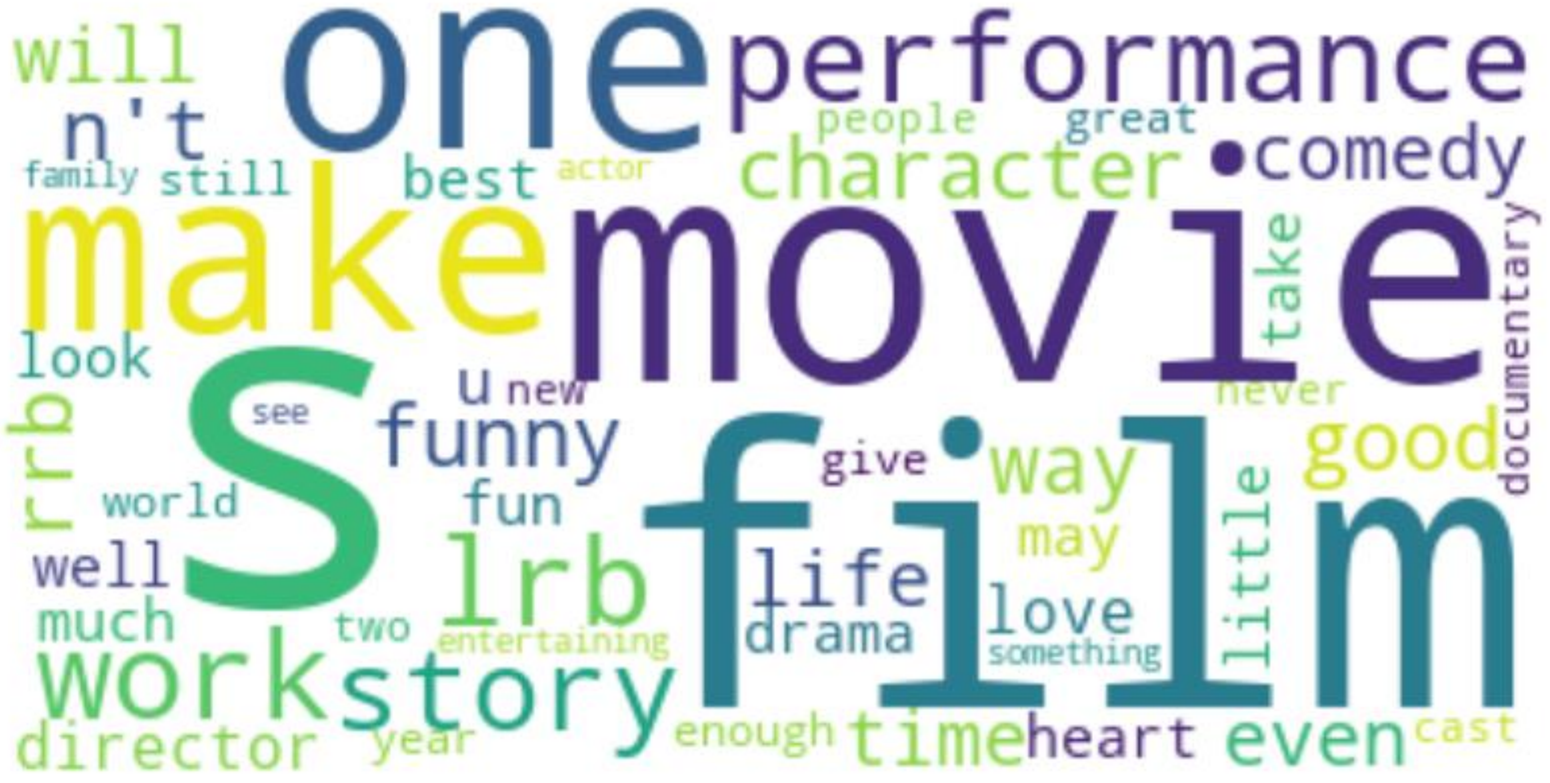
breaks down complex data and extracts meaningful insights



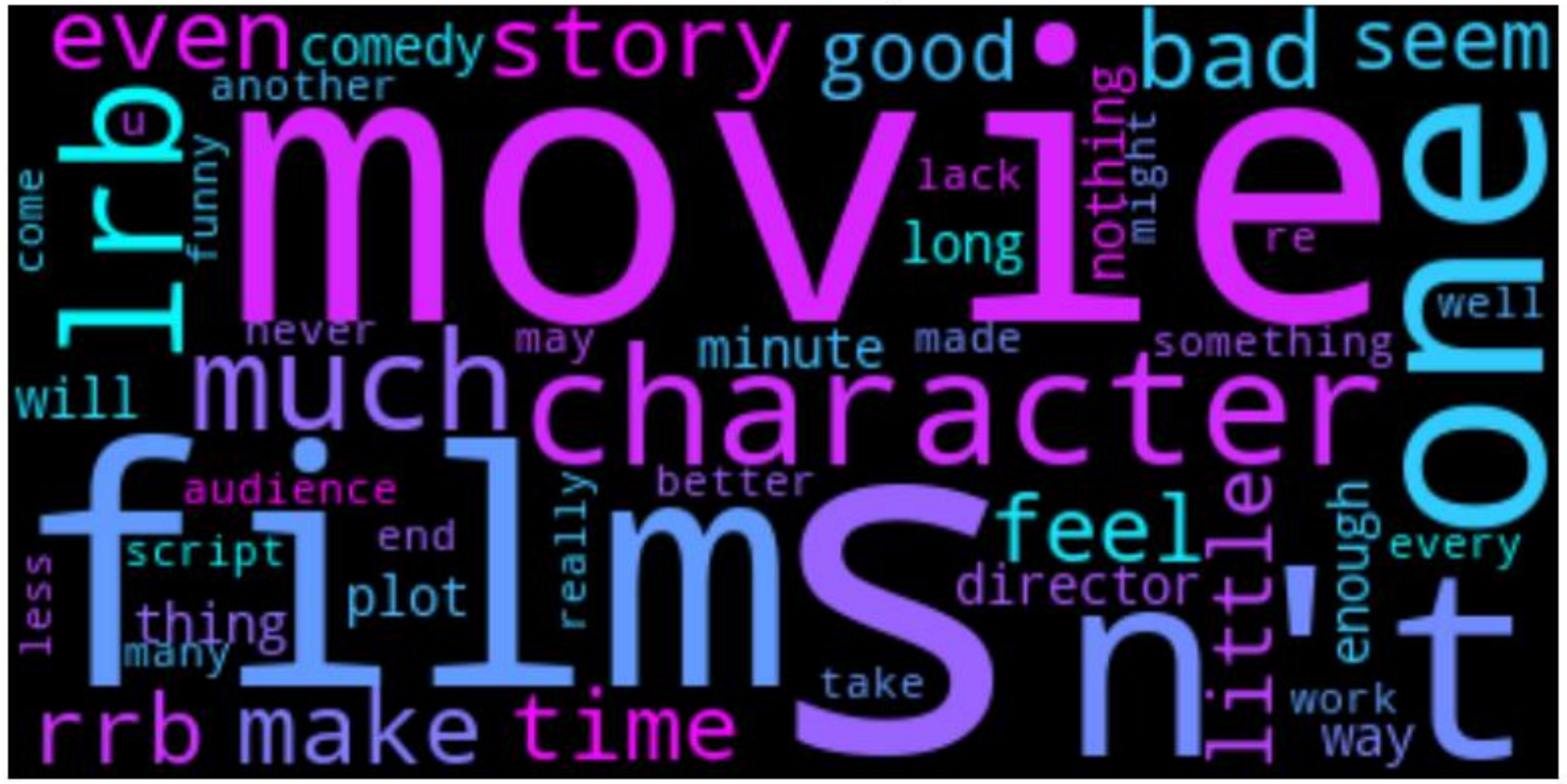


a nearly balanced class distribution between the two sentiment categories, "Positive" (1) and "Negative" (0)

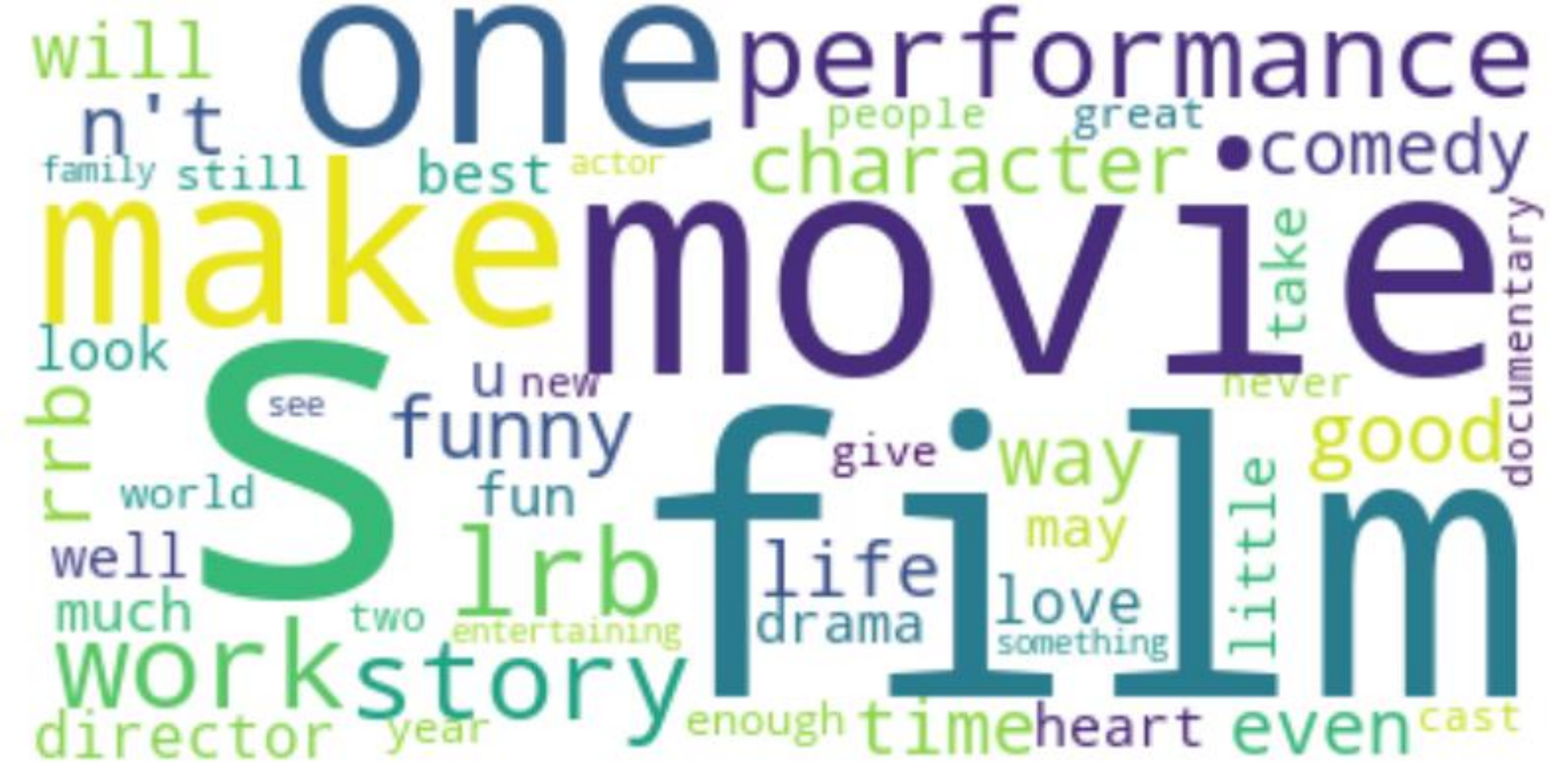
Most Common Words in Positive Reviews



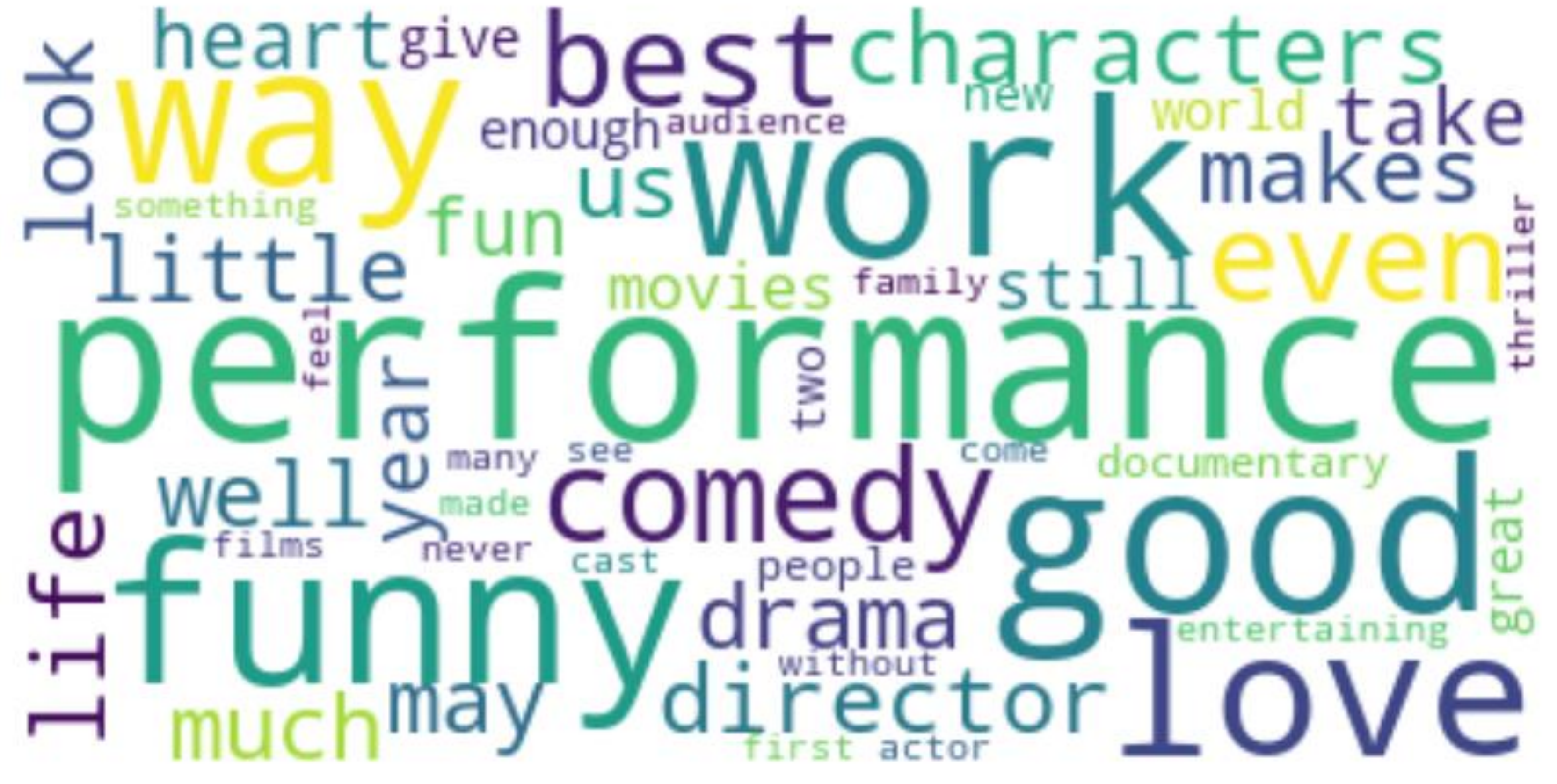
Most Common Words in Negative Reviews



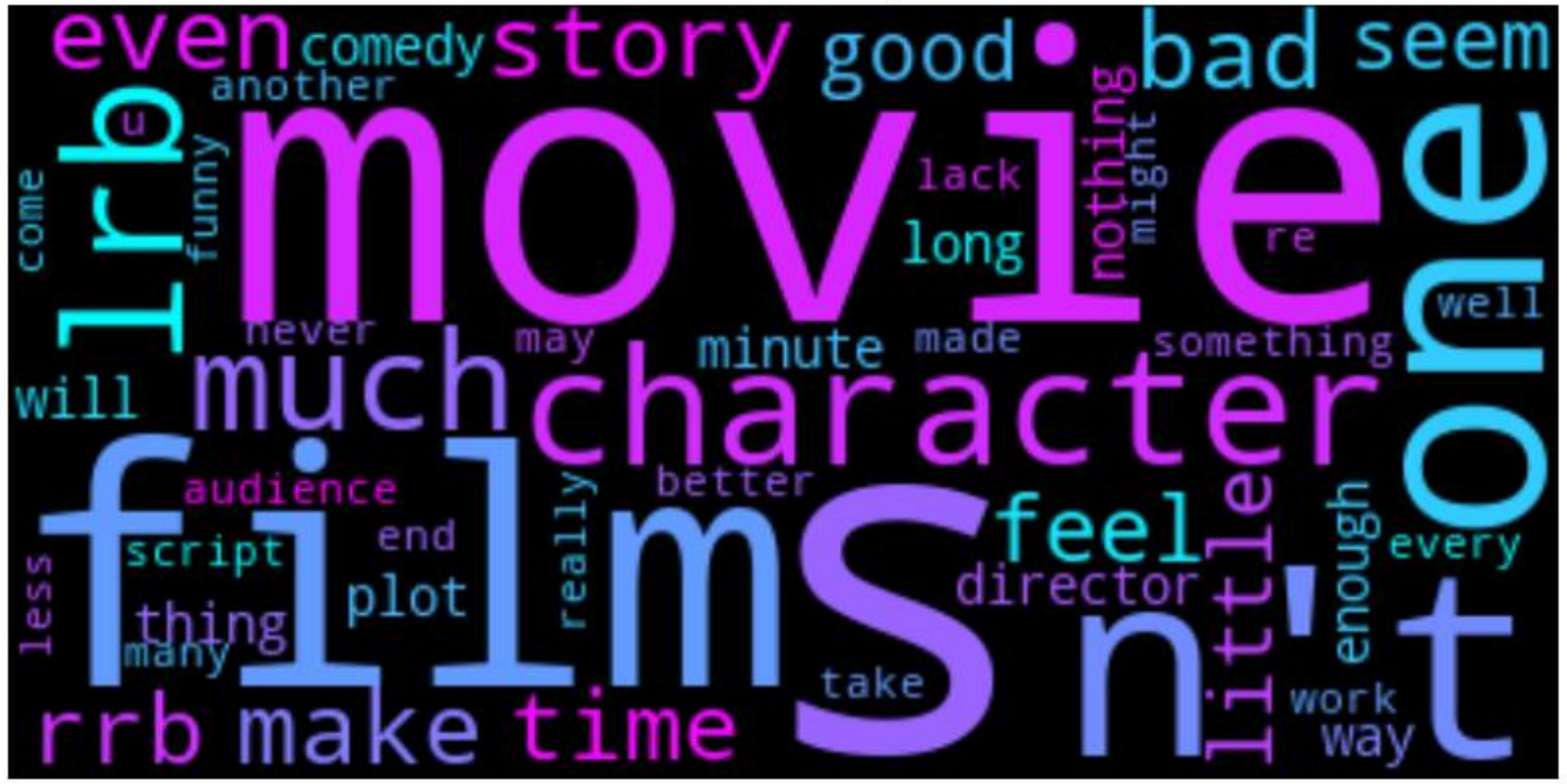
Most Common Words in Positive Reviews



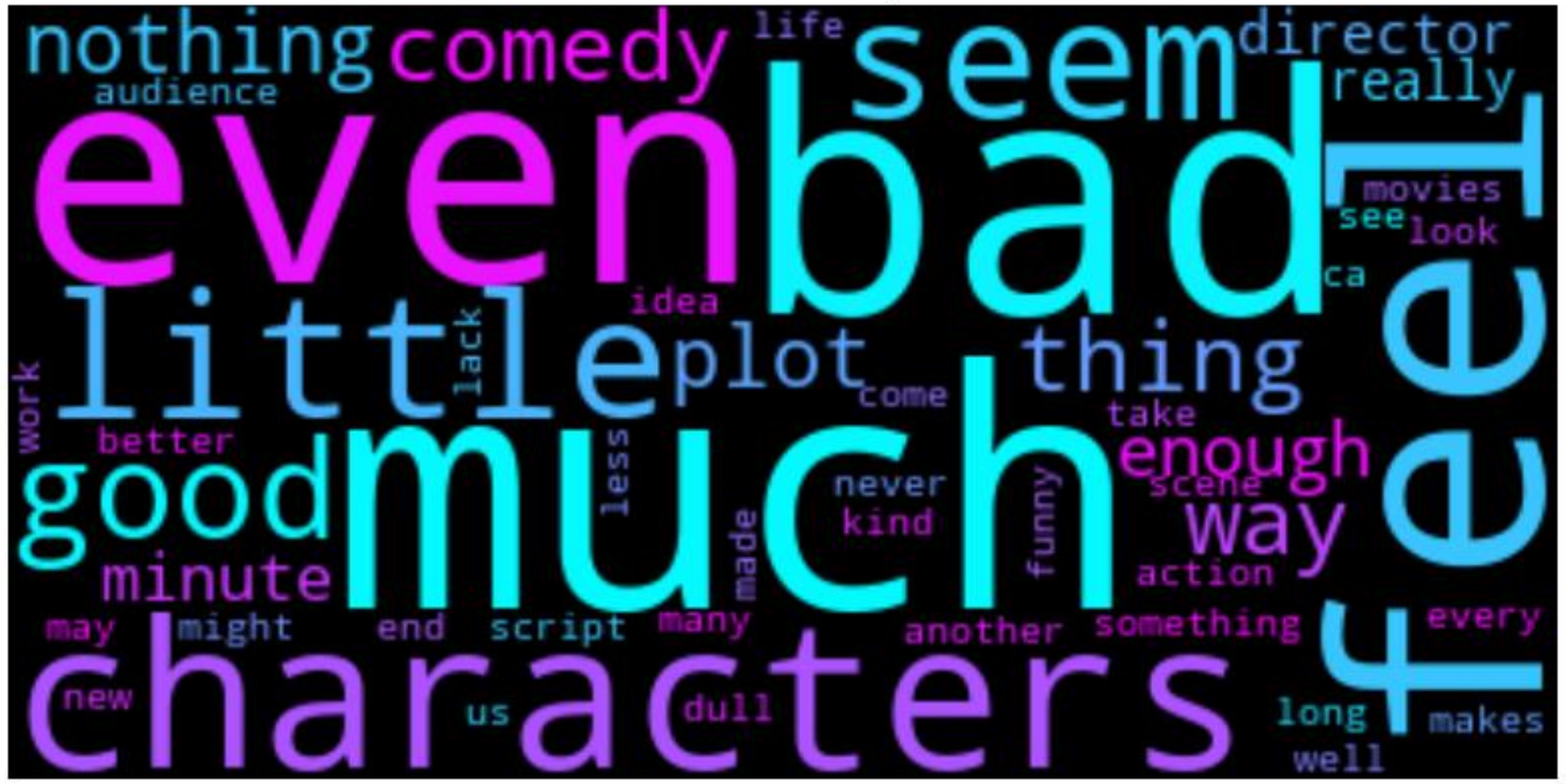
Most Common Words in Positive Reviews



Most Common Words in Negative Reviews



Most Common Words in Negative Reviews



```python
# Define a function to clean each review
def clean_text(text):
    # Check if the text is a string before applying string operations
    if isinstance(text, str):
        # Convert to lowercase
        text = text.lower()

        # Expand contractions (e.g., "can't" -> "cannot")
        text = fix(text)

        # Remove special characters, punctuation, and numbers using regex
        text = re.sub(r'[^a-z\s]', '', text)

        # Tokenize the text
        words = word_tokenize(text)

        # Remove stopwords and single-character tokens
        stop_words = set(stopwords.words('english'))

        # Custom stopwords (add "rrb", "lrb", and other irrelevant tokens)
        custom_stopwords = ['rrb', 'lrb', 'br','movie', 'one', 'character', 'make', 'time', 'story', 'film','nt']
        stop_words.update(custom_stopwords)

        # Remove stopwords and filter out words of length <= 1
        cleaned_words = [word for word in words if word not in stop_words and len(word) > 1]

        # Join the cleaned words back into a string
        cleaned_text = ' '.join(cleaned_words)

        return cleaned_text
    else:

        return ''

# Apply the cleaning function to the 'review' column
data['review'] = data['review'].apply(clean_text)
```

✓ 2.9s

Data Cleaning and Preprocessing

We gather relevant datasets from various sources and clean, normalize, and preprocess the data to ensure its quality and consistency. This step is crucial for training accurate and reliable AI models.

Preprocess the labeled data

- Tokenization and Lowercasing
- Stop Words
- Lemmatization
- `Vectorization`

```
data.duplicated().sum()
```

[5] ✓ 0.0s

```
11
```

```
data.drop_duplicates(inplace=True)
```

[6] ✓ 0.0s

```
data.isnull().sum()
```

[7] ✓ 0.0s

```
review    0
label     0
dtype: int64
```

```
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    tokens = word_tokenize(text.lower())
    filtered_tokens = [token for token in tokens if token not in stop_words]
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in filtered_tokens]
    return ' '.join(lemmatized_tokens)
```

[11] ✓ 0.0s

```
data['review'] = data['review'].apply(preprocess_text)
```

[12] ✓ 1.5s

Before and After Lemmatization

```
data=pd.concat([data_train,data_test,data_dev])
data
```

[4] ✓ 0.0s

	review	label
0	a stirring , funny and finally transporting re...	1
1	apparently reassembled from the cutting room f...	0
2	they presume their audience wo n't sit still f...	0
3	this is a visually stunning rumination on love...	1
4	jonathan parker 's bartleby should have been t...	1
...	...	...
867	something like scrubbing the toilet	0
868	smart , provocative and blisteringly funny	1
869	this one is definitely one to skip , even for ...	0
870	charles ' entertaining film chronicles seinfel...	1
871	an effectively creepy , fear inducing lrb not ...	1

9613 rows × 2 columns

```
data
```

[13] ✓ 0.0s

	review	label
0	stirring , funny finally transporting imaginin...	1
1	apparently reassembled cutting room floor give...	0
2	presume audience wo n't sit still sociology le...	0
3	visually stunning rumination love , memory , h...	1
4	jonathan parker 's bartleby end modern office ...	1
...	...	...
867	something like scrubbing toilet	0
868	smart , provocative blisteringly funny	1
869	one definitely one skip , even horror movie fa...	0
870	charles ' entertaining film chronicle seinfeld...	1
871	effectively creepy , fear inducing lrb fear re...	1

9602 rows × 2 columns

Vectorization

```
# Vectorization using TF-IDF
vectorizer = TfidfVectorizer(max_features=12491)
X_train_vectors = vectorizer.fit_transform(data_train['review'])
X_test_vectors = vectorizer.transform(data_test['review'])
X_dev_vectors = vectorizer.transform(data_dev['review'])
```

[15] ✓ 0.1s

```
# Fit your vectorizer on training data
vectorizer = TfidfVectorizer()
X_train_vectors = vectorizer.fit_transform(data_train['review'])  # Assuming X_train is your preprocessed training data
# Save the vectorizer to a file
with open('tfidf_vectorizer.pkl', 'wb') as f:
    pickle.dump(vectorizer, f)
```

[16] ✓ 0.0s

Model Development

We design and implement machine learning and deep learning models tailored to the specific requirements of the project.

Recommendations for selection

When selecting a model for sentiment analysis, consider the data size, feature complexity, and the need for interpretability. For large datasets with complex patterns, deep learning is advisable. For simpler problems, Logistic Regression or Naive Bayes may suffice. Ensure to validate models based on available computational resources and time.

```python
from sklearn.svm import SVC  # Import the correct class

# Train the SVM classifier on the training data
svm_classifier = SVC(kernel='linear')  # Use SVC, not svm
svm_classifier.fit(X_train_vectors, y_train)

# Evaluate the model on the development set
y_dev_pred = svm_classifier.predict(X_dev_vectors)

# Calculate accuracy on the dev set

dev_accuracy = accuracy_score(y_dev, y_dev_pred)
print(f"Dev Set Accuracy: {dev_accuracy}")

# After tuning hyperparameters, evaluate on the test set
y_test_pred = svm_classifier.predict(X_test_vectors)
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Test Set Accuracy: {test_accuracy}")
```

[18] ✓ 3.0s

```
Dev Set Accuracy: 0.7737321196358907
Test Set Accuracy: 0.7933368037480479
```

```python
# Predict the labels for the testing set
y_pred = svm_classifier.predict(X_test_vectors)
# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

[19] ✓ 0.4s

```
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.78      0.79       961
           1       0.78      0.81      0.80       960

    accuracy                           0.79      1921
   macro avg       0.79      0.79      0.79      1921
weighted avg       0.79      0.79      0.79      1921
```

```python
logistic_classifier_model = LogisticRegression()
logistic_classifier_model.fit(X_train_vectors, y_train)
# Evaluate the model on the development set
y_dev_pred = logistic_classifier_model.predict(X_dev_vectors)
# Calculate accuracy on the dev set
dev_accuracy = accuracy_score(y_dev, y_dev_pred)
print(f"Dev Set Accuracy: {dev_accuracy}")
logistic_pred = logistic_classifier_model.predict(X_test_vectors)
logistic_report = classification_report(y_test, logistic_pred)

print("***** Logistic Regression *****")
print(logistic_report)
```

[20] ✓ 0.0s

```
Dev Set Accuracy: 0.7737321196358907
***** Logistic Regression *****
              precision    recall  f1-score   support

           0       0.81      0.76      0.78       961
           1       0.77      0.82      0.80       960

    accuracy                           0.79      1921
   macro avg       0.79      0.79      0.79      1921
weighted avg       0.79      0.79      0.79      1921
```

```python
random_forest_model = RandomForestClassifier(criterion="gini", random_state=42)
random_forest_model.fit(X_train_vectors, y_train)
random_forest_pred = random_forest_model.predict(X_test_vectors)
random_forest_report = classification_report(y_test, random_forest_pred)

print("***** Random Forest *****")
print(random_forest_report)
```

[21] ✓ 5.6s

```
***** Random Forest *****
              precision    recall  f1-score   support

           0       0.72      0.77      0.75       961
           1       0.75      0.71      0.73       960

    accuracy                           0.74      1921
   macro avg       0.74      0.74      0.74      1921
weighted avg       0.74      0.74      0.74      1921
```

```python
# Assuming your input shape is determined by the size of the TF-IDF vectorizer (number of features)
input_shape = (X_train_vectors.shape[1],)

# Build the model
model = Sequential()

model.add(Dense(256, activation='relu', input_shape=input_shape, kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid'))

# Build the model explicitly
model.build(input_shape=input_shape)

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Summary of the model
model.summary()
```

✓ 0.3s

Result and Impact

We present the results of our AI project, showcasing improvements in performance, efficiency, and decision-making compared to traditional methods.

Model	Accuracy
Naive Bayes	79%
Support Vector Machines (SVM)	79%
Random Forest	74%
Logistic Regression	79%
Deep Learning Model	75.8%

- SVM: Effective in high-dimensional spaces; inefficient with large data.
- Logistic Regression: Simple, interpretable; struggles with complex patterns.
- Naive Bayes: Fast, good for text; assumes feature independence.
- Random Forest: Robust; slow to interpret.
- Deep Learning: Powerful; requires significant computational resources.

Challenges in Sentiment Analysis

Sentiment analysis faces challenges such as sarcasm, contextual meanings, and multilingual data. These factors can complicate model predictions, necessitating ongoing research and development to improve model robustness and accuracy.

The future of sentiment analysis

The future of sentiment analysis lies in advancements in AI and machine learning. As models become more sophisticated, they will better understand nuanced emotions and context, leading to more accurate sentiment predictions across diverse applications.


Thank You!