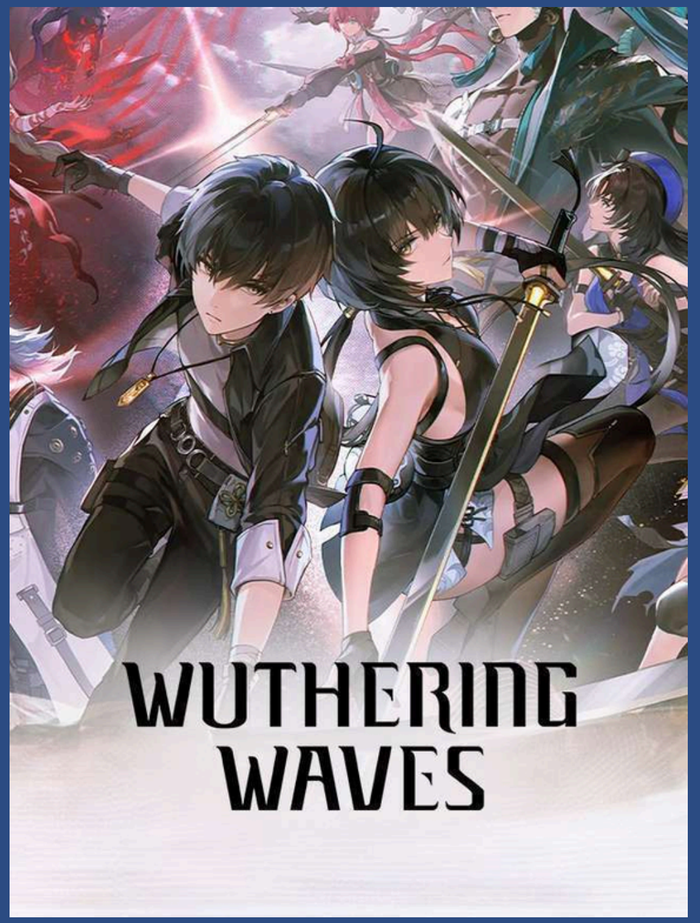
SENTIMENT ANALYSIS FOR WUTHERING WAVES

BY:
HARITS MUGHNI ZAKINU

BACKGROUND

Wuthering Waves is an action RPG released by **Kuro Games** in **May 2024**, featuring a theme of action and adventure. The game has garnered significant attention due to its unique blend of character-driven narratives and stunning visual storytelling. With a growing fan base and increasing engagement on its official YouTube channel, understanding player sentiment has become **crucial** for the developers to tailor future updates and content.



OBJECTIVE

Performing sentiment analysis on comments from the official Wuthering Waves YouTube channel, focusing on two main aspects: characters and story to understand player reactions using Random Forest Classifier and Support Vector Machine.

DATA COLLECTION & PREPARATION

DATA SCRAPING

Comments are taken from the official Wuthering Waves YouTube channel and saved in CSV format. The videos in question are titled as follows:

- “Wuthering Waves | Resonator Showcase | Changli — SEIZE THE INITIATIVE”,
- “Wuthering Waves Resonator Showcase | Jinhsi — HEED MY CALLING”,
- “Wuthering Waves | Resonator Showcase | Yinlin — SURRENDER OR DIE”,
- “Wuthering Waves | Resonator Showcase | Jiyan — THROUGH THE DARKEST OF NIGHTS”
- “Wuthering Waves | Resonator Showcase | Lingyang — Debut”,
- “Wuthering Waves | Resonator Showcase | Encore — BAA BAA!!”,
- “Wuthering Waves Story Cinematics | Pursuit”,
- “Wuthering Waves Story Cinematics | Daybreak”,
- “Wuthering Waves Featured Cinematics | AS FATE HAS DECREED”,
- “Wuthering Waves Story Cinematics | Battle Beneath the Crescent”,
- “Wuthering Waves Featured Cinematics | SAVING LIGHT”,
- “Opening Cinematics | Set Sail”

DATA SCRAPING

```python
# Taking comments from video ids for aspects of the character
comments_character = []
for video_id in video_ids_character:
    comments = video_comments(video_id, api_key)
    comments_character.extend(comments)

df_char = pd.DataFrame(comments_character, columns=['publishedAt', 'authorDisplayName', 'textDisplay'])
df_char.to_csv('/content/drive/MyDrive/Dataset/yt_comments_character.csv', index=False)

# Taking comments from video ids for story aspects
comments_story = []
for video_id in video_ids_story:
    comments = video_comments(video_id, api_key)
    comments_story.extend(comments)

df_story = pd.DataFrame(comments_story, columns=['publishedAt', 'authorDisplayName', 'textDisplay'])
df_story.to_csv('/content/drive/MyDrive/Dataset/yt_comments_story.csv', index=False)
```

```python
df_char = pd.read_csv('/content/drive/MyDrive/Dataset/yt_comments_character.csv')
df_story = pd.read_csv('/content/drive/MyDrive/Dataset/yt_comments_story.csv')
```

	publishedAt	authorDisplayName	textDisplay
0	2024-08-12T18:35:15Z	@erenop2319	Finally Got her after loosing the 50-50 welp a...
1	2024-08-12T15:13:30Z	@user-ie9xx5yo2z	I have to say that Changli's character mod...
2	2024-08-13T02:05:49Z	@avej99	I mean thats not hard to do tbh
3	2024-08-13T17:47:23Z	@WuWa-oh7rd	Agree
4	2024-08-12T06:05:15Z	@DonLuy	<a href="https://www.youtube.com/watch?v=jYjxj...

	publishedAt	authorDisplayName	textDisplay
0	2024-08-14T08:18:31Z	@vrika4280	I just finished the quest, and both this and J...
1	2024-08-13T01:16:21Z	@prevailege	I really like her design! It's just PERFECT!!
2	2024-08-12T06:51:42Z	@Ew-wth	Look at the replay amounts for each scene, lol...
3	2024-08-11T19:42:25Z	@S.k.Editor-14	BRO I CAN'T THE HAND MOMENT 😱🤚💦
4	2024-08-09T19:33:09Z	@CrimRui	I love it! Great stuff.

DATA UNDERSTANDING

All comments were collected on August 14, 2024, at 19:00 WIB, with a total of 14036 comments for the character aspect and 10737 comments for the story aspect.

```
df_char.shape
```

```
(14036, 3)
```

```
df_story.shape
```

```
(10737, 3)
```

LABELING DATA

Based on the criteria established by the VaderSentiment method, with the following guidelines:

- Positive Sentiment: compound score >= 0.05
- Neutral Sentiment: (compound score > -0.05) and (compound score < 0.05)
- Negative Sentiment: compound score <= -0.05

```python
def labeling(sentences):
  analyzer = SentimentIntensityAnalyzer()
  data = []
  for sentence in sentences:
    # Convert sentence to string if it's not already
    if not isinstance(sentence, str):
      sentence = str(sentence)
    vs = analyzer.polarity_scores(sentence)
    data.append(vs)
  return data
```

```python
def get_sentiment_3category(score):
    if score >= 0.05:
        return 'positive'
    elif score <= -0.05:
        return 'negative'
    else:
        return 'neutral'

df_char['sentiment'] = df_char['compound_score'].apply(get_sentiment_3category)
df_story['sentiment'] = df_story['compound_score'].apply(get_sentiment_3category)
```

DATA CLEANSING

There are some null values in both datasets, but we don't immediately remove these null values because the datasets will be processed through HTML tag removal, special character and number removal, case folding, repetitive character reduction, tokenization, filtering/stopwords, and lemmatization.

```
df_char.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14036 entries, 0 to 14035
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   publishedAt        14036 non-null  object 
 1   authorDisplayName  14008 non-null  object 
 2   textDisplay        14011 non-null  object 
 3   scores             14036 non-null  object 
 4   neg_score          14036 non-null  float64
 5   pos_score          14036 non-null  float64
 6   neu_score          14036 non-null  float64
 7   compound_score     14036 non-null  float64
 8   sentiment          14036 non-null  object 
dtypes: float64(4), object(5)
memory usage: 987.0+ KB
```

```
df_char.isnull().sum()
```

	0
publishedAt	0
authorDisplayName	28
textDisplay	25
scores	0
neg_score	0
pos_score	0
neu_score	0
compound_score	0
sentiment	0

dtype: int64

```
df_story.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10737 entries, 0 to 10736
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   publishedAt        10737 non-null  object 
 1   authorDisplayName  10734 non-null  object 
 2   textDisplay        10735 non-null  object 
 3   scores             10737 non-null  object 
 4   neg_score          10737 non-null  float64
 5   pos_score          10737 non-null  float64
 6   neu_score          10737 non-null  float64
 7   compound_score     10737 non-null  float64
 8   sentiment          10737 non-null  object 
dtypes: float64(4), object(5)
memory usage: 755.1+ KB
```

```
df_story.isnull().sum()
```

	0
publishedAt	0
authorDisplayName	3
textDisplay	2
scores	0
neg_score	0
pos_score	0
neu_score	0
compound_score	0
sentiment	0

dtype: int64

DATA CLEANSING

There were 5 duplicate entries in the character aspect dataset and 1 duplicate entry in the story aspect dataset. These duplicates were removed, resulting in 14,031 rows in the character aspect dataset and 10,736 rows in the story aspect dataset. Next, data preprocessing will be carried out as described in the previous slide.

```
df_char[df_char.duplicated(['authorDisplayName', 'textDisplay', 'publishedAt'])].shape
```

```
(5, 9)
```

```
df_char.drop_duplicates(['authorDisplayName', 'textDisplay', 'publishedAt'], inplace=True)
df_char.reset_index(drop=True, inplace=True)
df_char.shape
```

```
(14031, 9)
```

```
df_story[df_story.duplicated(['authorDisplayName', 'textDisplay', 'publishedAt'])].shape
```

```
(1, 9)
```

```
df_story.drop_duplicates(['authorDisplayName', 'textDisplay', 'publishedAt'], inplace=True)
df_story.reset_index(drop=True, inplace=True)
df_story.shape
```

```
(10736, 9)
```

PREPROCESSING

The text in the comments is processed by removing HTML tags, special characters such as numbers, punctuation, and symbols, leaving only the words. Then, capital letters are converted to lowercase, repetitive characters are reduced, text is tokenized, filtered for stopwords, and finally, lemmatized. Additionally, language detection is performed to filter the data based on language, ensuring that only comments in English are retained for further processing.

PREPROCESSING

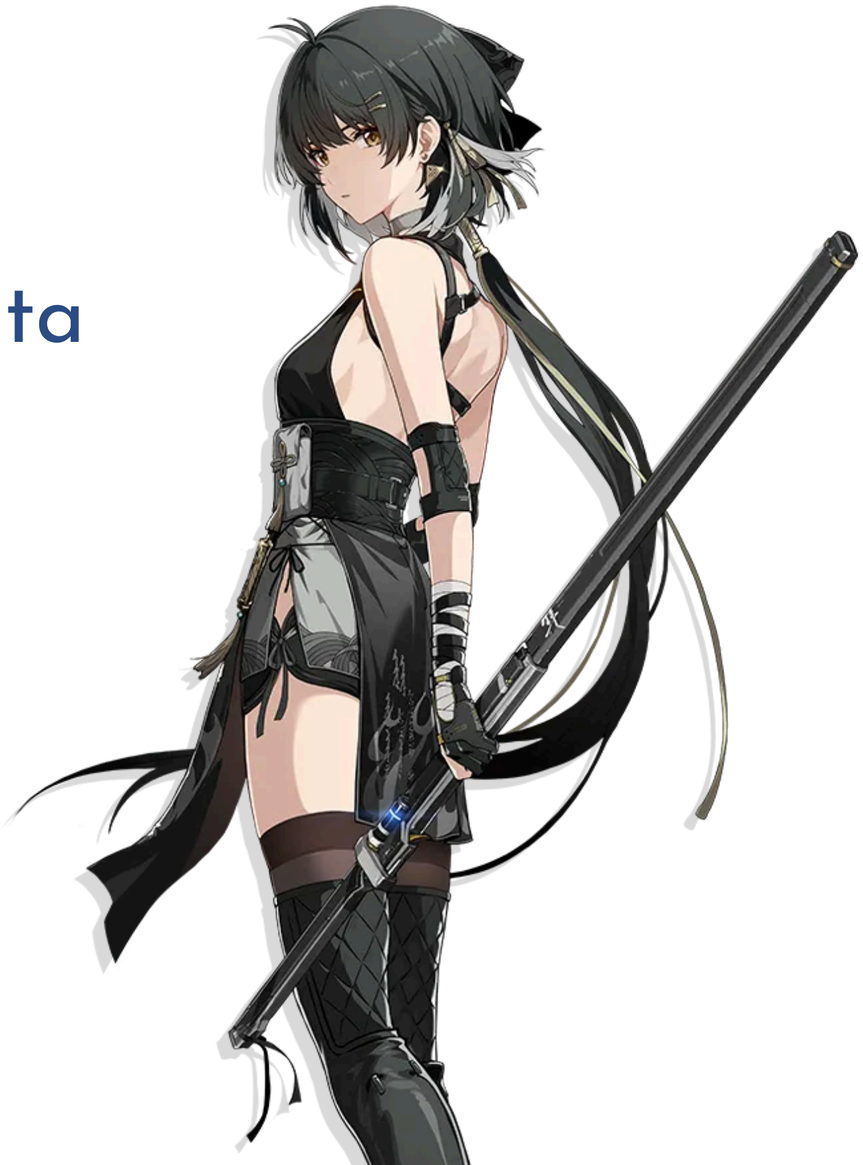
```python
def char_preprocessed_text(char_raw_text):

  if not isinstance(char_raw_text, str):     # Fixes float error if the input is not a string, just return an empty string
    return ''

  # Removes HTML Tags
  char_comments_text = BeautifulSoup(char_raw_text).get_text()

  # Removes non-letters/emojis, change to lowercase and then splitsit into tokens
  char_words_only = re.sub("[^a-zA-Z]", " ", char_comments_text)

  # Convert to lowercase
  words = char_words_only.lower().split()

  # Remove repetitive characters
  words = [re.sub(r'(.)\1{2,}', r'\1', word) for word in words]

  # Tokenization
  tokens = word_tokenize(" ".join(words))

  # Removing stopwords like the, in, of etc.
  char_stop_words = set(stopwords.words('english'))
  char_fil_words = [word for word in tokens if word not in char_stop_words]

  # Lemmatization
  lemmatizer = WordNetLemmatizer()
  lemmatized_words = [lemmatizer.lemmatize(word) for word in char_fil_words]

  return " ".join(lemmatized_words)
```

```python
def is_english(text):
    try:
        return detect(text) == 'en'
    except:
        return False

# Create a boolean mask indicating which comments are in English
english_mask_char = df_char['textDisplay'].apply(is_english)

# Filter the DataFrame to keep only English comments
df_char = df_char[english_mask_char]

# Reset the index to ensure it is sequential and starts at 0
df_char.reset_index(drop=True, inplace=True)

# Create a mask for non-English textDisplay
non_english_mask_char = ~english_mask_char

# Use the mask to locate the indexes of non-English textDisplay
indexes_of_non_english_char = df_char[non_english_mask_char].index

# Access and view the non-English content using the indexes
non_english_textDisplay_char = df_char.loc[indexes_of_non_english_char, 'textDisplay']
```

```python
def story_preprocessed_text(story_raw_text):

  if not isinstance(story_raw_text, str):     # Fixes float error if the input is not a string, just return an empty string
    return ''

  # Removes HTML Tags
  story_comments_text = BeautifulSoup(story_raw_text).get_text()

  # Removes non-letters/emojis, change to lowercase and then splitsit into tokens
  story_words_only = re.sub("[^a-zA-Z]", " ", story_comments_text)

  # Convert to lowercase
  words = story_words_only.lower().split()

  # Remove repetitive  characters
  words = [re.sub(r'(.)\1{2,}', r'\1', word) for word in words]

  # Tokenization
  tokens = word_tokenize(" ".join(words))

  # Removing stopwords like the, in, of etc.
  story_stop_words = set(stopwords.words('english'))
  char_fil_words = [word for word in tokens if word not in story_stop_words]

  # Lemmatization
  lemmatizer = WordNetLemmatizer()
  lemmatized_words = [lemmatizer.lemmatize(word) for word in char_fil_words]

  return " ".join(lemmatized_words)
```

```python
def is_english(text):
    try:
        return detect(text) == 'en'
    except:
        return False

# Create a boolean mask indicating which comments are in English
english_mask_char = df_story['textDisplay'].apply(is_english)

# Filter the DataFrame to keep only English comments
df_story = df_story[english_mask_char]

# Reset the index to ensure it is sequential and starts at 0
df_story.reset_index(drop=True, inplace=True)

# Create a mask for non-English textDisplay
non_english_mask_char = ~english_mask_char

# Use the mask to locate the indexes of non-English textDisplay
indexes_of_non_english_story = df_story[non_english_mask_char].index

# Access and view the non-English content using the indexes
non_english_textDisplay_story = df_story.loc[indexes_of_non_english_story, 'textDisplay']
```

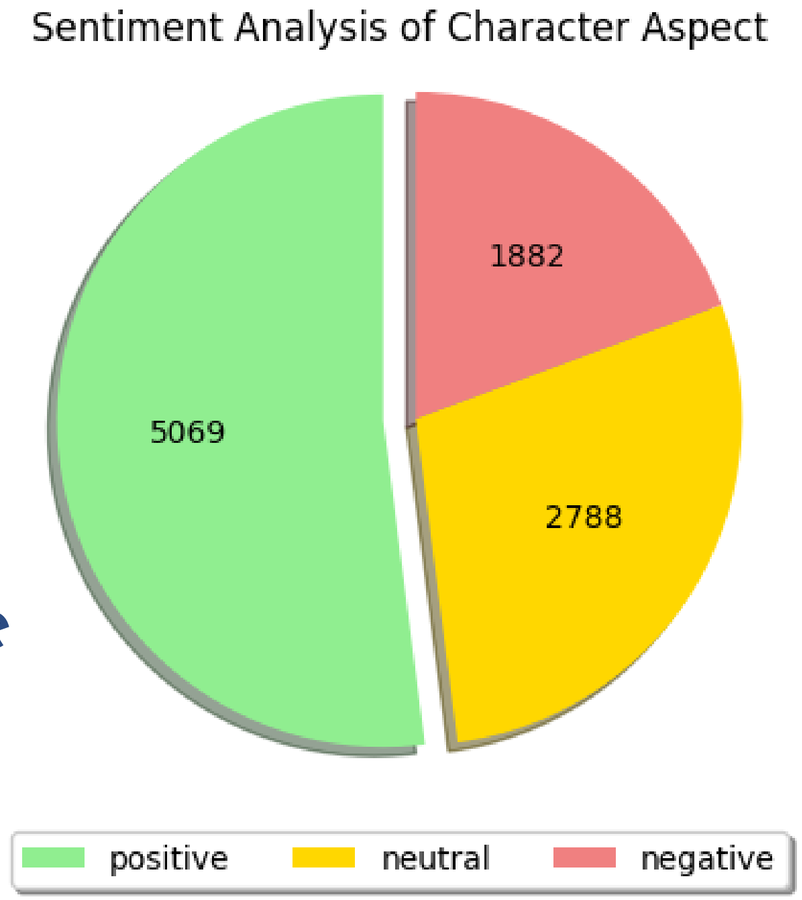
After removing duplicate data and preprocessing, there are 9739 rows of data remaining for the **character aspect** and 7776 rows of data for the **story aspect**. These will be used to analyze player reactions



EXPLORATION DATA ANALYSIS

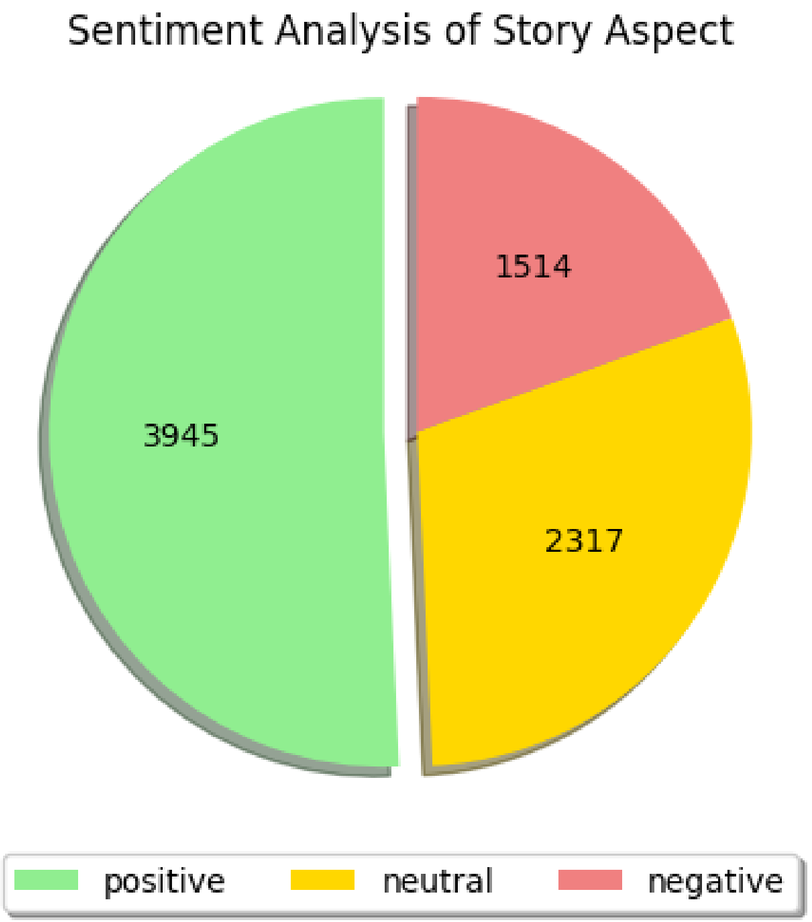
CALCULATE TOTAL SENTIMENT

The character aspect received 5069 positive sentiments (52.05%), 2788 neutral sentiments (28.63%), and 1882 negative sentiments (19.32%). Based on these results, players are quite satisfied with the characters in the game.

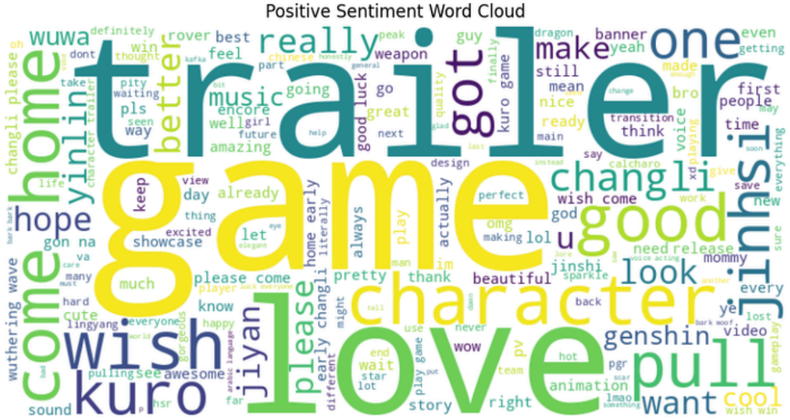


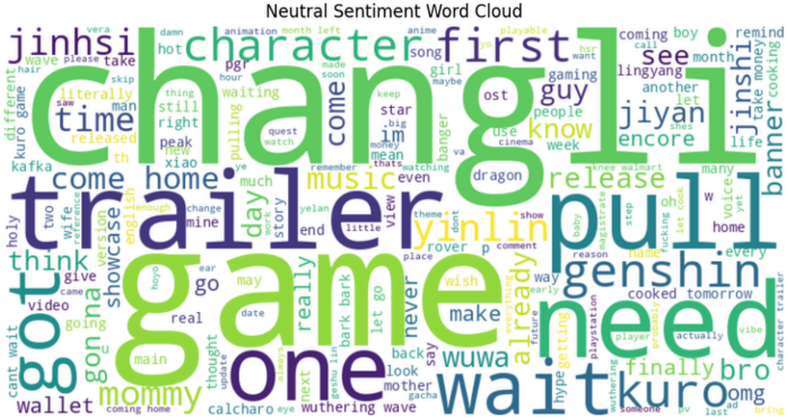
CALCULATE TOTAL SENTIMENT

The story aspect received 3945 positive sentiments (50.74%), 2317 neutral sentiments (29.80%), and 1514 negative sentiments (19.46%). Based on these results, players are quite satisfied with the story in the game.



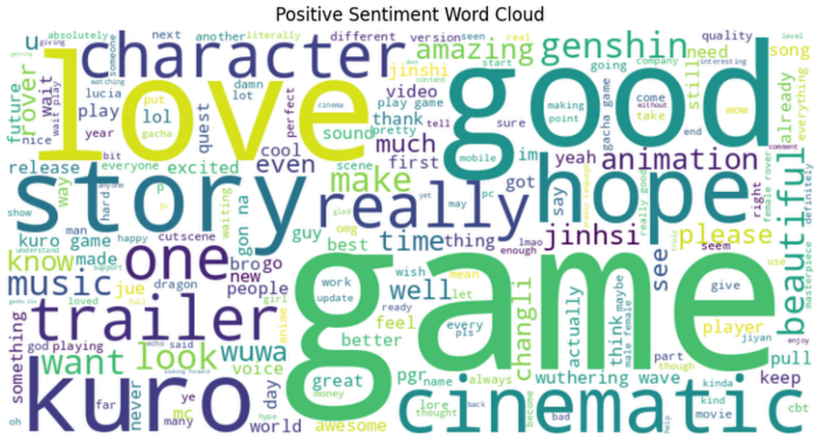
WORD CLOUD FOR CHARACTER ASPECT

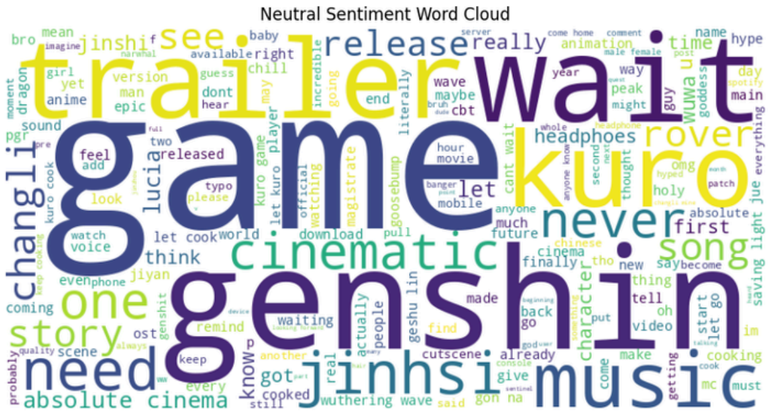


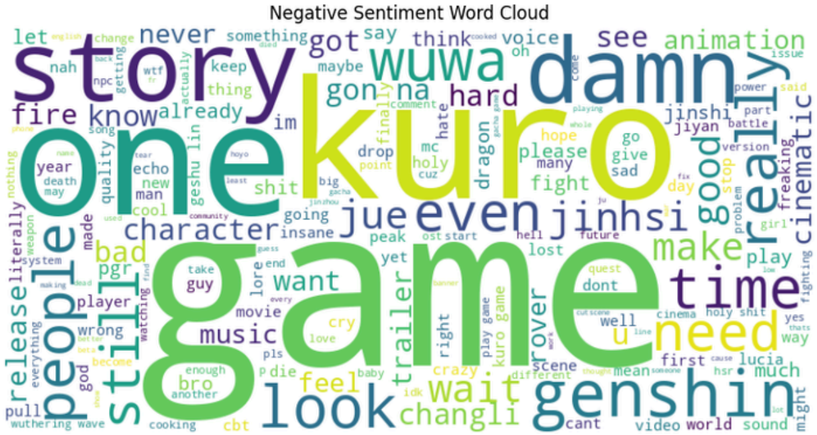


- In the positive sentiment class for the character aspect, characters like **Jinshi, Changli, Yinlin, and Jiyan** are mentioned the most, indicating that these characters are the most **popular**.
- In the negative sentiment class, the character **Changli** has the largest font size, indicating that there are quite a few players who are **not very happy** with this character.

WORD CLOUD FOR STORY ASPECT







- In the positive sentiment class for the story aspect, words like “**good**”, “**love**”, **and** “**cinematic**” are mentioned the most, indicating that many users have a **high appreciation** for the quality of the game's story.
- In the negative sentiment class, the frequent mention of words like **"damn", "really", and "good"** suggests that despite some positive elements, users are largely **dissatisfied or frustrated** with the story. They believe the story could be improved with certain changes or enhancements.

PRE-MODELLING

SPLITTING DATA

The data is divided into 80% for training and 20% for testing in the modeling.

```
# Split the data into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X_char, y_char, test_size=0.2, random_state=42)
```

```
# Split the data into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X_story, y_story, test_size=0.2, random_state=42)
```

FEATURE EXTRACTION

Next, feature extraction was performed using TF-IDF.

```
# Apply TF-IDF to the text for the training set and test set separately
tfidf = TfidfVectorizer()

# Fit and transform on training data and test data
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
```

```
# Apply TF-IDF to the text for the training set and test set separately
tfidf2 = TfidfVectorizer()

# Fit and transform on training data and test data
X_train_tfidf2 = tfidf2.fit_transform(X_train)
X_test_tfidf2 = tfidf2.transform(X_test)
```

BALANCING DATA WITH SMOTE

Since the number of negative and neutral sentiments is much lower compared to positive sentiments in both aspects, oversampling with SMOTE was performed to balance the data.

Before SMOTE (character aspect):

```
sentiment
positive    5069
neutral     2788
negative    1882
Name: count, dtype: int64
```

After SMOTE (character aspect):

```
sentiment
negative    4088
neutral     4088
positive    4088
Name: count, dtype: int64
```

Before SMOTE (story aspect):

```
sentiment
positive    3945
neutral     2317
negative    1514
Name: count, dtype: int64
```

After SMOTE (story aspect):

```
sentiment
negative    3178
neutral     3178
positive    3178
Name: count, dtype: int64
```

MODELING

RANDOM FOREST (CHARACTER ASPECT)

Train the Model

```
rf_model = RandomForestClassifier().fit(X_train_resampled, y_train_resampled)
```

Predict Data Training

```
y_train_pred = rf_model.predict(X_train_tfidf)
```

Predict Data Test

```
y_test_pred = rf_model.predict(X_test_tfidf)
```

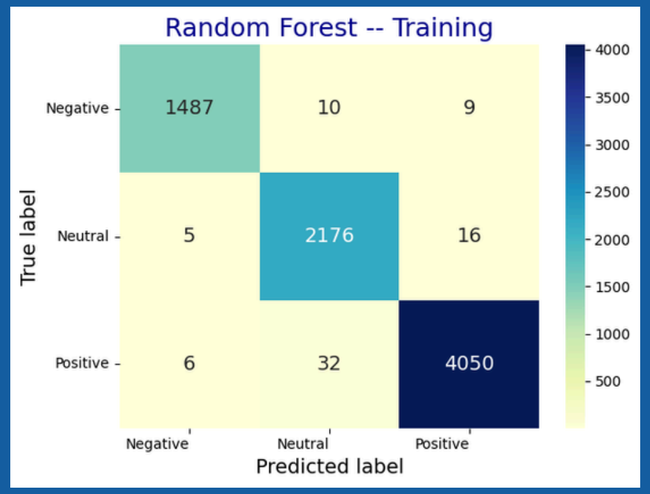
Evaluate Model

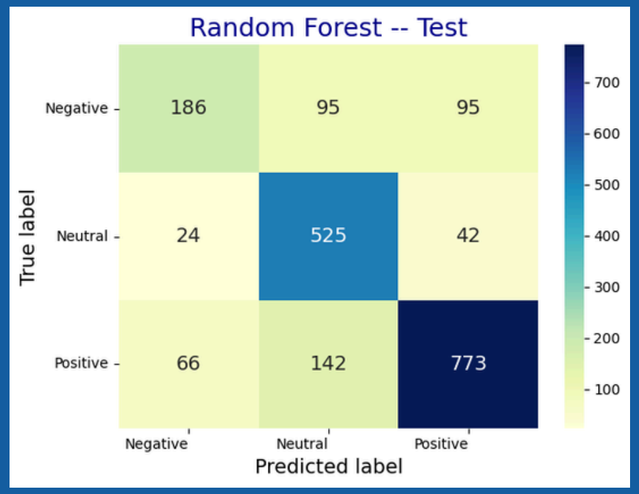
```
print("Classification Report Random Forest -- Training:\n", classification_report(y_train, y_train_pred))
```

```
print("Classification Report Random Forest -- Test:\n", classification_report(y_test, y_test_pred))
```

RANDOM FOREST (CHARACTER ASPECT)

Confusion Matrix





Classification Report Random Forest -- Training:

	precision	recall	f1-score	support
negative	0.99	0.99	0.99	1506
neutral	0.98	0.99	0.99	2197
positive	0.99	0.99	0.99	4088
accuracy			0.99	7791
macro avg	0.99	0.99	0.99	7791
weighted avg	0.99	0.99	0.99	7791

Classification Report Random Forest -- Test:

	precision	recall	f1-score	support
negative	0.67	0.49	0.57	376
neutral	0.69	0.89	0.78	591
positive	0.85	0.79	0.82	981
accuracy			0.76	1948
macro avg	0.74	0.72	0.72	1948
weighted avg	0.77	0.76	0.76	1948

SUPPORT VECTOR MACHINE (CHARACTER ASPECT)

Train the Model

```
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train_resampled, y_train_resampled)
y_train_pred = svm_model.predict(X_train_tfidf)
```

Predict Data Training

```
y_train_pred = svm_model.predict(X_train_tfidf)
```

Predict Data Test

```
y_test_pred = svm_model.predict(X_test_tfidf)
```

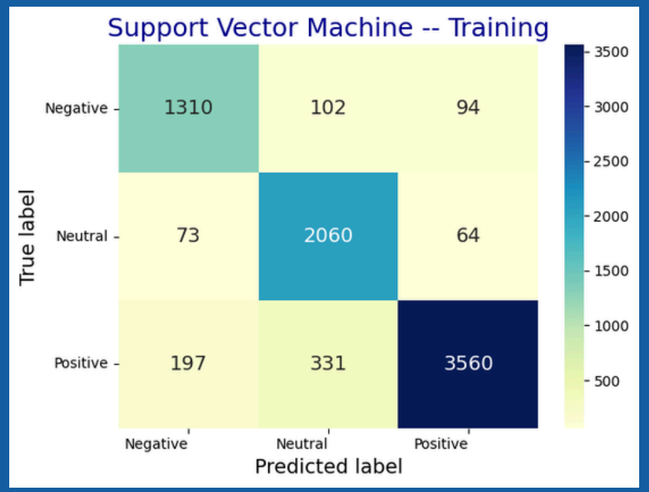
Evaluate Model

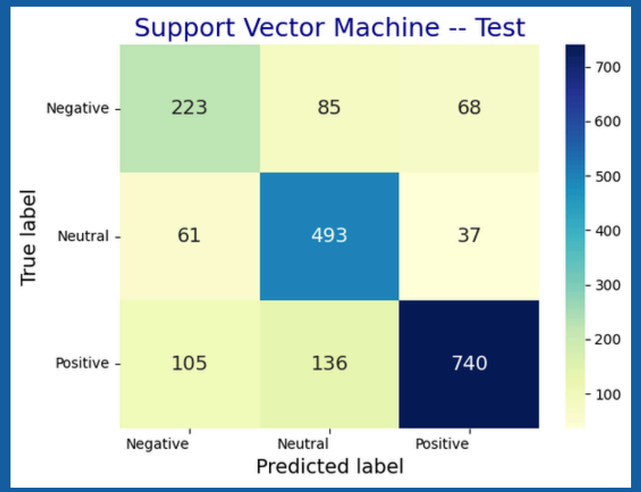
```
print("Classification Report Support Vector Machine -- Training:\n",
      classification_report(y_train, y_train_pred))
```

```
print("Classification Report Support Vector Machine -- Test:\n",
      classification_report(y_test, y_test_pred))
```

SUPPORT VECTOR MACHINE (CHARACTER ASPECT)

Confusion Matrix





Classification Report Support Vector Machine -- Training:

	precision	recall	f1-score	support
negative	0.83	0.87	0.85	1506
neutral	0.83	0.94	0.88	2197
positive	0.96	0.87	0.91	4088
accuracy			0.89	7791
macro avg	0.87	0.89	0.88	7791
weighted avg	0.90	0.89	0.89	7791

Classification Report Support Vector Machine -- Test:

	precision	recall	f1-score	support
negative	0.57	0.59	0.58	376
neutral	0.69	0.83	0.76	591
positive	0.88	0.75	0.81	981
accuracy			0.75	1948
macro avg	0.71	0.73	0.72	1948
weighted avg	0.76	0.75	0.75	1948

RANDOM FOREST (STORY ASPECT)

Train the Model

```
rf_model2 = RandomForestClassifier().fit(X_train_resampled, y_train_resampled)
```

Predict Data Training

```
y_train_pred2= rf_model2.predict(X_train_tfidf2)
```

Predict Data Test

```
y_test_pred2= rf_model2.predict(X_test_tfidf2)
```

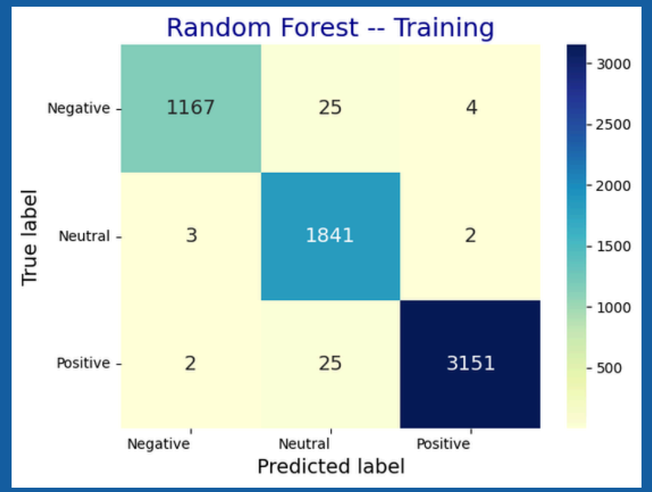
Evaluate Model

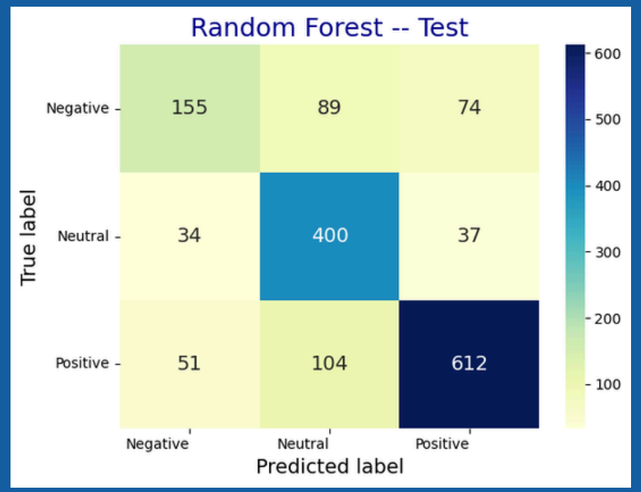
```
print("Classification Report Random Forest -- Training:\n",
      classification_report(y_train, y_train_pred2))
```

```
print("Classification Report Random Forest -- Test:\n",
      classification_report(y_test, y_test_pred2))
```

RANDOM FOREST (STORY ASPECT)

Confusion Matrix





Classification Report Random Forest -- Training:

	precision	recall	f1-score	support
negative	1.00	0.98	0.99	1196
neutral	0.97	1.00	0.99	1846
positive	1.00	0.99	0.99	3178
accuracy			0.99	6220
macro avg	0.99	0.99	0.99	6220
weighted avg	0.99	0.99	0.99	6220

Classification Report Random Forest -- Test:

	precision	recall	f1-score	support
negative	0.65	0.49	0.56	318
neutral	0.67	0.85	0.75	471
positive	0.85	0.80	0.82	767
accuracy			0.75	1556
macro avg	0.72	0.71	0.71	1556
weighted avg	0.75	0.75	0.75	1556

SUPPORT VECTOR MACHINE (STORY ASPECT)

Train the Model

```
svm_model2 = SVC(kernel='linear', random_state=42).fit(X_train_resampled, y_train_resampled)
```

Predict Data Training

```
y_train_pred2 = svm_model2.predict(X_train_tfidf2)
```

Predict Data Test

```
y_test_pred2 = svm_model2.predict(X_test_tfidf2)
```

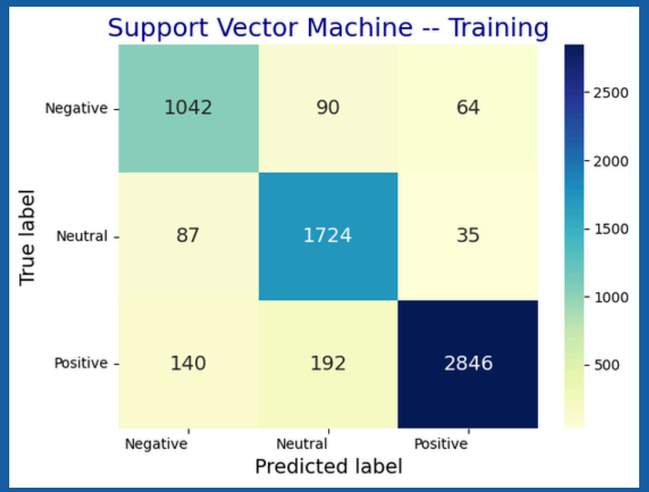
Evaluate Model

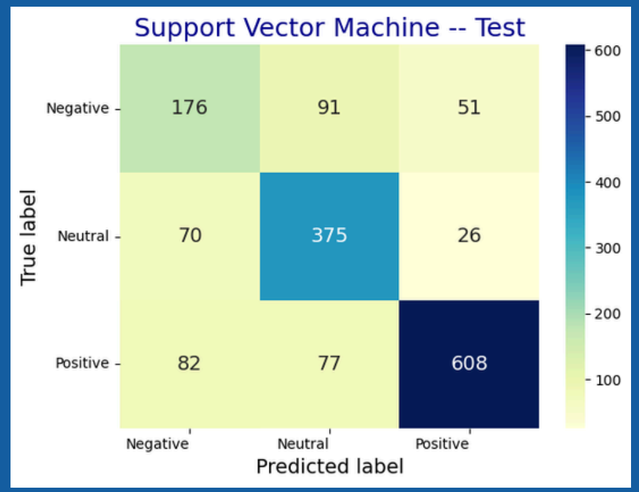
```
print("Classification Report Support Vector Machine -- Training:\n",
      classification_report(y_train, y_train_pred2))
```

```
print("Classification Report Support Vector Machine -- Test:\n",
      classification_report(y_test, y_test_pred2))
```

SUPPORT VECTOR MACHINE (STORY ASPECT)

Confusion Matrix





Classification Report Support Vector Machine -- Training:

	precision	recall	f1-score	support
negative	0.82	0.87	0.85	1196
neutral	0.86	0.93	0.90	1846
positive	0.97	0.90	0.93	3178
accuracy			0.90	6220
macro avg	0.88	0.90	0.89	6220
weighted avg	0.91	0.90	0.90	6220

Classification Report Support Vector Machine -- Test:

	precision	recall	f1-score	support
negative	0.54	0.55	0.54	318
neutral	0.69	0.80	0.74	471
positive	0.89	0.79	0.84	767
accuracy			0.74	1556
macro avg	0.70	0.71	0.71	1556
weighted avg	0.76	0.74	0.75	1556

FINDING THE BEST MODEL

Character Aspect

Model	Training Accuracy	Test Accuracy
Random Forest	99%	76%
Support Vector Machine	89%	75%

Story Aspect

Model	Training Accuracy	Test Accuracy
Random Forest	99%	75%
Support Vector Machine	90%	74%

CONCLUSION

- Although the game **Wuthering Waves** is interesting and less than three month old, **19.32%** of viewers have a negative sentiment towards the **character aspect**, and **19.46%** of viewers have a negative sentiment towards the **story aspect**.
- Even though **Random Forest** has higher accuracy on both the training set and test set, **SVM** provides more stable performance and shows better generalization ability. Therefore, **SVM can be considered the best model** for sentiment analysis on the character and story aspects due to its stability and the absence of significant overfitting indications.

THANK YOU