IS567 - Text Mining
Assignment 2

Task 1: Load the dataset

Task 1: Loading the dataset

```
chatgpt_train = pd.read_csv("chatgpt_train.csv")
chatgpt_train.head()
```

	date	title	review	rating
0	5/21/2023 16:42	Much more accessible for blind users than the web version	Up to this point I?€?ve mostly been using ChatGPT on my windows desktop using Google Chrome. While it?€?s doable, screen reader navigation is pretty difficult on the desktop site and you really have to be an advanced user to find your way through it. I have submitted numerous feedbacks to open AI about this but nothing has changed on that front.\nWell, the good news ?€? the iOS app pretty much addresses all of those problems. The UI seems really clean, uncluttered and designed well to be compatible with voiceover, the screen reader built into iOS. I applaud the inclusivity of this design ?€? I only wish they would give the same attention and care to the accessibility experience of the desktop app.\nI would have given this review five stars but I have just a couple minor quibbles. First, once I submit my prompt, voiceover starts to read aloud ChatGPT?€?s response before that response is finished, so I will hear the first few words of the response followed by voiceover reading aloud the ?€?stop generating?€? button, which isn?€?t super helpful. It would be great if you could better coordinate this alert so that it didn?€?t start reading the message until it had been fully generated. The other thing I?€?d like is a Feedback button easily accessible from within the main screen of the app, to make it as easy as possible to get continuing suggestions and feedback from your users.\nOtherwise, fantastic app so far!	4
1	7/11/2023 12:24	Much anticipated, wasn?€?t let down.	I?€?ve been a user since it?€?s initial roll out and have been waiting for a mobile application ever since using the web app. For reference I?€?m a software engineering student while working in IT full time. I have to say GPT is an crucial tool. It takes far less time to get information quickly that you?€?d otherwise have to source from stack-overflow, various red-hat articles, Ubuntu articles, searching through software documentation, Microsoft documentation ect. Typically chat gpt can find the answer in a fraction of a second that google can. Obviously it is wrong, a lot. But to have the ability to get quick information on my phone like I can in the web browser I?€?m super excited about and have already been using the mobile app since download constantly. And I?€?m excited for the future of this program becoming more accurate and it seems to be getting more and more precise with every roll out. Gone are the days scouring the internet for obscure pieces of information, chat gpt can find it for you with 2 or 3 prompts. I love this app and I?€?m so happy it?€?s on mobile now. The UI is also very sleek, easy to use. My only complaint with the interface is the history tab at the top right. I actually prefer the conversation tabs on the left in the web app but I understand it would make the app kind of clunky especially on mobile since the screen size is smaller. Anyway, awesome app 5 stars.	4
2	5/19/2023 10:16	Almost 5 stars, but?€? no search function	This app would almost be perfect if it wasn?€?t for ONE little thing: a ?€?search in?€? function. As anyone can imagine, these AI chats can get quuuuite long, and quite lengthy. And sometimes I wanna go into a chat & look up a specific part or parts of that particular chat by using a search function to look up key words within that chat & track down whatever part I was looking for. For example, in a chat, if I had searched way early into the chat ?€?how do lions hunt??€? And say days later, I wanted to revisit that particular response, I wanna be able to go into the actual chat go to a ?€?search in?€? function and be able to type in key words like ?€?lions?€? or ?€?hunt?€? to be able to automatically find that part in the chat instead of having to scroll through a massive chat to find that part. Similar to what you can do in Microsoft Word docs, or even on web browsers. I think the app already kind of has this, but it doesn?€?t work exactly like that. I tested it out, and all it does is, you type in key words, and it shows you that those words are present in the chat or chats, but it doesn?€?t actually	4

```
chatgpt_test = pd.read_csv("chatgpt_test.csv")
chatgpt_test.head()
```

	date	title	review	rating
0	5/19/2023 6:09	error unsupported country	cant login	2
1	5/19/2023 9:39	Hype junk	More harm than help.	1
2	5/19/2023 4:12	your gpt 4 is fake	Fix it	1
3	5/20/2023 3:01	Please impose IPadOS	We need IPadOS!!!	5
4	5/19/2023 20:49	Amazing	Great	5

Number of instances in the training dataset with blank reviews

```
len(chatgpt_train)
```

```
1834
```

Number of instances in the test dataset with blank reviews

```
len(chatgpt_test)
```

```
458
```

After loading, we have **1834** instances of training dataset and **458** instances of test dataset.

```
chatgpt_test.dropna(subset = ['review'], inplace=True)
chatgpt_test.head()
```

	date	title	review	rating
0	5/19/2023 6:09	error unsupported country	cant login	2
1	5/19/2023 9:39	Hype junk	More harm than help.	1
2	5/19/2023 4:12	your gpt 4 is fake	Fix it	1
3	5/20/2023 3:01	Please impose IPadOS	We need IPadOS!!!	5
4	5/19/2023 20:49	Amazing	Great	5

Number of instances in the training dataset with blank reviews removed

```
len(chatgpt_train)
```

```
1829
```

Number of instances in the test dataset with blank reviews removed

```
len(chatgpt_test)
```

```
458
```

After removing blank reviews, we have **1829** instances of training dataset and **458** instances of test dataset.

Task 2: POS Tagging

Copy of the training dataset

```
chatgpt_train_copy = chatgpt_train.copy()
```

```
chatgpt_train_copy
```

	date	title	review	rating
0	5/21/2023 16:42	Much more accessible for blind users than the web version	Up to this point I?€?ve mostly been using ChatGPT on my windows desktop using Google Chrome. While it?€?s doable, screen reader navigation is pretty difficult on the desktop site and you really have to be an advanced user to find your way through it. I have submitted numerous feedbacks to open AI about this but nothing has changed on that front.\nWell, the good news ?€? the iOS app pretty much addresses all of those problems. The UI seems really clean, uncluttered and designed well to be compatible with voiceover, the screen reader built into iOS. I applaud the inclusivity of this design ?€? I only wish they would give the same attention and care to the accessibility experience of the desktop app.\nI would have given this review five stars but I have just a couple minor quibbles. First, once I submit my prompt, voiceover starts to read aloud ChatGPT?€?s response before that response is finished, so I will hear the first few words of the response followed by voiceover reading aloud the ?€?stop generating?€? button, which isn?€?t super helpful. It would be great if you could better coordinate this alert so that it didn?€?t start reading the message until it had been fully generated. The other thing I?€?d like is a Feedback button easily accessible from within the main screen of the app, to make it as easy as possible to get continuing suggestions and feedback from your users.\nOtherwise, fantastic app so far!	4
1	7/11/2023 12:24	Much anticipated, wasn?€?t let down.	I?€?ve been a user since it?€?s initial roll out and have been waiting for a mobile application ever since using the web app. For reference I?€?m a software engineering student while working in IT full time. I have to say GPT is an crucial tool. It takes far less time to get information quickly that you?€?d otherwise have to source from stack-overflow, various red-hat articles, Ubuntu articles, searching through software documentation, Microsoft documentation ect. Typically chat gpt can find the answer in a fraction of a second that google can. Obviously it is wrong, a lot. But to have the ability to get quick information on my phone like I can in the web browser I?€?m super excited about and have already been using the mobile app since download constantly. And I?€?m excited for the future of this program becoming more accurate and it seems to be getting more and more precise with every roll out. Gone are the days scouring the internet for obscure pieces of information, chat gpt can find it for you with 2 or 3 prompts. I love this app and I?€?m so happy it?€?s on mobile now. The UI is also very sleek, easy to use. My only complaint with the interface is the history tab at the top right. I actually prefer the conversation tabs on the left in the web app but I understand it would make the app kind of clunky especially on mobile since the screen size is smaller. Anyway, awesome app 5 stars.	4

```
chatgpt_train_copy.loc[:, ('pos_tags')] = chatgpt_train_copy.loc[:, ('tokenized_review')].apply(nltk.pos_tag)
```

```
chatgpt_train_copy[['pos_tags']]
```

	pos_tags
0	[(up, RB), (to, TO), (this, DT), (point, NN), (i, NN), (?, .), (€, NN), (?, .), (ve, NN), (mostly, RB), (been, VBN), (using, VBG), (chatgpt, NN), (on, IN), (my, PRP), (windows, NNS), (desktop, VBP), (using, VBG), (google, NN), (chrome, NN), (. , .), (while, IN), (it, PRP), (?, .), (€, VB), (?, .), (s, NN), (doable, JJ), (, , ,), (screen, JJ), (reader, NN), (navigation, NN), (is, VBZ), (pretty, RB), (difficult, JJ), (on, IN), (the, DT), (desktop, NN), (site, NN), (and, CC), (you, PRP), (really, RB), (have, VB), (to, TO), (be, VB), (an, DT), (advanced, JJ), (user, NN), (to, TO), (find, VB), (your, PRP), (way, NN), (through, IN), (it, PRP), (., .), (i, NNS), (have, VBP), (submitted, VBN), (numerous, JJ), (feedbacks, NNS), (to, TO), (open, VB), (ai, NN), (about, IN), (this, DT), (but, CC), (nothing, NN), (has, VBZ), (changed, VBN), (on, IN), (that, DT), (front, NN), (., .), (well, RB), (,, ,), (the, DT), (good, JJ), (news, NN), (?, .), (€, NN), (?, .), (the, DT), (ios, NNS), (app, VBP), (pretty, RB), (much, JJ), (addresses, VBZ), (all, DT), (of, IN), (those, DT), (problems, NNS), (., .), (the, DT), (ui, JJ), (seems, VBZ), (really, RB), (clean, JJ), (,, ,), (uncluttered, JJ), (and, CC), ...
1	[(i, NN), (?, .), (€, NN), (?, .), (ve, RB), (been, VBN), (a, DT), (user, NN), (since, IN), (it, PRP), (?, .), (€, VB), (?, .), (s, JJ), (initial, JJ), (roll, NN), (out, RB), (and, CC), (have, VBP), (been, VBN), (waiting, VBG), (for, IN), (a, DT), (mobile, JJ), (application, NN), (ever, RB), (since, IN), (using, VBG), (the, DT), (web, NN), (app, NN), (., .), (for, IN), (reference, NN), (i, NN), (?, .), (€, NN), (?, .), (m, VB), (a, DT), (software, NN), (engineering, NN), (student, NN), (while, IN), (working, VBG), (in, IN), (it, PRP), (full, JJ), (time, NN), (., .), (i, NNS), (have, VBP), (to, TO), (say, VB), (gpt, NN), (is, VBZ), (an, DT), (crucial, JJ), (tool, NN), (., .), (it, PRP), (takes, VBZ), (far, RB), (less, JJR), (time, NN), (to, TO), (get, VB), (information, NN), (quickly, RB), (that, IN), (you, PRP), (?, .), (€, VB), (?, .), (d, JJ), (otherwise, RB), (have, VBP), (to, TO), (source, NN), (from, IN), (stack-overflow, JJ), (,, ,), (various, JJ), (red-hat, JJ), (articles, NNS), (,, ,), (ubuntu, JJ), (articles, NNS), (,, ,), (searching, VBG), (through, IN), (software, NN), (documentation, NN), (,, ,), (microsoft, JJ), (documentation, NN), (ect, NN), (., .), (typically, RB), (chat, WP), ...
2	[(this, DT), (app, NN), (would, MD), (almost, RB), (be, VB), (perfect, JJ), (if, IN), (it, PRP), (wasn, VBZ), (?, .), (€, VB), (?, .), (t, NN), (for, IN), (one, CD), (little, JJ), (thing, NN), (:, :), (a, DT), (?, .), (€, NN), (?, .), (search, NN), (in, IN), (?, .), (€, NN), (?, .), (function, NN), (., .), (as, IN), (anyone, NN), (can, MD), (imagine, VB), (,, ,), (these, DT), (ai, VBP), (chats, NNS), (can, MD), (get, VB), (quuuuite, RB), (long, RB), (,, ,), (and, CC), (quite, RB), (lengthy, JJ), (., .), (and, CC), (sometimes, RB), (i, JJ), (wan, VBP), (na, TO), (go, VB), (into, IN), (a, DT), (chat, NN), (&, CC), (look, VB), (up, RP), (a, DT), (specific, JJ), (part, NN), (or, CC), (parts, NNS), (of, IN), (that, DT), (particular, JJ), (chat, NN), (by, IN), (using, VBG), (a, DT), (search, NN), (function, NN), (to, TO), (look, VB), (up, RP), (key, JJ), (words, NNS), (within, IN), (that, DT), (chat, NN), (&, CC), (track, VB), (down, RP), (whatever, WDT), (part, NN), (i, NN), (was, VBD), (looking, VBG), (for, IN), (., .), (for, IN), (example, NN), (,, ,), (in, IN), (a, DT), (chat, NN), (,, ,), (if, IN), (i, VBN), (had, VBD), ...

3 examples of review and POS Tagging

```
chatgpt_train_copy.loc[35:35, ('review', 'pos_tags')]
```

	review	pos_tags
35	With the exceptional performance seen on the browser of desktop, I was unsure as to UI on mobile. I can say that it?€?s even more intuitive while trying to navigate the now native app. \n\nThe haptic feedback is especially unique and engaging with responses while being generated. \n\nI look forward to future improvements and abilities that ChatGPT will be able to integrate into beyond the sandbox. I do hope that once certain barriers are met conversations can move past the guard rails. \n\nImprovements; Sharing threads you create with colleagues outside of screenshots would be ideal for broadening collaboration. Hypothetical War-games, strategy and the like while having a conversation in a group thread on Game Theory would be ideal. \n\nWith the ability of inputting other app?€?s Terms of Services (Tik-Tok May 19, 2023) for example; prior to agreeing to them, the ability to compare and contrast with the previous terms would be helpful to cut out the mindless jargon. ChatGPT already has this ability, now that it?€?s native to iOS. \n\nHaving it the past several days, I?€?ve already seen an update, with some refreshed improvements. If this?€? OpenAi?€?s platform aperitif, I very much look forward to the coming entr??e.	[(with, IN), (the, DT), (exceptional, JJ), (performance, NN), (seen, VBN), (on, IN), (the, DT), (browser, NN), (of, IN), (desktop, NN), (,, ,), (i, NN), (was, VBD), (unsure, JJ), (as, IN), (to, TO), (ui, VB), (on, IN), (mobile, NN), (., .), (i, NN), (can, MD), (say, VB), (that, IN), (it, PRP), (?, .), (€, VB), (?, .), (s, VB), (even, RB), (more, RBR), (intuitive, JJ), (while, IN), (trying, VBG), (to, TO), (navigate, VB), (the, DT), (now, RB), (native, JJ), (app, NN), (., .), (the, DT), (haptic, JJ), (feedback, NN), (is, VBZ), (especially, RB), (unique, JJ), (and, CC), (engaging, VBG), (with, IN), (responses, NNS), (while, IN), (being, VBG), (generated, VBN), (., .), (i, JJ), (look, VBP), (forward, RB), (to, TO), (future, JJ), (improvements, NNS), (and, CC), (abilities, NNS), (that, IN), (chatgpt, NN), (will, MD), (be, VB), (able, JJ), (to, TO), (integrate, VB), (into, IN), (beyond, IN), (the, DT), (sandbox, NN), (., .), (i, VB), (do, VBP), (hope, VB), (that, DT), (once, RB), (certain, JJ), (barriers, NNS), (are, VBP), (met, VBN), (conversations, NNS), (can, MD), (move, VB), (past, IN), (the, DT), (guard, NN), (rails, VBZ), (., .), (improvements, NNS), (;, :), (sharing, VBG), (threads, NNS), (you, PRP), (create, VBP), (with, IN), (colleagues, NNS), ...

```
chatgpt_train_copy.loc[100:100, ('review', 'pos_tags')]
```

	review	pos_tags
100	Simple and not complicated at all user interface. Easy to navigate around and overall a good user experience. However, there is no option to generate a different response to the same prompt and to edit a prompt I wrote like how you would in the web version. In my opinion, I would still use the web version due to those features. The mobile version would be better for quicker access and response.	[(simple, NN), (and, CC), (not, RB), (complicated, VBN), (at, IN), (all, DT), (user, JJ), (interface, NN), (., .), (easy, JJ), (to, TO), (navigate, VB), (around, IN), (and, CC), (overall, VB), (a, DT), (good, JJ), (user, NN), (experience, NN), (., .), (however, RB), (,, ,), (there, EX), (is, VBZ), (no, DT), (option, NN), (to, TO), (generate, VB), (a, DT), (different, JJ), (response, NN), (to, TO), (the, DT), (same, JJ), (prompt, NN), (and, CC), (to, TO), (edit, VB), (a, DT), (prompt, NN), (i, JJ), (wrote, VBD), (like, IN), (how, WRB), (you, PRP), (would, MD), (in, IN), (the, DT), (web, JJ), (version, NN), (., .), (in, IN), (my, PRP$), (opinion, NN), (,, ,), (i, NN), (would, MD), (still, RB), (use, VB), (the, DT), (web, NN), (version, NN), (due, JJ), (to, TO), (those, DT), (features, NNS), (., .), (the, DT), (mobile, JJ), (version, NN), (would, MD), (be, VB), (better, JJR), (for, IN), (quicker, JJR), (access, NN), (and, CC), (response, NN), (., .)]

```
chatgpt_train_copy.loc[1500:1500, ('review', 'pos_tags')]
```

	review	pos_tags
1500	The ?€?very simplistic?€? ui hits a sweet spot for me ????	[(the, DT), (?, .), (€, NN), (?, .), (very, RB), (simplistic, JJ), (?, .), (€, NN), (?, .), (ui, JJ), (hits, VBZ), (a, DT), (sweet, JJ), (spot, NN), (for, IN), (me, PRP), (?, .), (?, .), (?, .), (?, .)]

Q) What parts of speech could be useful for sentiment analysis? Why?

Ans) First, we need to identify words that carry the sentiment:

- Adjectives are the words describing nouns and express sentiments. Words like ‘happy’, ‘exceptional’, ‘good’ etc show positive sentiment and ‘terrible’, ‘sad’, etc show negative sentiment.

- Adverbs modify the verbs and say more about them emphasizing the sentiment quantity. They add more context. For eg. ‘better’ or ‘more’ expresses more positive sentiment than good.

- Conjunctions or transition words can help us understand the tone and purpose of the statements apart from the words. They include words like ‘and’ expresses a continuation of the emotion whereas ‘but’ indicates a contradiction. For eg. ‘this phone is pretty and inexpensive’ is a positive sentiment whereas ‘this phone is pretty but expensive’ is a contradiction.

- Interjections and words like ‘wow’, ‘ouch’ express a strong positive or negative sentiment.

- Negation words like ‘not’ convey the complete opposite meaning of the actual word and express a contradiction to the existing context.

Q) Errors in the POS Tagging

Ans) In an example POS tagged review:

- In the review 100, the word ‘simple’ has been assigned the category of a Noun. But there is a possibility that it may be describing the noun and it can act as an Adjective as well.

- In the review 100, the word ‘complicated’ has been assigned the category of a VBD (form of a verb) which sounds correct. But based on the contect it is describing the noun and it can act as an Adjective as well.

- ‘No’ can have the properties of ‘Adverb’, ‘Adjective’ or ‘Noun’.

- In the review 1500, the word ‘ui’ has been considered as an adjective when it actually is a noun referring to the UI interface of a browser.

Task 3: Extract unigram features

Task 3: Extract unigram features

Fit training dataset

```
# range is (1,1) for extracting unigrams
vectorizer = CountVectorizer(ngram_range = (1,1))

# tokenize and build vocabulary
vectorizer.fit(chatgpt_train['review'])
print(vectorizer.vocabulary_, '\n')

# create feature vector representation
vector = vectorizer.transform(chatgpt_train['review'])

print(vector.shape,"\n")

# complete vectors
print(vector.toarray())
```

```
{'up': 5224, 'to': 4985, 'this': 4932, 'point': 3613, 've': 5282, 'mostly': 3134, 'been': 493, 'using': 5253, 'chat
gpt': 776, 'on': 3331, 'my': 3165, 'windows': 5454, 'desktop': 1302, 'google': 2139, 'chrome': 806, 'while': 5421,
'it': 2666, 'doable': 1426, 'screen': 4267, 'reader': 3883, 'navigation': 3184, 'is': 2660, 'pretty': 3692, 'diffic
ult': 1352, 'the': 4906, 'site': 4443, 'and': 256, 'you': 5540, 'really': 3891, 'have': 2248, 'be': 474, 'an': 250,
'advanced': 151, 'user': 5250, 'find': 1907, 'your': 5542, 'way': 5382, 'through': 4954, 'submitted': 4698, 'numero
us': 3277, 'feedbacks': 1877, 'open': 3341, 'ai': 197, 'about': 60, 'but': 667, 'nothing': 3255, 'has': 2241, 'chan
ged': 754, 'that': 4903, 'front': 2032, 'well': 5403, 'good': 2135, 'news': 3221, 'ios': 2640, 'app': 291, 'much':
3147, 'addresses': 137, 'all': 213, 'of': 3303, 'those': 4937, 'problems': 3720, 'ui': 5130, 'seems': 4306, 'clea
n': 821, 'uncluttered': 5148, 'designed': 1295, 'compatible': 907, 'with': 5464, 'voiceover': 5336, 'built': 657,
'into': 2619, 'applaud': 301, 'inclusivity': 2470, 'design': 1294, 'only': 3337, 'wish': 5460, 'they': 4922, 'woul
d': 5503, 'give': 2109, 'same': 4217, 'attention': 392, 'care': 704, 'accessibility': 81, 'experience': 1775, 'give
n': 2111, 'review': 4126, 'five': 1928, 'stars': 4612, 'just': 2698, 'couple': 1081, 'minor': 3081, 'quibbles': 384
3, 'first': 1923, 'once': 3332, 'submit': 4697, 'prompt': 3757, 'starts': 4618, 'read': 3882, 'aloud': 222, 'respon
se': 4089, 'before': 497, 'finished': 1917, 'so': 4489, 'will': 5449, 'hear': 2261, 'few': 1889, 'words': 5485, 'fo
```

```
# use review for model
train_text = chatgpt_train["review"]
test_text = chatgpt_test["review"]

# set the n-gram range
vectorizer = CountVectorizer(ngram_range = (1,1))

# create training data representation
train_data_repr = vectorizer.fit_transform(train_text.values.astype('U'))
print("NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS")
print(train_data_repr.shape,"\n")

# create test data representation
test_data_repr = vectorizer.transform(test_text.values.astype('U'))
print("NUMBER OF FEATURES IN TEST DATASET WITH UNIGRAMS")
print(test_data_repr.shape,"\n")
```

```
NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS
(1829, 5551)

NUMBER OF FEATURES IN TEST DATASET WITH UNIGRAMS
(458, 5551)
```

Task 4: Train and evaluate classifier

Task 4: Training and evaluate Naive Bayes classifier with unigram features

```
# define true labels from train set
X_train = train_data_repr
y_train = chatgpt_train["rating"]
X_test = test_data_repr
y_test = chatgpt_test["rating"]
```

```
# Naive Bayes model
model_unigram = MultinomialNB()
model_unigram.fit(X_train, y_train)
```

```
MultinomialNB()
```

```
# predict the labels for the test data
predictions = model_unigram.predict(X_test)
predictions
```

```
array([1, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 3, 1, 1, 5, 5, 5, 1, 1, 5, 5, 5,
       1, 5, 5, 1, 5, 5, 1, 5, 4, 5, 5, 5, 1, 5, 5, 5, 5, 5, 1, 3, 5, 5,
       5, 1, 5, 5, 5, 5, 5, 5, 1, 5, 1, 5, 3, 4, 5, 5, 1, 5, 5, 1, 1, 5,
       5, 5, 5, 1, 5, 4, 5, 1, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 1, 3, 5, 5,
       5, 5, 1, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5,
       5, 5, 5, 5, 5, 1, 5, 5, 5, 1, 1, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5,
       1, 1, 1, 5, 1, 1, 5, 5, 5, 5, 5, 2, 2, 5, 3, 1, 5, 5, 5, 1, 5, 5,
       5, 5, 1, 3, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 2, 1, 1, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5,
       5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, 3, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 2, 5, 5, 1, 2, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 1, 3, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 3, 5, 5, 5, 5,
```

Saving the model

```
import pickle

filename_1 = 'NaiveBayesModel_unigram.sav'
pickle.dump(model_unigram, open(filename_1, 'wb'))
```

Load the model

```
loaded_model_unigram = pickle.load(open(filename_1, 'rb'))
result = loaded_model_unigram.score(X_test, y_test)
print(result)
```

```
0.6331877729257642
```

Evaluating the unigram model using metrics

```
print ("Overall Accuracy score: ", accuracy_score(y_test, predictions))
print ("Overall Recall score: ", recall_score(y_test, predictions, average='macro'))
print ("Overall Precision score: ", precision_score(y_test, predictions, average='macro'))
print ("Overall F1 score: ", f1_score(y_test, predictions, average='macro'), "\n")
print ("Individual label performance: ")
print (classification_report(y_test, predictions))
print ("Confusion Matrix: ")
print (confusion_matrix(y_test, predictions))
```

```
Overall Accuracy score:  0.6331877729257642
Overall Recall score:  0.34043281122804264
Overall Precision score:  0.4365165675446049
Overall F1 score:  0.35240689572225525

Individual label performance:
              precision    recall  f1-score   support

           1       0.57      0.49      0.52       115
           2       0.40      0.08      0.13        26
           3       0.13      0.08      0.10        25
           4       0.39      0.16      0.23        44
           5       0.69      0.90      0.78       248

    accuracy                           0.63       458
   macro avg       0.44      0.34      0.35       458
weighted avg       0.59      0.63      0.59       458

Confusion Matrix:
[[ 56   0   6   4  49]
 [ 13   2   1   1   9]
 [  8   0   2   2  13]
 [  6   0   4   7  27]
 [ 16   3   2   4 223]]
```

Task 5: Add bigram features

Task 5: Add bigram features

```
# range is (1,2) for extracting unigrams and bigrams
vectorizer = CountVectorizer(ngram_range = (1,2))

# tokenize and build vocabulary
vectorizer.fit(chatgpt_train['review'])
print(vectorizer.vocabulary_, '\n')

# create feature vector representation
vector = vectorizer.transform(chatgpt_train['review'])

print(vector.shape,"\n")

# complete vectors
print(vector.toarray())
```

```
{'up': 37347, 'to': 35638, 'this': 35036, 'point': 26826, 've': 38075, 'mostly': 22562, 'been': 5228, 'using': 3787
6, 'chatgpt': 7119, 'on': 24675, 'my': 22722, 'windows': 39767, 'desktop': 9651, 'google': 14656, 'chrome': 7449,
'while': 39475, 'it': 18831, 'doable': 10297, 'screen': 29943, 'reader': 28240, 'navigation': 23095, 'is': 18313,
'pretty': 27163, 'difficult': 10007, 'the': 33738, 'site': 31021, 'and': 2014, 'you': 40659, 'really': 28300, 'hav
e': 15450, 'be': 4887, 'an': 1842, 'advanced': 888, 'user': 37745, 'find': 12881, 'your': 40851, 'way': 38846, 'thr
ough': 35416, 'submitted': 32364, 'numerous': 23978, 'feedbacks': 12668, 'open': 25051, 'ai': 1105, 'about': 245,
'but': 6177, 'nothing': 23775, 'has': 15322, 'changed': 6939, 'that': 33429, 'front': 13871, 'well': 39085, 'good':
14581, 'news': 23360, 'ios': 18072, 'app': 3207, 'much': 22599, 'addresses': 835, 'all': 1313, 'of': 24053, 'thos
e': 35291, 'problems': 27335, 'ui': 36999, 'seems': 30277, 'clean': 7481, 'uncluttered': 37074, 'designed': 9622,
'compatible': 7863, 'with': 39808, 'voiceover': 38509, 'built': 6146, 'into': 17933, 'applaud': 3641, 'inclusivit
y': 17156, 'design': 9596, 'only': 24947, 'wish': 39783, 'they': 34842, 'would': 40362, 'give': 14402, 'same': 2969
6, 'attention': 4519, 'care': 6749, 'accessibility': 423, 'experience': 12045, 'given': 14429, 'review': 29330, 'fi
ve': 13029, 'stars': 31907, 'just': 19631, 'couple': 8745, 'minor': 22042, 'quibbles': 27990, 'first': 12972, 'onc
e': 24831, 'submit': 32358, 'prompt': 27504, 'starts': 31973, 'read': 28223, 'aloud': 1467, 'response': 29107, 'bef
ore': 5318, 'finished': 12958, 'so': 31214, 'will': 39669, 'hear': 15691, 'few': 12753, 'words': 40165, 'followed':
13182, 'by': 6364, 'reading': 28247, 'stop': 32107, 'generating': 14216, 'button': 6333, 'which': 39425, 'isn': 187
53, 'super': 32564, 'helpful': 15794, 'great': 14912, 'if': 16454, 'could': 8657, 'better': 5515, 'coordinate': 858
2, 'alert': 1274, 'didn': 9915, 'start': 31928, 'message': 21890, 'until': 37315, 'had': 15106, 'fully': 13925, 'ge
nerated': 14191, 'other': 25542, 'thing': 34920, 'like': 20546, 'feedback': 12623, 'easily': 10780, 'accessible': 4
```

Number of features in the dataset using unigrams and bigrams

```
# use review for model building

train_text = chatgpt_train["review"]
test_text = chatgpt_test["review"]

# set the n-gram range
vectorizer = CountVectorizer(ngram_range = (1,2))

# create training data representation
train_data_repr = vectorizer.fit_transform(train_text.values.astype('U'))
print("NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS, BIGRAMS")
print(train_data_repr.shape,"\n")

# create test data representation
test_data_repr = vectorizer.transform(test_text.values.astype('U'))
print("NUMBER OF FEATURES IN TEST DATASET WITH UNIGRAMS, BIGRAMS")
print(test_data_repr.shape,"\n")
```

```
NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS, BIGRAMS
(1829, 41006)

NUMBER OF FEATURES IN TEST DATASET WITH UNIGRAMS, BIGRAMS
(458, 41006)
```

Train Naive Bayes with unigram and bigram features

```
from sklearn.naive_bayes import MultinomialNB

# define true labels from train set
X_train = train_data_repr
y_train = chatgpt_train["rating"]
X_test = test_data_repr
y_test = chatgpt_test["rating"]
```

```
# Naive Bayes model
model_unigram_bigram = MultinomialNB()
model_unigram_bigram.fit(X_train, y_train)
```

```
MultinomialNB()
```

```
# predict the labels for the test data
predictions = model_unigram_bigram.predict(X_test)
predictions
```

```
array([1, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 3, 1, 1, 4, 5, 5, 5, 1, 5, 5, 5,
       1, 5, 5, 1, 5, 5, 1, 5, 5, 5, 5, 5, 1, 5, 1, 5, 5, 5, 1, 5, 5, 5,
       5, 1, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 3, 4, 5, 5, 1, 5, 5, 4, 1, 5,
       5, 5, 5, 1, 5, 4, 5, 5, 5, 4, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5,
       5, 5, 5, 5, 5, 1, 5, 5, 5, 1, 1, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5,
       1, 5, 1, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, 4, 5, 5, 5, 5, 5, 5,
       5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 4, 1, 1, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5,
       5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 4, 5, 5, 1, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 1, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5,
```

Saving the unigram + bigram model

```
import pickle

filename_2 = 'NaiveBayesModel_unigram_bigram.sav'
pickle.dump(model_unigram_bigram, open(filename_2, 'wb'))
```

Loading the unigram + bigram model

```
loaded_model_unigram_bigram = pickle.load(open(filename_2, 'rb'))
result = loaded_model_unigram_bigram.score(X_test, y_test)
print(result)
```

```
0.6244541484716157
```

Evaluating unigram + bigram model using metrics

```
print ("Accuracy score: ", accuracy_score(y_test, predictions))
print ("Overall Recall score: ", recall_score(y_test, predictions, average='macro'))
print ("Overall Precision score: ", precision_score(y_test, predictions, average='macro'))
print ("Overall F1 score: ", f1_score(y_test, predictions, average='macro'), "\n")
print ("Individual label performance: ")
print (classification_report(y_test, predictions), "\n")
print ("Confusion Matrix: ")
print (confusion_matrix(y_test, predictions))
```

```
Accuracy score:  0.6244541484716157
Overall Recall score:  0.28008542649496365
Overall Precision score:  0.27141238135713275
Overall F1 score:  0.26332604636538254

Individual label performance:
              precision    recall  f1-score   support

           1       0.63      0.43      0.51       115
           2       0.00      0.00      0.00        26
           3       0.00      0.00      0.00        25
           4       0.08      0.02      0.04        44
           5       0.65      0.95      0.77       248

    accuracy                           0.62       458
   macro avg       0.27      0.28      0.26       458
weighted avg       0.52      0.62      0.55       458


Confusion Matrix:
[[ 49   0   3   3  60]
 [ 11   0   1   3  11]
 [  7   0   0   0  18]
 [  5   0   1   1  37]
 [  6   0   0   6 236]]
```

Task 6: Add trigram features

Task 6: Add trigram features

```
# range is (1,3) for extracting unigrams, bigrams, and trigrams
vectorizer = CountVectorizer(ngram_range = (1,3))

# tokenize and build vocabulary
vectorizer.fit(chatgpt_train['review'])
print(vectorizer.vocabulary_, '\n')

# create feature vector representation
vector = vectorizer.transform(chatgpt_train['review'])

print(vector.shape,"\n")

# complete vectors
print(vector.toarray())
```

```
21, 'chatgpt': 17504, 'on': 58432, 'my': 53540, 'windows': 94178, 'desktop': 22813, 'google': 33804, 'chrome': 1839
6, 'while': 93506, 'it': 44026, 'doable': 24127, 'screen': 69557, 'reader': 66160, 'navigation': 54558, 'is': 4239
7, 'pretty': 63984, 'difficult': 23554, 'the': 77753, 'site': 71859, 'and': 4696, 'you': 96345, 'really': 66300, 'h
ave': 35652, 'be': 12331, 'an': 4140, 'advanced': 1970, 'user': 89238, 'find': 29554, 'your': 97009, 'way': 91835,
'through': 83130, 'submitted': 74618, 'numerous': 56594, 'feedbacks': 29094, 'open': 59443, 'ai': 2388, 'about': 55
8, 'but': 15060, 'nothing': 56114, 'has': 35314, 'changed': 17059, 'that': 76814, 'front': 32033, 'well': 92506, 'g
ood': 33615, 'news': 55136, 'ios': 41752, 'app': 7980, 'much': 53253, 'addresses': 1872, 'all': 2939, 'of': 56719,
'those': 82872, 'problems': 64337, 'ui': 87498, 'seems': 70303, 'clean': 18445, 'uncluttered': 87661, 'designed': 2
2761, 'compatible': 19210, 'with': 94270, 'voiceover': 91098, 'built': 15002, 'into': 41434, 'applaud': 9504, 'incl
usivity': 39781, 'design': 22692, 'only': 59201, 'wish': 94206, 'they': 81524, 'would': 95677, 'give': 33168, 'sam
e': 69067, 'attention': 11604, 'care': 16702, 'accessibility': 982, 'experience': 27741, 'given': 33251, 'review':
68373, 'five': 29878, 'stars': 73741, 'just': 46614, 'couple': 21063, 'minor': 52035, 'quibbles': 65639, 'first': 2
9742, 'once': 58942, 'submit': 74607, 'prompt': 64655, 'starts': 73884, 'read': 66123, 'aloud': 3323, 'response': 6
7868, 'before': 13360, 'finished': 29720, 'so': 72233, 'will': 93923, 'hear': 36312, 'few': 29287, 'words': 95225,
'followed': 30178, 'by': 15665, 'reading': 66172, 'stop': 74140, 'generating': 32757, 'button': 15599, 'which': 933
69, 'isn': 43859, 'super': 75007, 'helpful': 36543, 'great': 34405, 'if': 37977, 'could': 20858, 'better': 13767,
'coordinate': 20714, 'alert': 2869, 'didn': 23369, 'start': 73791, 'message': 51751, 'until': 88117, 'had': 34839,
'fully': 32139, 'generated': 32709, 'other': 60563, 'thing': 81739, 'like': 48619, 'feedback': 28980, 'easily': 251
55, 'accessible': 1008, 'from': 31780, 'within': 94984, 'main': 50265, 'make': 50329, 'as': 10563, 'easy': 25199,
'possible': 63536, 'get': 32853, 'continuing': 20113, 'suggestions': 74932, 'users': 89476, 'otherwise': 60725, 'fa
```

```
# use review for model building

train_text = chatgpt_train["review"]
test_text = chatgpt_test["review"]

# set the n-gram range
vectorizer = CountVectorizer(ngram_range = (1,3))

# create training data representation
train_data_repr = vectorizer.fit_transform(train_text.values.astype('U'))
print("NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS, BIGRAMS, TRIGRAMS")
print(train_data_repr.shape,"\n")

# create test data representation
test_data_repr = vectorizer.transform(test_text.values.astype('U'))
print("NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS, BIGRAMS, TRIGRAMS")
print(test_data_repr.shape,"\n")
```

```
NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS, BIGRAMS, TRIGRAMS
(1829, 97367)

NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS, BIGRAMS, TRIGRAMS
(458, 97367)
```

```
from sklearn.naive_bayes import MultinomialNB

# define true labels from train set
X_train = train_data_repr
y_train = chatgpt_train["rating"]
X_test = test_data_repr
y_test = chatgpt_test["rating"]
```

```
# Naive Bayes model
model_unigram_bigram_trigram = MultinomialNB()
model_unigram_bigram_trigram.fit(X_train, y_train)
```

```
MultinomialNB()
```

```
# predict the labels for the test data
predictions = model_unigram_bigram_trigram.predict(X_test)
predictions
```

```
array([1, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 3, 1, 1, 5, 5, 5, 5, 1, 5, 5, 5,
       5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 1, 5, 5, 5, 1, 5, 5, 5,
       5, 1, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 3, 4, 5, 5, 1, 5, 5, 4, 1, 5,
       5, 5, 5, 1, 5, 4, 5, 5, 5, 4, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5,
       5, 5, 5, 5, 5, 1, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5,
       1, 5, 1, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 4, 1, 1, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5,
       5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 1, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5,
       5, 1, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 1, 5, 1, 1, 5, 1, 1, 1, 1, 1, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1,
       1, 5, 1, 1, 5, 5, 5, 5, 1, 5, 5, 5, 5, 1, 5, 1, 4, 1, 4, 5, 5, 1,
       5, 5, 5, 1, 1, 1, 1, 1, 5, 1, 5, 5, 5, 5, 1, 1, 1, 5, 5, 5, 1, 5,
```

Saving the unigram + bigram + trigram model

```
import pickle

filename_3 = 'NaiveBayesModel_unigram_bigram_trigram.sav'
pickle.dump(model_unigram_bigram_trigram, open(filename_3, 'wb'))
```

Loading the unigram + bigram + trigram model

```
loaded_model_unigram_bigram_trigram = pickle.load(open(filename_3, 'rb'))
result = loaded_model_unigram_bigram_trigram.score(X_test, y_test)
print(result)
```

```
0.6200873362445415
```

Evaluating unigram + bigram + trigram model using metrics

```
print ("Accuracy score: ", accuracy_score(y_test, predictions))
print ("Overall Recall score: ", recall_score(y_test, predictions, average='macro'))
print ("Overall Precision score: ", precision_score(y_test, predictions, average='macro'))
print ("Overall F1 score: ", f1_score(y_test, predictions, average='macro'), "\n")
print ("Individual label performance: ")
print (classification_report(y_test, predictions), "\n")
print ("Confusion Matrix: ")
print (confusion_matrix(y_test, predictions))
```

```
Accuracy score:  0.6200873362445415
Overall Recall score:  0.270070126227209
Overall Precision score:  0.2570713391739675
Overall F1 score:  0.250021017234132

Individual label performance:
              precision    recall  f1-score   support

           1       0.65      0.38      0.48       115
           2       0.00      0.00      0.00        26
           3       0.00      0.00      0.00        25
           4       0.00      0.00      0.00        44
           5       0.64      0.97      0.77       248

    accuracy                           0.62       458
   macro avg       0.26      0.27      0.25       458
weighted avg       0.51      0.62      0.54       458


Confusion Matrix:
[[ 44   0   3   3  65]
 [ 10   0   1   2  13]
 [  6   0   0   0  19]
 [  4   0   1   0  39]
 [  4   0   0   4 240]]
```

Task 7: Adding TF-IDF features

Task 7: Add TF-IDF Features

The Naive Bayes with unigram features has the highest overall F1 score = 0.3524

Feature Extraction using TfidfVectorizer for training and testing data on Review¶

```
# Using TF-IDF
text = chatgpt_train["review"]

tf = TfidfVectorizer()
tf.fit(text)

print("VOCABULARY", "\n")
print(tf.vocabulary_,"\n")

# encode document
data = tf.transform(text)

# summarize encoded vector
print("FEATURE SET SHAPE")
print(data.shape,"\n")

# feature set
print("FEATURE SET ARRAY")
print(data.toarray())
```

```
VOCABULARY

{'up': 5224, 'to': 4985, 'this': 4932, 'point': 3613, 've': 5282, 'mostly': 3134, 'been': 493, 'using': 5253, 'chat
gpt': 776, 'on': 3331, 'my': 3165, 'windows': 5454, 'desktop': 1302, 'google': 2139, 'chrome': 806, 'while': 5421,
'it': 2666, 'doable': 1426, 'screen': 4267, 'reader': 3883, 'navigation': 3184, 'is': 2660, 'pretty': 3692, 'diffic
ult': 1352, 'the': 4906, 'site': 4443, 'and': 256, 'you': 5540, 'really': 3891, 'have': 2248, 'be': 474, 'an': 250,
'advanced': 151, 'user': 5250, 'find': 1907, 'your': 5542, 'way': 5382, 'through': 4954, 'submitted': 4698, 'numero
us': 3277, 'feedbacks': 1877, 'open': 3341, 'ai': 197, 'about': 60, 'but': 667, 'nothing': 3255, 'has': 2241, 'chan
ged': 754, 'that': 4903, 'front': 2032, 'well': 5403, 'good': 2135, 'news': 3221, 'ios': 2640, 'app': 291, 'much':
```

Feature Extraction using TfidfVectorizer for training and testing data on Review

```
train_data_tfidf = tf.fit_transform(chatgpt_train['review'])
print("TRAINING DATA FEATURE SET SHAPE")
print(train_data_tfidf.shape,"\n")

test_data_tfidf = tf.transform(chatgpt_test['review'])
print("TEST DATA FEATURE SET SHAPE")
print(test_data_tfidf.shape,"\n")

idf = tf.idf_

# print out feature names and the IDF values
print("FEATURE SET AND IDF VALUES")
print(dict(zip(tf.get_feature_names(), idf)))
```

```
TRAINING DATA FEATURE SET SHAPE
(1829, 5551)

TEST DATA FEATURE SET SHAPE
(458, 5551)

FEATURE SET AND IDF VALUES
{'10': 5.567632266669026, '100': 6.902633333401366, '101': 7.818924065275521, '11': 7.125776884715576, '12': 5.8040
21044733257, '128': 7.818924065275521, '12pro': 7.413458957167357, '13': 6.0271645960474665, '13pro': 7.81892406527
5521, '14': 5.216234379831137, '15': 6.720311776607412, '156': 7.818924065275521, '16': 5.94712188837393, '17': 7.8
18924065275521, '18': 7.413458957167357, '19': 7.125776884715576, '1940': 7.818924065275521, '1949': 7.8189240652755
21, '20': 5.376577029906317, '2007': 7.818924065275521, '2020': 7.818924065275521, '2021': 4.798499179131159, '202
2': 6.902633333401366, '2023': 6.432629704155631, '20mins': 7.413458957167357, '21': 7.818924065275521, '23': 7.818
924065275521, '24': 7.125776884715576, '25': 6.902633333401366, '27': 7.818924065275521, '2nd': 7.413458957167357,
'30': 6.720311776607412, '31': 7.818924065275521, '32k': 7.818924065275521, '35': 7.818924065275521, '360': 7.81892
```

Naive Bayes Model using TF-IDF features

```
# define true labels from train set

X_train = train_data_tfidf
y_train = chatgpt_train["rating"]
X_test = test_data_tfidf
y_test = chatgpt_test["rating"]
```

```
model_tfidf = MultinomialNB()
model_tfidf.fit(X_train, y_train)
```

```
MultinomialNB()
```

```
predictions = model_tfidf.predict(X_test)
predictions
```

```
array([1, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 1, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
```

```
print ("Accuracy score: ", accuracy_score(y_test, predictions))
print ("Overall Recall score: ", recall_score(y_test, predictions, average='macro'))
print ("Overall Precision score: ", precision_score(y_test, predictions, average='macro'))
print ("Overall F1 score: ", f1_score(y_test, predictions, average='macro'), "\n")
print ("Individual label performance: ")
print (classification_report(y_test, predictions), "\n")
print ("Confusion Matrix: ")
print (confusion_matrix(y_test, predictions))
```

```
Accuracy score:  0.5829694323144105
Overall Recall score:  0.23397615708274894
Overall Precision score:  0.24875389408099688
Overall F1 score:  0.20132625994694964

Individual label performance:
              precision    recall  f1-score   support

           1       0.67      0.17      0.28       115
           2       0.00      0.00      0.00        26
           3       0.00      0.00      0.00        25
           4       0.00      0.00      0.00        44
           5       0.58      1.00      0.73       248

    accuracy                           0.58       458
   macro avg       0.25      0.23      0.20       458
weighted avg       0.48      0.58      0.46       458


Confusion Matrix:
[[ 20   0   0   0  95]
 [  3   0   0   0  23]
 [  3   0   0   0  22]
 [  3   0   0   0  41]
 [  1   0   0   0 247]]
```

Task 8: Train model with other columns

A) Title column

Based on the above results, the Naive Bayes model trained on unigram features is the best in terms of accuracy and overall F1 score

Training the Naive Bayes with unigram features on title column

```
# use review for model
train_text_title = chatgpt_train["title"]
test_text_title = chatgpt_test["title"]

# set the n-gram range
vectorizer = CountVectorizer(ngram_range = (1,1))

# create training data representation
train_data_title_repr = vectorizer.fit_transform(train_text_title.values.astype('U'))
print("NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS ON TITLE COLUMN")
print(train_data_title_repr.shape,"\n")

# create test data representation
test_data_title_repr = vectorizer.transform(test_text_title.values.astype('U'))
print("NUMBER OF FEATURES IN TEST DATASET WITH UNIGRAMS ON TITLE COLUMN")
print(test_data_title_repr.shape,"\n")
```

```
NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS ON TITLE COLUMN
(1829, 1435)

NUMBER OF FEATURES IN TEST DATASET WITH UNIGRAMS ON TITLE COLUMN
(458, 1435)
```

```
# define true labels from train set
X_train = train_data_title_repr
y_train = chatgpt_train["rating"]
X_test = test_data_title_repr
y_test = chatgpt_test["rating"]
```

```
# Naive Bayes model
model_unigram_title = MultinomialNB()
model_unigram_title.fit(X_train, y_train)
```

```
MultinomialNB()
```

```
predictions = model_unigram_title.predict(X_test)
predictions
```

```
array([1, 5, 5, 5, 5, 5, 5, 5, 1, 1, 5, 5, 5, 1, 5, 5, 5, 1, 5, 5, 5, 5,
       1, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 1, 1, 5, 5,
       1, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, 4, 5, 5, 1, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 3, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 1, 4, 5, 5,
       5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 5, 5, 5,
       5, 5, 5, 5, 5, 4, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 1,
       1, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 4, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 1, 5, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5,
```

```
print ("Accuracy score: ", accuracy_score(y_test, predictions))
print ("Overall Recall score: ", recall_score(y_test, predictions, average='macro'))
print ("Overall Precision score: ", precision_score(y_test, predictions, average='macro'))
print ("Overall F1 score: ", f1_score(y_test, predictions, average='macro'), "\n")
print ("Individual label performance: ")
print (classification_report(y_test, predictions), "\n")
print ("Confusion Matrix: ")
print (confusion_matrix(y_test, predictions))
```

```
Accuracy score:  0.6135371179039302
Overall Recall score:  0.2935476220833865
Overall Precision score:  0.34369109499057526
Overall F1 score:  0.29203395065226506

Individual label performance:
              precision    recall  f1-score   support

           1       0.66      0.35      0.45       115
           2       0.00      0.00      0.00        26
           3       0.14      0.04      0.06        25
           4       0.29      0.14      0.18        44
           5       0.63      0.94      0.76       248

    accuracy                           0.61       458
   macro avg       0.34      0.29      0.29       458
weighted avg       0.54      0.61      0.55       458


Confusion Matrix:
[[ 40   0   1   6  68]
 [  6   0   3   2  15]
 [  5   0   1   2  17]
 [  3   0   0   6  35]
 [  7   0   2   5 234]]
```

B) Title + Review column

Training the Naive Bayes with unigram features on title + review column

Combine the title and review column for training and test data

```
chatgpt_train['title_review'] = chatgpt_train['title'] + " " + chatgpt_train['review']
chatgpt_test['title_review'] = chatgpt_test['title'] + " " + chatgpt_test['review']
```

Training

```
# use review for model
train_text_title_review = chatgpt_train['title_review']
test_text_title_review = chatgpt_test['title_review']

# set the n-gram range
vectorizer = CountVectorizer(ngram_range = (1,1))

# create training data representation
train_data_title_review_repr = vectorizer.fit_transform(train_text_title_review.values.astype('U'))
print("NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS ON TITLE + REVIEW COLUMN")
print(train_data_title_review_repr.shape,"\n")

# create test data representation
test_data_title_review_repr = vectorizer.transform(test_text_title_review.values.astype('U'))
print("NUMBER OF FEATURES IN TEST DATASET WITH UNIGRAMS ON TITLE + REVIEW COLUMN")
print(test_data_title_review_repr.shape,"\n")
```

```
NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS ON TITLE + REVIEW COLUMN
(1829, 5742)

NUMBER OF FEATURES IN TEST DATASET WITH UNIGRAMS ON TITLE + REVIEW COLUMN
(458, 5742)
```

```
# define true labels from train set

X_train = train_data_title_review_repr
y_train = chatgpt_train["rating"]
X_test = test_data_title_review_repr
y_test = chatgpt_test["rating"]
```

```
# Naive Bayes model
model_unigram_title_review = MultinomialNB()
model_unigram_title_review.fit(X_train, y_train)
```

```
MultinomialNB()
```

```
predictions = model_unigram_title_review.predict(X_test)
predictions
```

```
array([1, 1, 5, 3, 5, 5, 5, 5, 5, 1, 5, 3, 5, 1, 5, 5, 5, 1, 1, 5, 5, 5,
       1, 1, 3, 1, 5, 5, 1, 5, 4, 5, 5, 5, 1, 5, 5, 5, 5, 5, 1, 1, 5, 5,
       1, 1, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 3, 4, 5, 5, 1, 5, 5, 3, 1, 5,
       5, 5, 5, 5, 5, 3, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 1, 3, 5, 5,
       5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5,
       5, 5, 5, 5, 1, 1, 5, 5, 5, 1, 1, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 1,
       1, 1, 1, 5, 1, 1, 5, 5, 5, 5, 5, 2, 2, 5, 3, 3, 5, 5, 5, 5, 5, 5,
       5, 5, 1, 3, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 4, 5, 5, 2, 1, 1, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5,
       5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 3, 3, 5,
       5, 5, 5, 5, 5, 5, 4, 5, 2, 5, 2, 1, 2, 5, 5, 5, 1, 1, 5, 5, 5, 5,
       5, 5, 4, 5, 5, 5, 1, 3, 1, 5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5, 5,
       1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 4, 5, 5, 5, 5,
       1, 4, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5, 5, 1, 5, 5, 5, 5, 5, 5, 5
```

```
print ("Accuracy score: ", accuracy_score(y_test, predictions))
print ("Overall Recall score: ", recall_score(y_test, predictions, average='macro'))
print ("Overall Precision score: ", precision_score(y_test, predictions, average='macro'))
print ("Overall F1 score: ", f1_score(y_test, predictions, average='macro'), "\n")
print ("Individual label performance: ")
print (classification_report(y_test, predictions), "\n")
print ("Confusion Matrix: ")
print (confusion_matrix(y_test, predictions))
```

```
Accuracy score:  0.6419213973799127
Overall Recall score:  0.3558083052991889
Overall Precision score:  0.4300508552935737
Overall F1 score:  0.3694878450223124

Individual label performance:
              precision    recall  f1-score   support

           1       0.58      0.50      0.53       115
           2       0.29      0.08      0.12        26
           3       0.09      0.08      0.09        25
           4       0.48      0.23      0.31        44
           5       0.72      0.90      0.80       248

    accuracy                           0.64       458
   macro avg       0.43      0.36      0.37       458
weighted avg       0.60      0.64      0.61       458


Confusion Matrix:
[[ 57   2   6   4  46]
 [ 10   2   5   2   7]
 [ 11   0   2   2  10]
 [  7   0   4  10  23]
 [ 14   3   5   3 223]]
```

Task 9: Interpret and Classification Results

There are multiple models that we have trained and on the basis of them, the Naive Bayes model with unigram feature set trained on the combination of the title and review column has the maximum accuracy of 64.19%. In addition to this, the recall, precision and F1 score for the same model has the highest values among all.
This indicates that this model is good at classifying the most positive instances and good at making positive predictions and minimising the false positives and false negatives.

Unigrams are the individual words taken from the text document the constitute the feature space. The presence or absence of these words is used for classification.

Unigrams work well with Naive Bayes because:

- They are easy to compute and the simplest form of features.
- Naive Bayes is dependent on the frequency of words and unigrams can clearly indicate the count of words in text document.
- The 'title' column gives us the succinct information or key sentiment about the review and the 'review' column consists of the in-depth review. The 'review' column is more useful than title when considered alone. But, choosing the combination of title and review columns can enhance the available information useful for classification.
- A major reason why unigrams work well is because the number of features is the least when compared with other n-grams and hence they can avoid overfitting when feature space is less. When compared with TF-IDF features, we may lose the context and interpretability of words.

Metrics:

- Accuracy Score (0.642): This metric measures the overall correctness of your model's predictions. It indicates that approximately 64.2% of the instances were correctly classified.

- Overall Recall Score (0.356): Recall represents the model's ability to correctly identify instances of a particular class. An overall recall of 0.356 suggests that the model has a relatively low ability to capture all instances of the positive class across all classes.

- Overall Precision Score (0.430): Precision measures the accuracy of positive predictions made by the model. An overall precision of 0.430 indicates that when the model predicts a positive class, it is correct approximately 43% of the time.

- Overall F1 Score (0.369): The F1 score provides a balanced measure of both precision and recall. An overall F1 score of 0.369 suggests that the model's performance is relatively balanced between precision and recall but still relatively low.

Individual Class Performance:

- Class 1: For this class, the model exhibits moderate precision (0.58) but only moderately high recall (0.50). The performance is balanced, as evidenced by the F1 score's (0.53), which is in the middle.

- Class 2: The model's F1 score (0.12) is low due to the model's low recall (0.08) and precision (0.29) for this class. This may indicate that the model finds it difficult to assign the proper class to instances in this class.

- Class 3: The model has a low F1 score (0.09) for this class due to its low precision (0.09) and recall (0.08), which are comparable to those of Class 2.

- Class 4: The model's precision for this class is moderate (0.48), but its recall is relatively low (0.23), resulting in an F1 score of 0.31, which denotes a partial performance balance.

- Class 5: The model exhibits strong performance in this class, with high precision (0.72), recall (0.90), and F1 score (0.80). Most Class 5 instances are correctly identified by it.

Confusion Matrix:

- The confusion matrix provides a breakdown of the model's predictions and true values for each class. It shows where the model correctly predicted each class and where it made errors.

- From the confusion matrix, we can see the distribution of true positives, false positives, false negatives, and true negatives for each class, helping identify which classes are more challenging for the model.

Task 10: Error Analysis

Task 10: Error Analysis

```
prediction = predictions
labels = chatgpt_test["rating"]
inputs = chatgpt_test["title"] + " --- " + chatgpt_test["review"]

unique_rating = chatgpt_test["rating"].unique()
unique_rating.sort()

for rating in unique_rating:
    count = 0
    print(f"RATING {rating!r}")
    print("----------")

    for i in range(len(inputs)):
        if prediction[i] != labels[i] and labels[i] == rating and count < 3:
            count += 1
            print("Title --- Review: ", inputs[i])
            print(f'It has been classified with rating {prediction[i]!r} but has actual rating {labels[i]!r}')
            print ("\n")
```

```
RATING 1
----------
Title --- Review:  your gpt 4 is fake --- Fix it
It has been classified with rating 5 but has actual rating 1


Title --- Review:  Version 3 --- Old version of ChatGPT.
It has been classified with rating 5 but has actual rating 1


Title --- Review:  Unusable on iPad --- Unusable on iPad Pro currently.
It has been classified with rating 3 but has actual rating 1



RATING 2
----------
Title --- Review:  error unsupported country --- cant login
It has been classified with rating 1 but has actual rating 2


Title --- Review:  no gpt4.0 ???cheater --- i subscribe the plus for gpt4???but just get gpt3.0 with fake label 4.
0.release real gpt4 now?????????
It has been classified with rating 1 but has actual rating 2


Title --- Review:  Outdated --- This app is based on chatgpt 3 (cutoff date Sep 2021)
It has been classified with rating 3 but has actual rating 2
```

```
RATING 3
----------
Title --- Review:  Why not support ios15? --- ????
It has been classified with rating 1 but has actual rating 3


Title --- Review:  Even --- Ok
It has been classified with rating 1 but has actual rating 3


Title --- Review:  ?????????ipad --- ipad is not supported
It has been classified with rating 1 but has actual rating 3
```

```
RATING 4
----------
Title --- Review:  First --- First
It has been classified with rating 5 but has actual rating 4


Title --- Review:  WOW --- I am in love with CHATGPT
It has been classified with rating 5 but has actual rating 4


Title --- Review:  great --- great!
It has been classified with rating 5 but has actual rating 4
```

```
RATING 5
----------
Title --- Review:  Please impose IPadOS --- We need IPadOS!!!
It has been classified with rating 3 but has actual rating 5


Title --- Review:  ?????????????????????? --- ??????????????????????????plus??????????
€?????????????????????????????????????????????????????????????????????€????????????????????????????????????????????
It has been classified with rating 1 but has actual rating 5


Title --- Review:  Vietnam --- H??y b??? sung v??ng Vi???t Nam ????? t??i c?? th??? mua g??i plus v?? kh??ng ph???i
chuy???n v??ng qua M??? ????? t???i
It has been classified with rating 1 but has actual rating 5
```

- When we look at the example from rating 1, ‘Tile --- Review: your gpt 4 is fake --- Fix it’, was humans we can understand that the customer has faced some issues with gpt4 and hence have given bad comments about the same. But, in the combination of title and review there is no use of strong words that indicate negative sentiments. The statement is more imperative and hence may have been classified as 5 when in reality it is a bad review with label 1.

- The example ‘Title --- Review: no gpt4.0 ???cheater --- i subscribe the plus for gpt4???but just get gpt3.0 with fake label 4.0.release real gpt4 now?????????’. In this as well the customer has been cheated of gpt4 and it is a bad review that has an actual rating of 2. But, there are words like ‘no’ that might have indicated the model to consider it as a negative statement hence giving it the lowest class of 1.

- The example ‘Title --- Review: WOW --- I am in love with CHATGPT’ has been classified with a rating of 5, when actually it is a 4-start review. When we observe the statement, we see that there is an interjection of ‘WOW’ and positive words like ‘love’. Due to the presence of such words, the classifier considered it as a extremely positive statement hence giving it a 5 star rating.

To improve the model performance we can try some things like:

- Augmenting the data by adding different statements that capture different words or phrases or context, languages etc. This will help by learning from diverse knowledge and generalise better.

- The major problem is understanding the context of the statements. Hence performing semantic analysis and discourse analysis may help.

- We can make use of ensemble techniques by combining the decision of multiple single classifiers to get an average or majority voting.

- Cross validation can be performed.

Task 11: Three-way classification vs five-way classification

Task 11: Three-way vs five-way classification

Creating a new column for 3 way classes -> positive, negative and neutral

```
def three_way_classes(rating):
    if rating == 1 or rating == 2:
        val = 'negative'
    elif rating == 4 or rating == 5:
        val = 'positive'
    else:
        val = 'neutral'

    return val
```

Apply it to the train and test dataset

```
chatgpt_train['3_way_classes'] = chatgpt_train['rating'].apply(three_way_classes)
chatgpt_train[['3_way_classes']]
```

	3_way_classes
0	positive
1	positive
2	positive
3	positive
4	positive
...	...
1829	negative
1830	positive
1831	negative

```
chatgpt_test['3_way_classes'] = chatgpt_test['rating'].apply(three_way_classes)
chatgpt_test[['3_way_classes']]
```

	3_way_classes
0	negative
1	negative
2	negative
3	positive
4	positive
...	...
453	positive
454	positive
455	positive
456	negative
457	positive

458 rows × 1 columns

Naive Bayes model trained on unigram features is the best in terms of accuracy and overall F1 score

Training the Naive Bayes with unigram features on the 3 way classification

```
# use review for model
train_text_title_review = chatgpt_train['review']
test_text_title_review = chatgpt_test['review']

# set the n-gram range
vectorizer = CountVectorizer(ngram_range = (1,1))

# create training data representation
train_data_title_review_repr = vectorizer.fit_transform(train_text_title_review.values.astype('U'))
print("NUMBER OF FEATURES IN TRAINING DATASET WITH 3 WAY CLASSICATION")
print(train_data_title_review_repr.shape,"\n")

# create test data representation
test_data_title_review_repr = vectorizer.transform(test_text_title_review.values.astype('U'))
print("NUMBER OF FEATURES IN TEST DATASET WITH 3 WAY CLASSICATION")
print(test_data_title_review_repr.shape,"\n")
```

```
NUMBER OF FEATURES IN TRAINING DATASET WITH 3 WAY CLASSICATION
(1829, 5551)

NUMBER OF FEATURES IN TEST DATASET WITH 3 WAY CLASSICATION
(458, 5551)
```

```
# define true labels from train set

X_train = train_data_title_review_repr
y_train = chatgpt_train["3_way_classes"]
X_test = test_data_title_review_repr
y_test = chatgpt_test["3_way_classes"]
```

```
# Naive Bayes model

model_3_way_classification = MultinomialNB()
model_3_way_classification.fit(X_train, y_train)
```

```
MultinomialNB()
```

```
predictions = model_3_way_classification.predict(X_test)
predictions
```

```
array(['negative', 'positive', 'negative', 'positive', 'positive',
       'positive', 'positive', 'positive', 'positive', 'positive',
       'positive', 'negative', 'negative', 'negative', 'positive',
       'positive', 'positive', 'negative', 'negative', 'positive',
       'positive', 'positive', 'negative', 'positive', 'positive',
       'negative', 'positive', 'positive', 'negative', 'positive',
       'positive', 'positive', 'positive', 'positive', 'negative',
       'positive', 'negative', 'positive', 'positive', 'positive',
       'negative', 'neutral', 'positive', 'positive', 'positive',
       'negative', 'positive', 'positive', 'positive', 'positive',
       'positive', 'positive', 'negative', 'positive', 'negative',
       'positive', 'neutral', 'neutral', 'positive', 'positive',
       'negative', 'positive', 'positive', 'negative', 'negative',
       'positive', 'positive', 'positive', 'positive', 'negative',
       'positive', 'positive', 'positive', 'negative', 'positive',
       'negative', 'positive', 'positive', 'positive', 'positive',
```

```
print ("Accuracy score: ", accuracy_score(y_test, predictions))
print ("Overall Recall score: ", recall_score(y_test, predictions, average='macro'))
print ("Overall Precision score: ", precision_score(y_test, predictions, average='macro'))
print ("Overall F1 score: ", f1_score(y_test, predictions, average='macro'), "\n")
print ("Individual label performance: ")
print (classification_report(y_test, predictions), "\n")
print ("Confusion Matrix: ")
print (confusion_matrix(y_test, predictions))
```

```
Accuracy score:  0.7358078602620087
Overall Recall score:  0.5007752841737103
Overall Precision score:  0.5420844714259355
Overall F1 score:  0.5116057233704292

Individual label performance:
              precision    recall  f1-score   support

    negative       0.69      0.53      0.60       141
     neutral       0.17      0.08      0.11        25
    positive       0.77      0.89      0.83       292

    accuracy                           0.74       458
   macro avg       0.54      0.50      0.51       458
weighted avg       0.71      0.74      0.72       458


Confusion Matrix:
[[ 75   4  62]
 [  8   2  15]
 [ 26   6 260]]
```

Comparison of three-way classification vs five-way classification:

- In three-way classification, we have to classify the texts into just 3 categories of positive, negative or neutral.

- Since there are less number of classes, it is simple and easy to differentiate the classes. As opposed to five-way classification, as we have more number of classes identifying tight boundaries to categorise the texts is difficult as evident from the above experiments.

- As we observe, the accuracy and metrics like precision, recall and F1 score are very good for three-way classification when compared to five-way classification.

- The accuracy for three-way classification is as high as **74%** against **64%**.

- In terms of overall performance, three-way classification performs better than ive-way classification in terms of accuracy, recall, precision, and F1 score. Three-way classification seems to be more capable of discriminating between its three classes (negative, neutral, and positive) than ive-way classification, which has trouble with its five classes. Additionally, three-way classification has a higher F1 score, indicating a better harmony between recall and precision.

Why observe the difference in three-way versus five-way classification?

There can be many reasons behind this:

- Since three-way classification has lesser number of classes than five-way classification, it is easier to distinguish the classes and the boundaries can be little bit relaxed. But in five-way due to more number of classes, tighter decision boundaries need to be enforced and this may be difficult to establish.

- Data may be imbalanced and there can be fewer training examples for some of the classes. In our case, as we see, the class 5 has the highest number of training examples (885). Then we have class 1 which has less that half the number of samples of class 5. The class 2 has the lowest number of training examples. Hence, the model was unable to clearly distinguish the class 2 and 3 reviews in majority of the experiments.

```
chatgpt_train['rating'].value_counts()
```

```
5    885
1    379
4    258
3    195
2    112
Name: rating, dtype: int64
```

- Due to an increase in the number of classes, feature space may also increase leading to overfitting.

References:

1) geeksforgeeks.com

2) Canvas Class Slides: https://canvas.illinois.edu/courses/37566/pages/course-schedule

3) https://www.nltk.org/book/ch05.html

4) https://www.nltk.org/book/ch06.html

4) https://www.nltk.org/