For part 2, we searched through many publicly available datasets online that included hate speech and abusive language. We chose to use the dataset from https://github.com/t-davidson/hate-speech-and-offensive-language which is an extension of the paper “Automated Hate Speech Detection and the Problem of Offensive Language.” We chose this dataset since it is substantially large and thoroughly labeled. This dataset includes 27k labeled tweets, each with a classification label for the categories: “hate_speech”, “offensive_language”, “neither”. This dataset is particularly useful for our case since the abusive tweets are in the context of social media, which will likely be a good representation of the messages that we may see in Discord channels.

We first tried running our dataset on a publicly available GitHub repository called “HateSonar.” When given a text string, it outputs the confidence scores for three classes: hate speech, abusive speech, or neither. We have written the code to parse this output and extract the class prediction, accompanied by its confidence score. Out customized output for a few examples from our dataset can be seen below:

```
text: !!! RT @mayasolovely: As a woman you shouldn't complain about cleaning up your house. &amp; as a man y
ou should always take the trash out...
class: neither
score: 0.4860948118990985
text: !!!!! RT @mleew17: boy dats cold...tyga dwn bad for cuffin dat hoe in the 1st place!!
class: offensive_language
score: 0.8864990519327637
text: !!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby4life: You ever fuck a bitch and she start to cry? You b
e confused as shit
class: offensive_language
score: 0.9640148354615952
```

When comparing the outputs of the HateSonar model to the actual labels in the dataset, we observed that the HateSonar model was quite good at properly predicting the class that the tweets belonged. However, when closely examining many of the outputs for the examples in our dataset, we found that the model would often simply be looking for curse words in the tweets to be able to classify the tweet as offensive or hate speech. Therefore, when it would come across a tweet that was offensive, but didn’t include any curse words, it would often mistakenly classify it as “neither.”

We also tried two publicly available APIs: Hatebase and Tisane. Running Tisane on our dataset examples provide the following output:


```
b'{"text":"!!! RT @mayasolovely: As a woman you shouldn\'t complain about cleaning up your house. &amp; as a
 man you should always take the trash out...","entities_summary":[{"type":["person","identifier","username"]
,"name":"@mayasolovely","mentions":[{"sentence_index":1,"offset":3,"length":13}]}]}'
____________
b'{"text":"!!!!! RT @mleew17: boy dats cold...tyga dwn bad for cuffin dat hoe in the 1st place!!","sentiment
_expressions":[{"sentence_index":1,"offset":17,"length":9,"polarity":"negative"},{"sentence_index":2,"offset
":0,"length":12,"polarity":"negative"}],"entities_summary":[{"type":["person","identifier","username"],"name
":"@mleew17","mentions":[{"sentence_index":1,"offset":3,"length":8}]},{"type":"person","name":"tyga dwn","me
ntions":[{"sentence_index":2,"offset":0,"length":8}]},{"type":"numeric","name":"1st","mentions":[{"sentence_
index":2,"offset":39,"length":3}],"subtype":"ordinal"}]}'
```


As we can observe, the output analyzing the text provides much more information than the HateSonar analysis. Based on the examples we saw, the API extracts any names it sees throughout the string. This is extremely useful for the strategies we want to implement: for instance, if a comment is tagged as hateful or negative, we can assume that there’s a good chance that the names mentioned in the text could be the target of potential abuse. Hence, we

could try to automatically mark the potential offender's posts and messages as spam for users whose names have been mentioned.
Running Hatebase on Postman on an example of hate speech from our dataset gives the following output:


```
"result": [
    {
        "vocabulary_id": "HqMnJp4z8",
        "term": "Nuer wew",
        "hateful_meaning": "A Nuer person who is perceived as having sold out their tribe to serve the South
            Sudanese government for financial gain (therefore being seen as aligned with the Dinka tribe); literal
            translation is \"Nuer of money\"",
        "nonhateful_meaning": "",
        "average_offensiveness": 58,
        "language": "nus",
        "plural_of": null,
        "variant_of": null,
        "transliteration_of": null,
        "is_about_nationality": false,
        "is_about_ethnicity": true,
        "is_about_religion": false,
        "is_about_gender": false,
        "is_about_sexual_orientation": false,
        "is_about_disability": false,
        "is_about_class": false,
        "number_of_sightings": 0,
        "number_of_sightings_this_year": 0,
        "number_of_sightings_this_month": 0,
```


The Hatebase API automatically detects and accurately analyzes text of different languages. It not only ranks the offensiveness of the inputted text, but also categorizes the type of abuse (e.g. gender, religion, disability, etc.) which provides us with extremely useful context about the nature of comments.

Manually analyzing the performance of the 3 sentiment analyzers we experimented with, we observe that Hatebase has the most accurate predictions. Tisane and HateSonar both often incorrectly classified speech with swear words that were potentially offensive but did not qualify as hate speech as being overly abusive/hateful. However, Hatebase rarely had this issue in the examples we observed.

Furthermore, the in-depth analysis of the type of abuse identified by Hatebase is incredibly useful for tracking patterns in hate speech for groups and users which allows us to identify and block content from toxic users and channels. We can implement this pattern identification by running our Hatebase analyzer on the historical log of content in our Discord channels and extracting the number of offences in each category and the average severity of such offences.

Hatebase's analysis also allows us to create solutions that preemptively protect our users from popular forms of abuse. For example, after identifying the key types of abuse, we can have our bot create pop-up messages to users about what constitutes abuse and how they can report any instances they encounter.