Video Game Sentiment Analysis

Satyen Sabnis

WHAT IS SENTIMENT ANALYSIS?

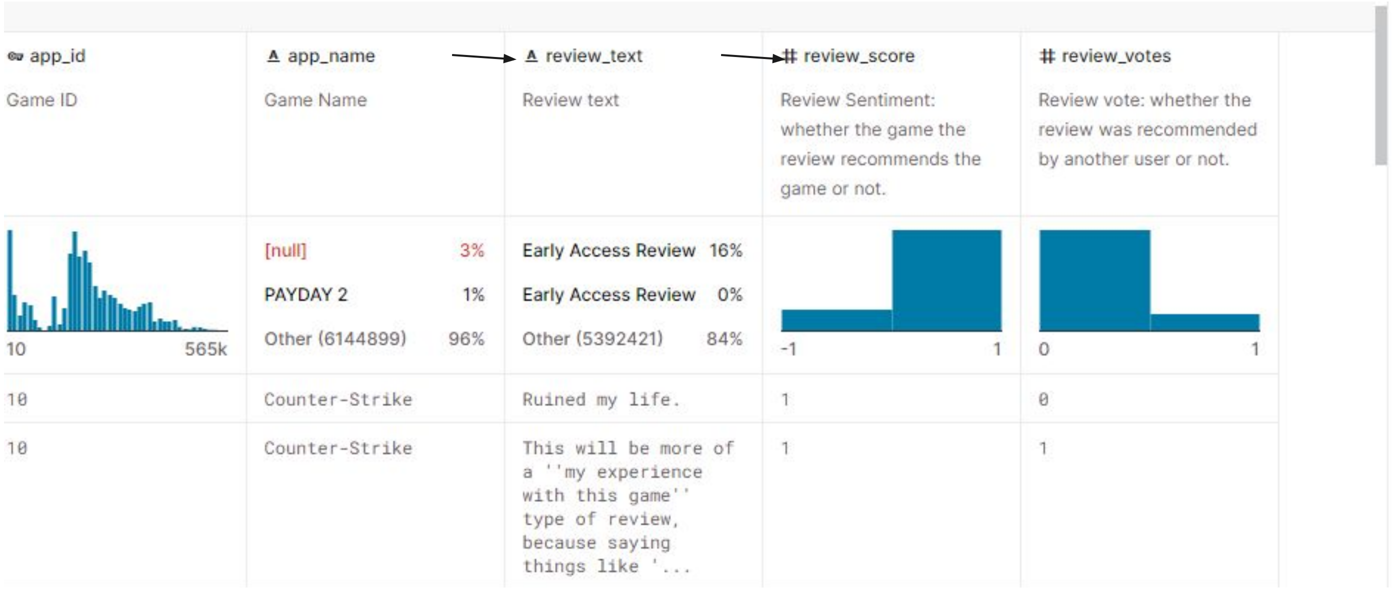
- ➔ Process of analyzing text to determine the **emotional tone**: positive, negative, and neutral of the user who wrote it
- ➔ Often used in businesses to determine **how well their product is being received** and what changes they can make to improve their product based on customer feedback/review

Problem/Goal

Imagine you are a data scientist in a video game company, your task is to determine how well a new video game (Star Wars Jedi: Survivor) they just deployed is being perceived through Twitter. Determine key insights on the positives of the game and what are some areas of the game that needs improvement on.

STEAM REVIEWS DATASET

Size: 6,417,106 rows × 5 columns → (Subset) 50,000 rows × 2 columns



app_id	app_name	review_text	review_score	review_votes
Game ID	Game Name	Review text	Review Sentiment: whether the game the review recommends the game or not.	Review vote: whether the review was recommended by another user or not.
10 565k	[null] 3% PAYDAY 2 1% Other (6144899) 96%	Early Access Review 16% Early Access Review 0% Other (5392421) 84%	-1 1	0 1
10	Counter-Strike	Ruined my life.	1	0
10	Counter-Strike	This will be more of a ''my experience with this game'' type of review, because saying things like '...	1	1

Key Feature: review_text - steam game reviews

Target Variable: review_score - Whether the user recommended the game or not i.e the sentiment of the game

Main Objective - Predict how well/not well Star Wars: Jedi Survivor is being received by users through Twitter.

I used the the steam reviews dataset to train my machine learning model using review_text as the feature and review_score as the predictor. A game is normally recommended or not based on how much the user enjoyed it, i.e. the sentiment. That same model will then be used to predict the sentiment of twitter users of Star Wars: Jedi Survivor

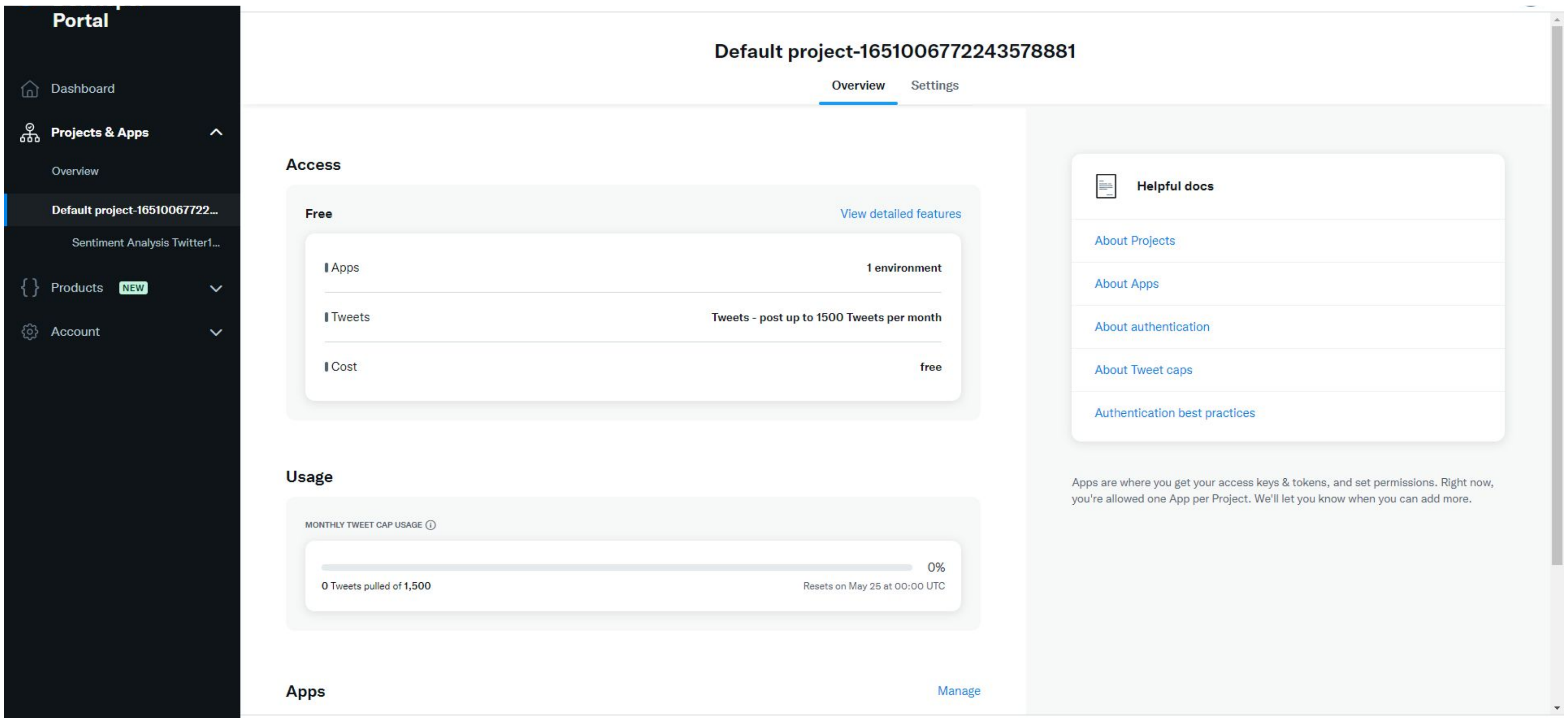
Source: Kaggle Steam Review Dataset

METHODOLOGY IN PERFORMING SENTIMENT ANALYSIS

- **Method 1:** TextBlob
  - Used textblob, a python library used for processing textual data, to perform sentiment analysis on tweets
  - Uses a pre-trained machine learning model, and gives a sentiment score (- Negative Sentiment, + Positive Sentiment, 0 Neutral Sentiment) and subjectivity score (0: Objective, 1: Subjective)
  - Calculate accuracy based on steam reviews already labeled data set

- **Method 2/3:** SkLearn(LinearSVC/Logistic Regression)
  - Train and tested my own machine learning model using sklearn to predict the sentiments on tweets
  - Used a steam game reviews dataset to help train my model and then test on that data
  - Calculate accuracy based on labeled data set

GETTING TWEETS FROM TWITTER API



Portal

Dashboard

Projects & Apps

Overview

Default project-16510067722...

Sentiment Analysis Twitter1...

Products NEW

Account

Default project-1651006772243578881

Overview Settings

Access

Free View detailed features

Apps	1 environment
Tweets	Tweets - post up to 1500 Tweets per month
Cost	free

Usage

MONTHLY TWEET CAP USAGE

0%

0 Tweets pulled of 1,500

Resets on May 25 at 00:00 UTC

Apps Manage

Helpful docs

About Projects

About Apps

About authentication

About Tweet caps

Authentication best practices

Apps are where you get your access keys & tokens, and set permissions. Right now, you're allowed one App per Project. We'll let you know when you can add more.

PREPROCESSING/CLEANING

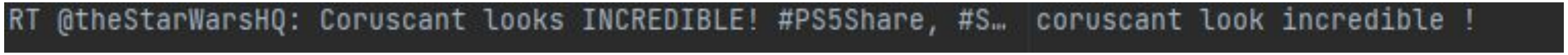
Natural Language Processing

- ➔ Before performing sentiment analysis you need to make sure your data is processed/cleaned
- ➔ Their is a lot of **redundant** and **unnecessary** information in text that should not be included, these are called **stop words**
- ➔ Preprocessing text helps improve **accuracy** of your model

Looked through twitter data and identified any words/symbols that were unnecessary and removed them from the tweet

Ex: RT, @, http://, # (any words starting with these symbols were unnecessary info)

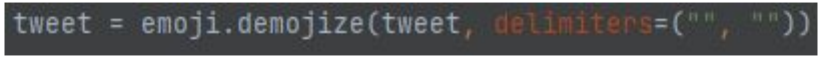
Raw tweet **Cleaned Tweet**



```
RT @theStarWarsHQ: Coruscant looks INCREDIBLE! #PS5Share, #S…   coruscant look incredible !
```

EMOJI/LANGUAGE PROCESSING

- **20.69%** of tweets contains at least one emoji, my twitter data on the game Star Wars Jedi: Survivor was no different
- Can gain **insights** from emoji's to use for sentiment analysis
    - By themselves emojis don't provide any information but you can convert them into raw text to identify the emotions in words that is attached to that emoji



```
tweet = emoji.demojize(tweet, delimiters=("", ""))
```

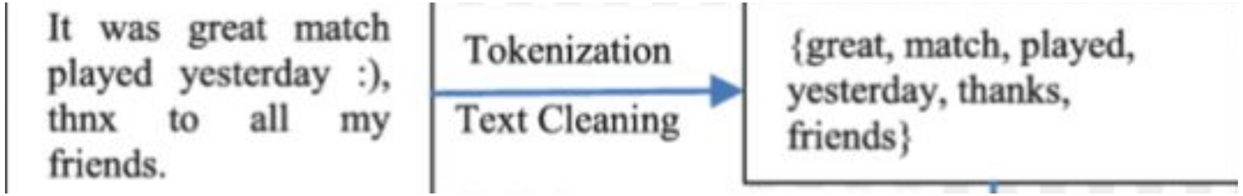
❤️ ➔ **red_heart**

- Translated all the tweets to english as this is the language the analysis was being done

TOKENIZING AND LEMMATIZING TEXT

Used tokenization to break down the tweets into words

- This helps you to normalize and better analyze the text by looking at each word


It was great match played yesterday :), thnx to all my friends.
Tokenization
Text Cleaning
{great, match, played, yesterday, thanks, friends}

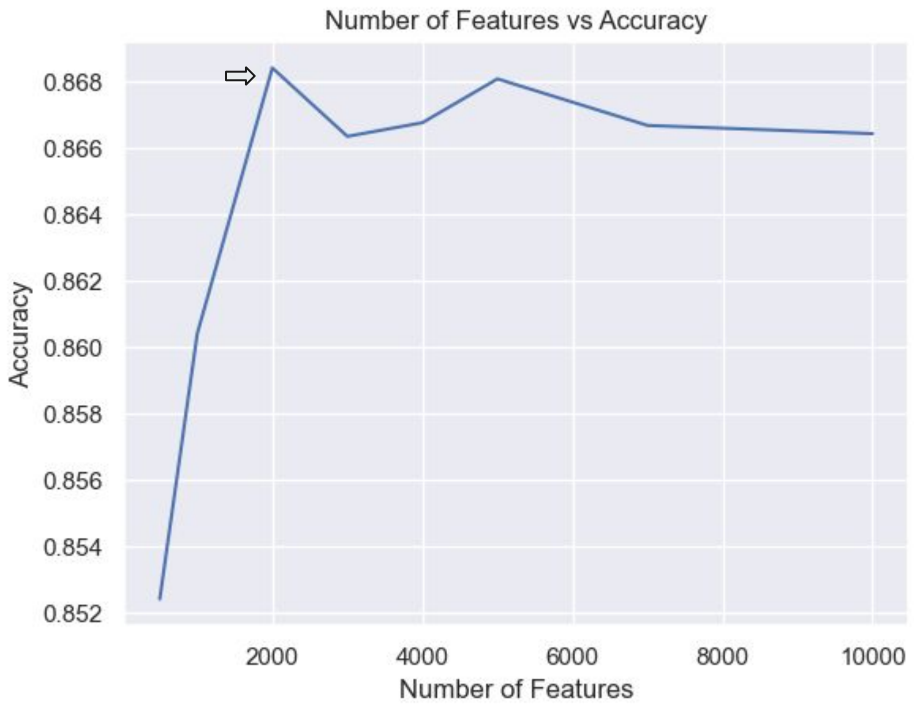
Used lemmatization to reduce the word to its root form

- Reduces the number of unique words and pinpoints analysis on a small subset of words

EX: CATS -> CAT

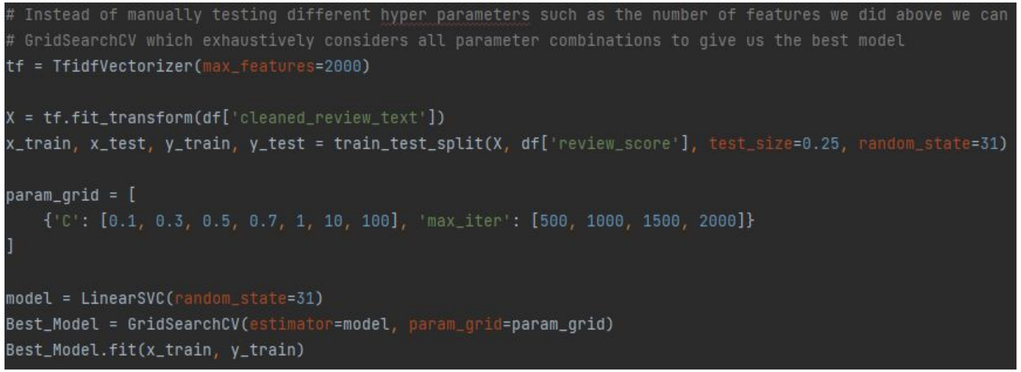
TRAINING BEST MACHINE LEARNING MODEL

- Used a **TfidfVectorizer**, transforms a collection of raw documents into a matrix of TF-IDF features
    - Calculates the importance of a word in text
- Used **LinearSVC** as a classifier to predict the sentiment, applied Grid Search to test different hyperparameters that would lead to the best model(highest accuracy)
- In addition, to help determine the best model I looked at how the max number of features used by the vectorizer affected the accuracy of the model
    - Ex: n number of features represents the n most frequently occurring words in the document as features



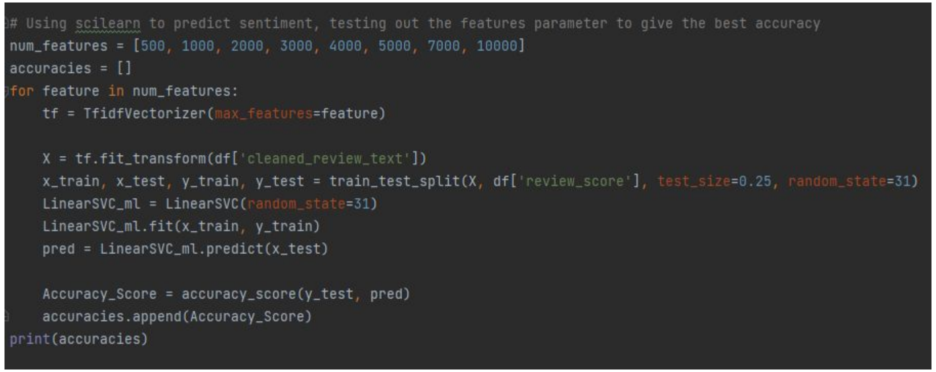
Overview Of Implementation

Grid Search



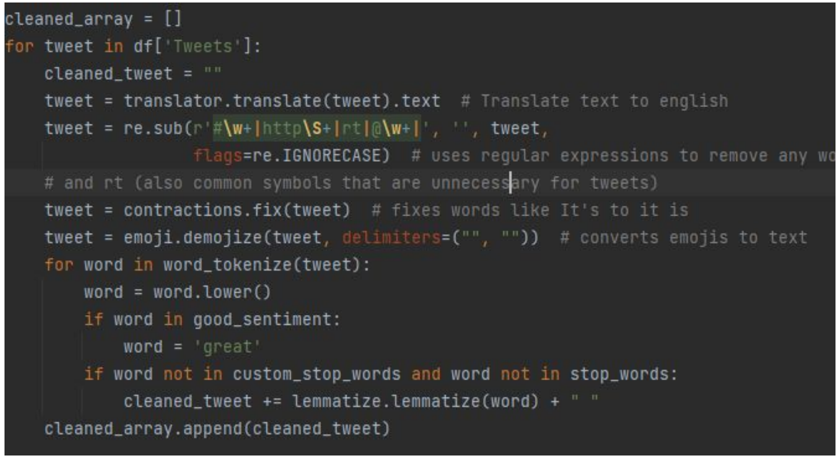
```
# Instead of manually testing different hyper parameters such as the number of features we did above we can
# GridSearchCV which exhaustively considers all parameter combinations to give us the best model
tf = TfidfVectorizer(max_features=2000)

X = tf.fit_transform(df['cleaned_review_text'])
x_train, x_test, y_train, y_test = train_test_split(X, df['review_score'], test_size=0.25, random_state=31)

param_grid = [
    {'C': [0.1, 0.3, 0.5, 0.7, 1, 10, 100], 'max_iter': [500, 1000, 1500, 2000]}
]

model = LinearSVC(random_state=31)
Best_Model = GridSearchCV(estimator=model, param_grid=param_grid)
Best_Model.fit(x_train, y_train)
```

Best Number of Features



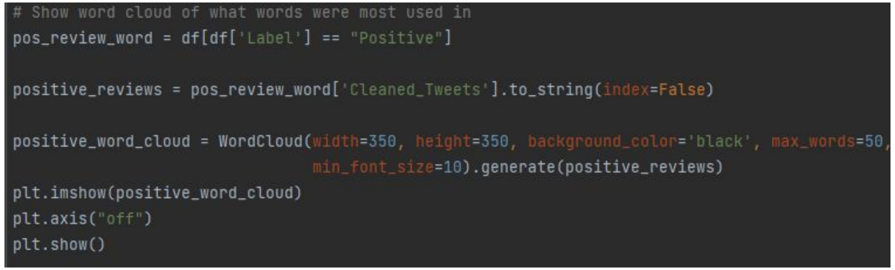
```
# Using scilearn to predict sentiment, testing out the features parameter to give the best accuracy
num_features = [500, 1000, 2000, 3000, 4000, 5000, 7000, 10000]
accuracies = []
for feature in num_features:
    tf = TfidfVectorizer(max_features=feature)

    X = tf.fit_transform(df['cleaned_review_text'])
    x_train, x_test, y_train, y_test = train_test_split(X, df['review_score'], test_size=0.25, random_state=31)
    LinearSVC_ml = LinearSVC(random_state=31)
    LinearSVC_ml.fit(x_train, y_train)
    pred = LinearSVC_ml.predict(x_test)

    Accuracy_Score = accuracy_score(y_test, pred)
    accuracies.append(Accuracy_Score)
print(accuracies)
```

Pre-Processing



```
cleaned_array = []
for tweet in df['Tweets']:
    cleaned_tweet = ""
    tweet = translator.translate(tweet).text  # Translate text to english
    tweet = re.sub(r'#\w+|http\S+|rt|@\w+|', '', tweet,
                   flags=re.IGNORECASE)  # uses regular expressions to remove any wo
    # and rt (also common symbols that are unnecessary for tweets)
    tweet = contractions.fix(tweet)  # fixes words like It's to it is
    tweet = emoji.demojize(tweet, delimiters=("", ""))  # converts emojis to text
    for word in word_tokenize(tweet):
        word = word.lower()
        if word in good_sentiment:
            word = 'great'
        if word not in custom_stop_words and word not in stop_words:
            cleaned_tweet += lemmatize.lemmatize(word) + " "
    cleaned_array.append(cleaned_tweet)
```

Word Cloud



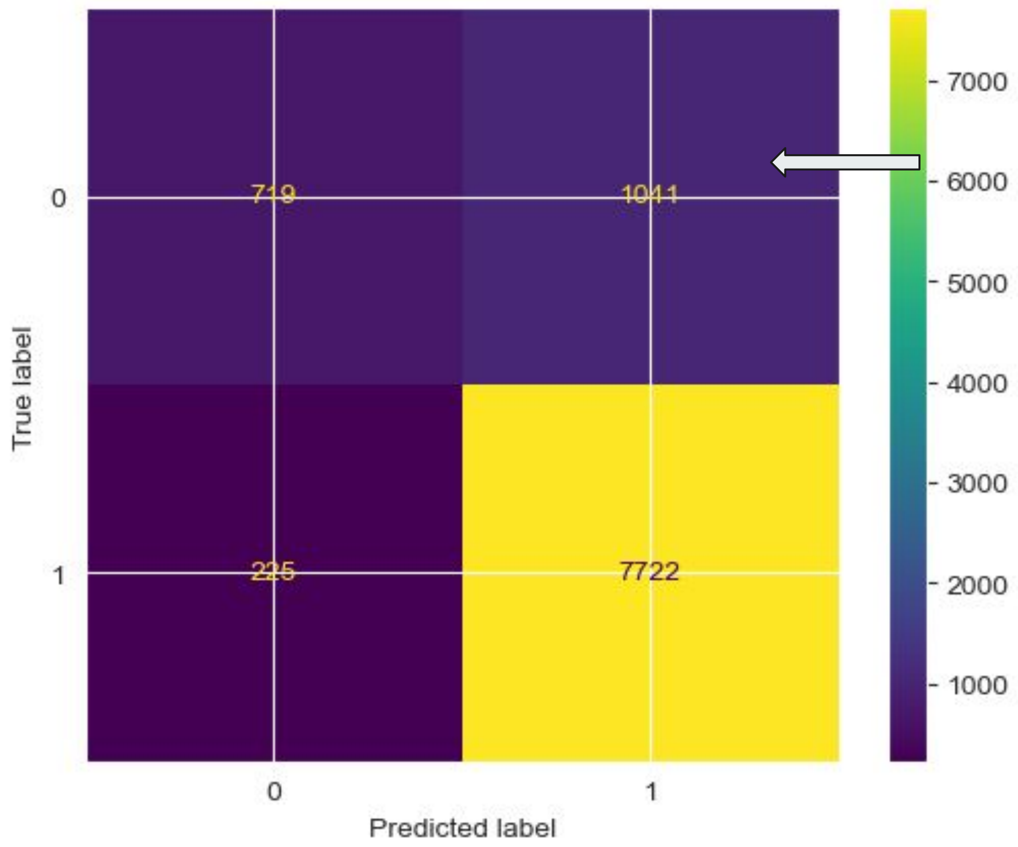
```
# Show word cloud of what words were most used in
pos_review_word = df[df['Label'] == "Positive"]

positive_reviews = pos_review_word['Cleaned_Tweets'].to_string(index=False)

positive_word_cloud = WordCloud(width=350, height=350, background_color='black', max_words=50,
                                min_font_size=10).generate(positive_reviews)
plt.imshow(positive_word_cloud)
plt.axis("off")
plt.show()
```

MODEL PERFORMANCE

TEXTBLOB	SKLEARN (LINEAR SVC)	SKLEARN (LOGISTIC REGRESSION)
76.1%	87.1%	86.9%

To calculate accuracy I compared the predicted sentiment from the models to if the user recommended the game or not i.e interpreted as the sentiment (1-Positive, 0-Negative)
I used the accuracy_score function from sklearn to evaluate the performance

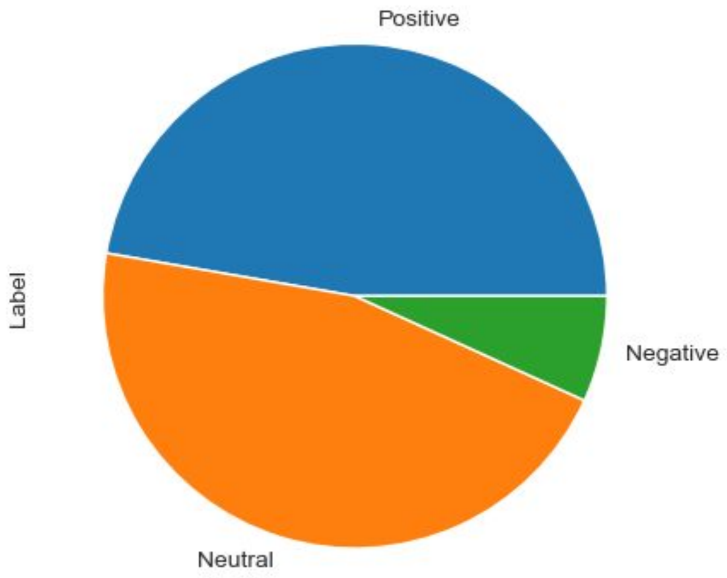
MODEL PERFORMANCE: CONFUSION MATRIX



RESULTS

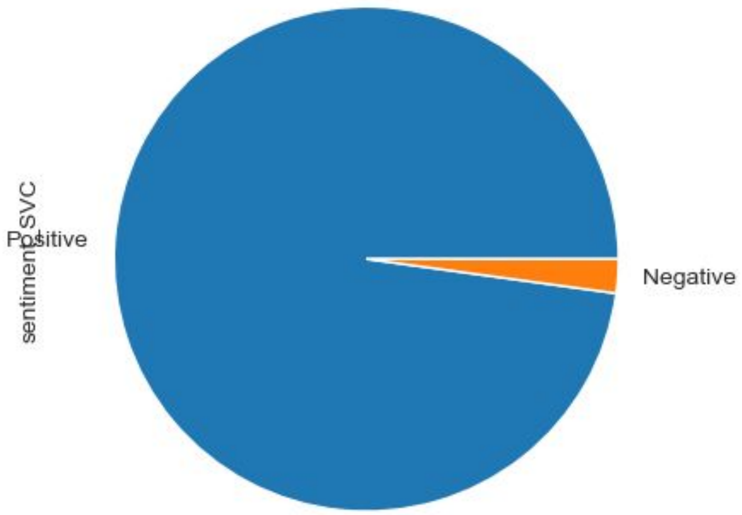
SENTIMENT OF TWEETS

TEXTBLOB



47% Positive, 46% Neutral, 6% Negative

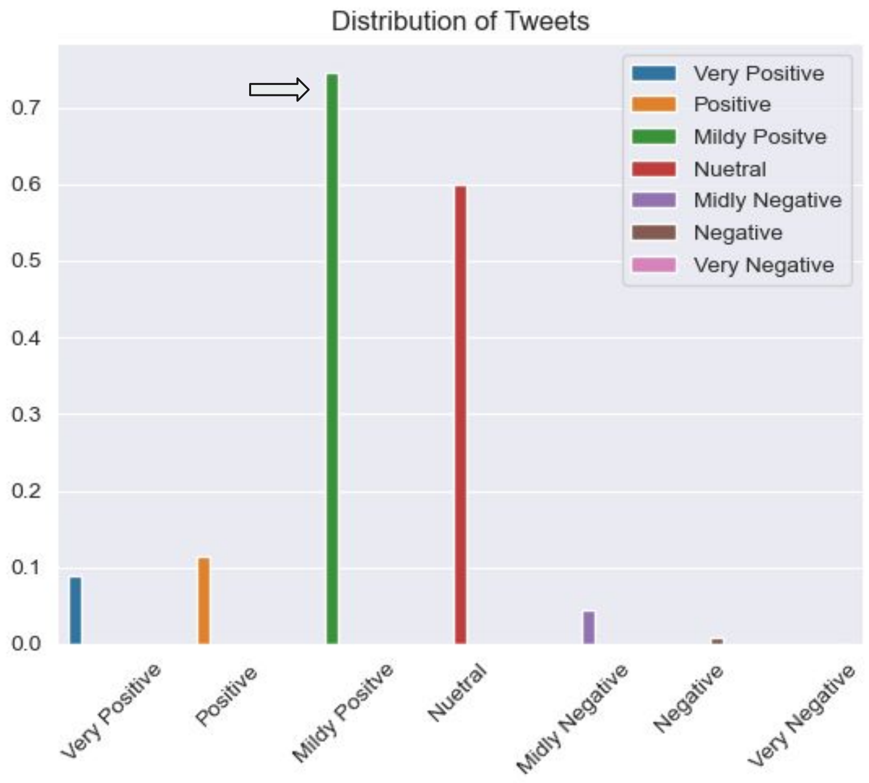
SKLEARN/CUSTOM MODEL



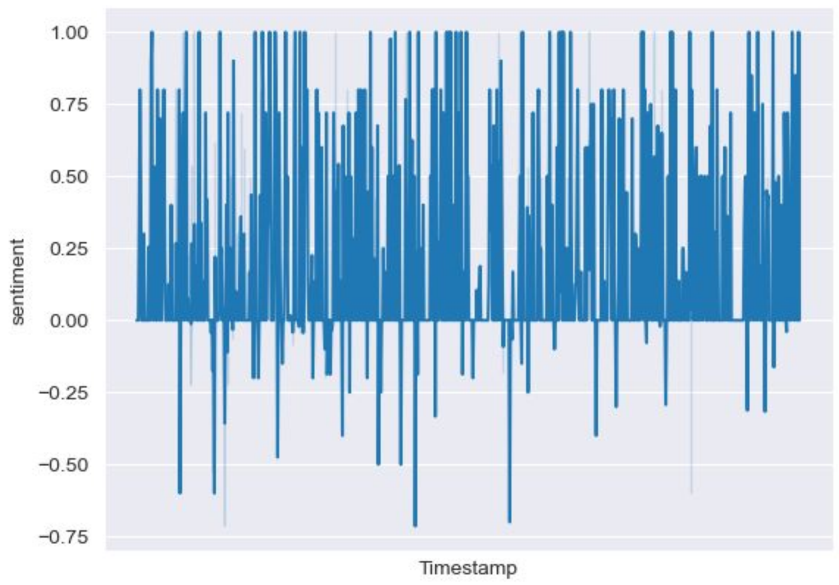
97% Positive, 3% Negative

VARYING ATTITUDES ON STAR WARS JEDI: SURVIVOR

Distribution Of Perception

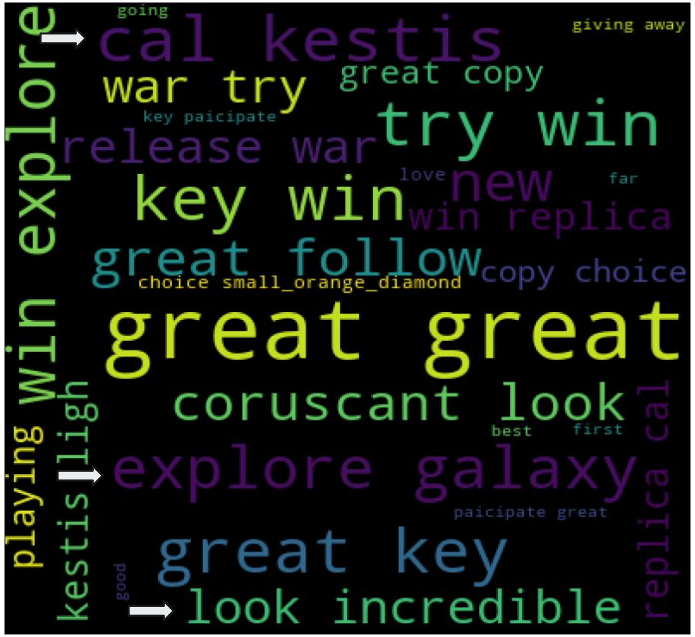


Release Day Perception (Timeline)

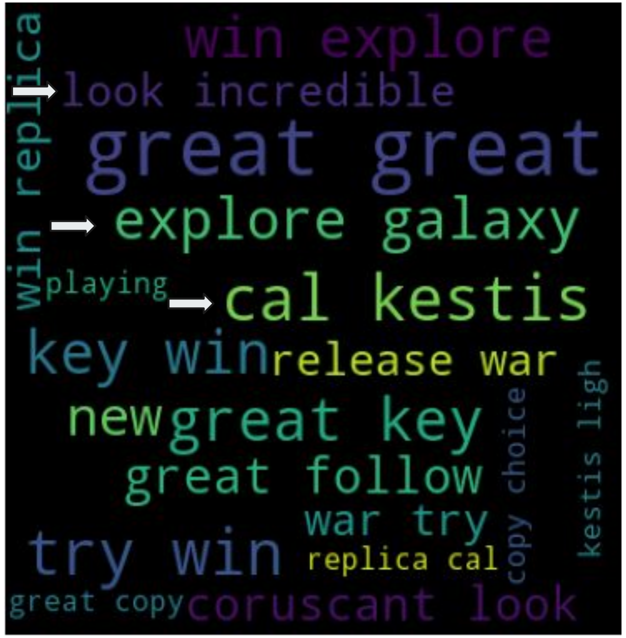


WHAT WERE THE GOOD THINGS ABOUT THE GAME?

TEXTBLOB



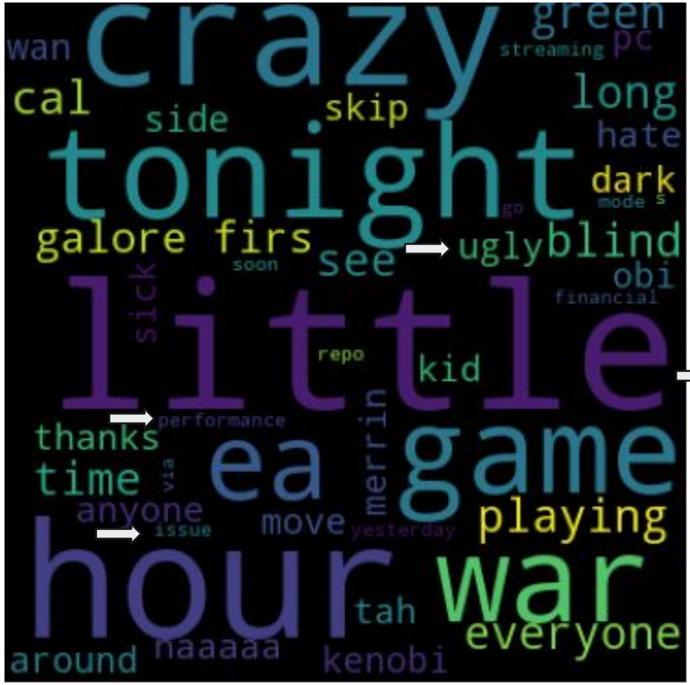
SKLEARN/CUSTOM MODEL



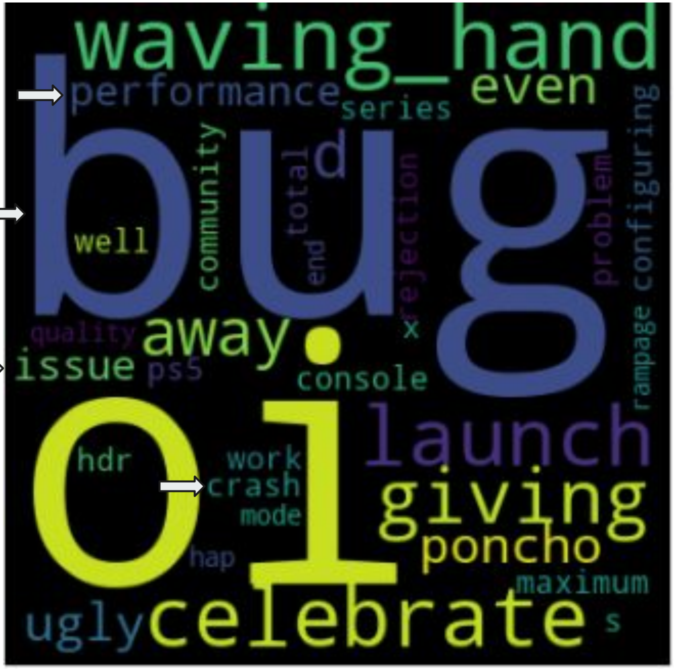
- ➔ Exploration
- ➔ How the game looks
- ➔ Coruscant (a location in the game)
- ➔ Cal Kestis (the main character of the game)

WHAT WERE THE BAD THINGS ABOUT THE GAME?

TEXTBLOB

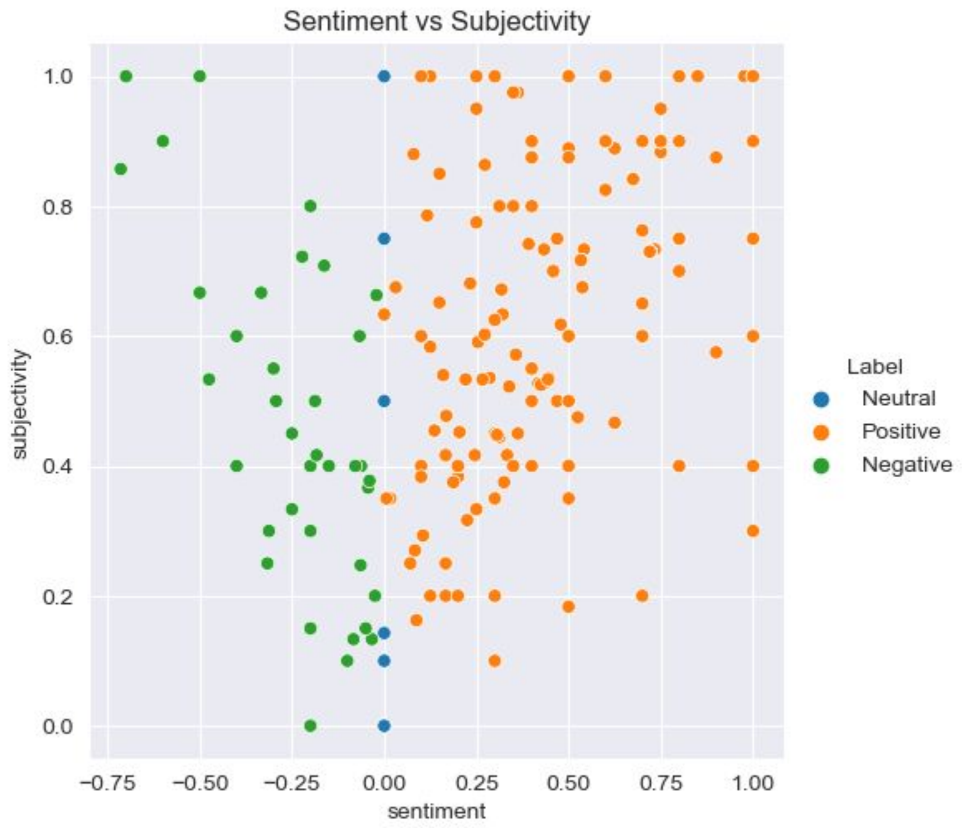


SKLEARN/CUSTOM MODEL



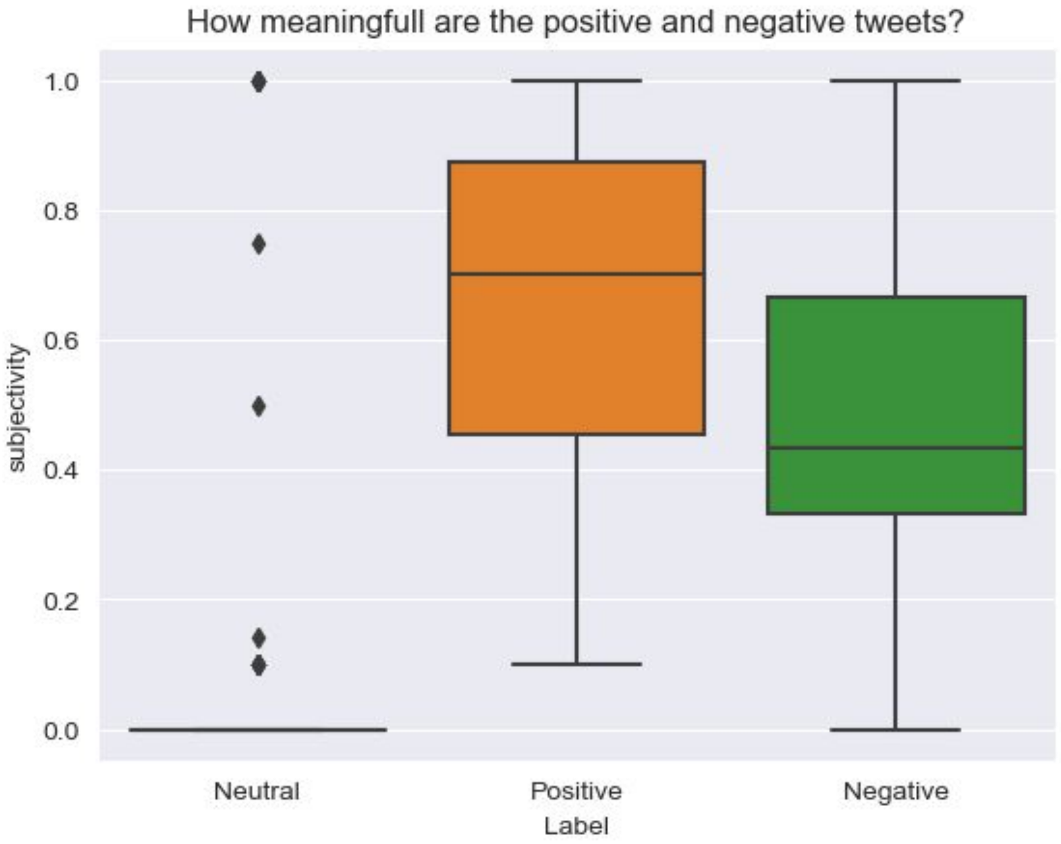
- ➔ Performance
- ➔ Bugs/Crashes
- ➔ FPS

SENTIMENT VS SUBJECTIVITY

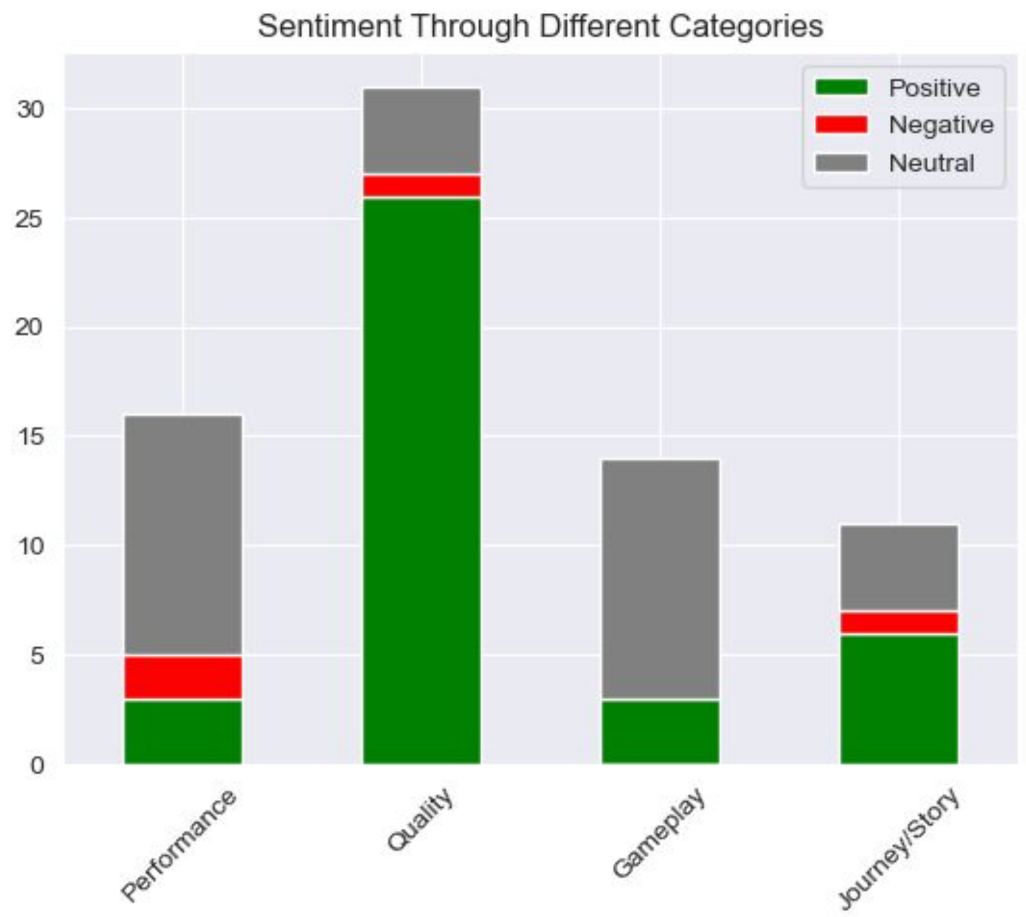


Evidently, as the sentiment increases to high or decreases to high the subjectivity also increases on both sides. This means the more overly positive or negative you get the less **objective** you are. Looking at how subjective or not a tweet is important to keep in mind when doing analysis

HOW BIASED ARE THESE TWEETS?



HOW DIFFERENT AREAS OF THE GAME IS PERCEIVED?



CHALLENGES

- ➔ Sentiment Analysis was a whole new concept that was pretty unknown to me before starting this project
- ➔ **Solution:** Spent time watching videos, looking at other people’s implementation of sentiment analysis, and looking through scikit learn documentation

- ➔ After predicting my sentiments, I had trouble in trying to visualize and show key plots that would best represent my results
- ➔ **Solution:** I looked at common plots and visualization that is used when performing sentiment analysis i.e sentiment over a period of time, word clouds, bar plots, etc.

CONCLUSION

- ➔ Star Wars Jedi: Survivor was perceived decently well by the audience at least around release day(only 3-6% negative tweets on the game)
- ➔ Good things about the game: exploration, look of the game, characters/locations
- ➔ Bad things: Performance, Crashes/Bugs
- ➔ This type of sentiment analysis project is important because they give businesses/companies insights on what they are doing well in terms of their product and what needs to improve
- ➔ A limitation of this analysis is that normally a lot of tweets around release day are very biased towards positive reactions/sentiments because of many promotional and ad campaigns tweets. That is something to keep in mind.
- ➔ **Next Steps:**
  - ◆ In a couple months, look at the rolling mean of the sentiment of the game to see how the sentiment changed or stayed the same over time
  - ◆ This analysis was a more generalized analysis, look deeper at the subjectivity of the tweets to analyze non-bias and un-opinionated tweets as a better metric
  - ◆ Use more complex machine learning models to improve accuracy. Models that are able to look at relationships between words, ex: being able to detect sarcasm

THANK YOU!