In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import nltk
from sklearn.feature_extraction.text import CountVectorizer
from wordcloud import WordCloud,STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import re,string,unicodedata
from sklearn.metrics import classification_report,confusion_matrix,accuracy_s
from sklearn.model_selection import train_test_split
from string import punctuation
from nltk import pos_tag
from nltk.corpus import wordnet
import re
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
```

In [2]:
```
%%time
df = pd.read_csv('comments.csv')
```

```
Wall time: 162 ms
```

In [3]:
```
df.shape
```

Out[3]:
```
(18409, 5)
```

In [4]:
```
df = df[:150000]
```

In [5]:
```
df.head()
```

Out[5]:

	Unnamed: 0	Video ID	Comment	Likes	Sentiment
0	0	wAZZ-UWGVHI	Let's not forget that Apple Pay in 2014 requir...	95.0	1.0
1	1	wAZZ-UWGVHI	Here in NZ 50% of retailers don’t even have co...	19.0	0.0
2	2	wAZZ-UWGVHI	I will forever acknowledge this channel with t...	161.0	2.0
3	3	wAZZ-UWGVHI	Whenever I go to a place that doesn’t take App...	8.0	0.0
4	4	wAZZ-UWGVHI	Apple Pay is so convenient, secure, and easy t...	34.0	2.0

In [6]:
```
df.shape
```

Out[6]:
```
(18409, 5)
```

In [7]:
```
df = df.drop(columns=['Unnamed: 0'])
```

In [8]:
```
df.head()
```

Out[8]:

	Video ID	Comment	Likes	Sentiment
0	wAZZ-UWGVHI	Let's not forget that Apple Pay in 2014 requir...	95.0	1.0
1	wAZZ-UWGVHI	Here in NZ 50% of retailers don't even have co...	19.0	0.0
2	wAZZ-UWGVHI	I will forever acknowledge this channel with t...	161.0	2.0
3	wAZZ-UWGVHI	Whenever I go to a place that doesn't take App...	8.0	0.0
4	wAZZ-UWGVHI	Apple Pay is so convenient, secure, and easy t...	34.0	2.0

In [9]:
```
#Customize stopword as per data
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
new_stopwords = ["would","shall","could","might"]
stop_words.extend(new_stopwords)
stop_words.remove("not")
stop_words=set(stop_words)
print(stop_words)
```

```
{'now', 'about', "couldn't", 'just', 'only', 'all', 'what', 'him', 'but', 'w
ill', 'can', 'of', 'itself', 've', 'mustn', "it's", 'doing', 'who', 'if', 'h
imself', 're', 'isn', 'ain', "that'll", "don't", 'too', 'aren', 'it', 'or',
'over', 'them', 'myself', 'theirs', 'does', 'an', 'after', 'would', 'could',
'haven', 'with', 'off', 'for', "doesn't", 'had', 'yourselves', 'was', 'mor
e', 'so', "you're", "you'd", 'ourselves', 'because', 'been', 's', 'don', 'wh
ile', "aren't", 'when', 'through', 'where', 'my', 'most', 'be', 'at', 'again
st', 'from', 'a', 'might', "haven't", 'each', "didn't", 'these', 'he', 'unti
l', 'needn', 'm', 'didn', "isn't", 'should', "she's", 'again', 'did', 'shal
l', 'same', 'weren', 'which', 'then', "weren't", 'very', 'wasn', 'herself',
'how', 'both', 'no', 'i', 'above', 'before', 'on', 'themselves', 'are', 'ow
n', "won't", 'have', 'and', 'we', 'shouldn', "wasn't", 'has', 't', 'won', 'a
ny', 'than', 'between', 'there', 'why', 'her', 'mightn', 'further', 'doesn',
"should've", "you'll", 'our', 'once', 'they', 'into', 'during', 'other',
'd', 'shan', 'below', 'yours', 'few', 'its', 'wouldn', "hasn't", 'by', 'who
m', 'were', 'nor', 'is', 'me', 'the', 'to', 'y', 'hasn', 'your', 'some', "ha
dn't", "shan't", 'being', 'his', 'she', 'am', 'hers', 'this', "shouldn't",
'll', 'ours', "mustn't", 'do', 'o', 'in', "you've", "mightn't", 'up', 'had
n', "needn't", 'as', 'that', 'having', 'ma', 'couldn', "wouldn't", 'those',
'such', 'out', 'you', 'down', 'here', 'their', 'yourself', 'under'}
```

In [10]:
```
'''-----------------------------Data Cleaning and Preprocessing pipeline-----

#Removing special character
def remove_special_character(content):
    return re.sub('\W+',' ', content )#re.sub('\[[^&@#!]]*\]', '', content)

# Removing URL's
def remove_url(content):
    return re.sub(r'http\S+', '', content)


#Removing the stopwords from text
def remove_stopwords(content):
    clean_data = []
    for i in content.split():
        if i.strip().lower() not in stop_words and i.strip().lower().isalpha(
            clean_data.append(i.strip().lower())
    return " ".join(clean_data)

# Function to expand English contractions
def contraction_expansion(content):
    contractions = {
        r"won\'t": "would not", r"can\'t": "can not", r"don\'t": "do not", r"
        r"needn\'t": "need not", r"hasn\'t": "has not", r"haven\'t": "have no
        r"mightn\'t": "might not", r"didn\'t": "did not", r"it\'s": "it is",
    }
    for pattern, replacement in contractions.items():
        content = re.sub(pattern, replacement, content)
    return content

# Data preprocessing function
def data_cleaning(content):
    if not isinstance(content, str):
        return ''
    content = contraction_expansion(content)
    content = remove_special_character(content)
    content = remove_url(content)
    content = remove_stopwords(content)
    return content
```

In [11]:
```
%%time
df['cleaned_comments'] = df['Comment'].apply(data_cleaning)
print(df)
```

```
          Video ID                                            Comment  Likes  \
0      wAZZ-UWGVHI  Let's not forget that Apple Pay in 2014 requir...   95.0   
1      wAZZ-UWGVHI  Here in NZ 50% of retailers don’t even have co...   19.0   
2      wAZZ-UWGVHI  I will forever acknowledge this channel with t...  161.0   
3      wAZZ-UWGVHI  Whenever I go to a place that doesn’t take App...    8.0   
4      wAZZ-UWGVHI  Apple Pay is so convenient, secure, and easy t...   34.0   
...            ...                                                ...    ...   
18404  cyLWtMSry58  I really like the point about engineering tool...    0.0   
18405  cyLWtMSry58  I’ve just started exploring this field. And th...   20.0   
18406  cyLWtMSry58  Excelente video con una pregunta filosófica pr...    1.0   
18407  cyLWtMSry58  Hey Daniel, just discovered your channel a cou...   35.0   
18408  cyLWtMSry58  This is great. Focus is key. A playful approac...    0.0   

       Sentiment                                   cleaned_comments  
0            1.0  let not forget apple pay required brand new ip...  
1            0.0  nz retailers even contactless credit card mach...  
2            2.0  forever acknowledge channel help lessons ideas...  
3            0.0  whenever go place take apple pay happen often ...  
4            2.0  apple pay convenient secure easy use used kore...  
...          ...                                                ...  
18404        2.0  really like point engineering toolboxes think ...  
18405        2.0  started exploring field really good reminder g...  
18406        1.0  excelente video con una pregunta filosófica pr...  
18407        2.0  hey daniel discovered channel couple days ago ...  
18408        2.0  great focus key playful approach also speed th...  

[18409 rows x 5 columns]
Wall time: 1.55 s
```

In [12]:
```
df.head()
```

Out[12]:

	Video ID	Comment	Likes	Sentiment	cleaned_comments
0	wAZZ-UWGVHI	Let's not forget that Apple Pay in 2014 requir...	95.0	1.0	let not forget apple pay required brand new ip...
1	wAZZ-UWGVHI	Here in NZ 50% of retailers don’t even have co...	19.0	0.0	nz retailers even contactless credit card mach...
2	wAZZ-UWGVHI	I will forever acknowledge this channel with t...	161.0	2.0	forever acknowledge channel help lessons ideas...
3	wAZZ-UWGVHI	Whenever I go to a place that doesn’t take App...	8.0	0.0	whenever go place take apple pay happen often ...
4	wAZZ-UWGVHI	Apple Pay is so convenient, secure, and easy t...	34.0	2.0	apple pay convenient secure easy use used kore...

In [13]:
```
#Checking for missing value
df.isna().sum()
```

Out[13]:
```
Video ID            0
Comment             1
Likes               0
Sentiment           0
cleaned_comments    0
dtype: int64
```

In [14]:
```
df['cleaned_comments'].describe()
```

Out[14]:
```
count     18409
unique    17731
top
freq         52
Name: cleaned_comments, dtype: object
```

In [20]:
```
df.head()
```

Out[20]:

	Video ID	Comment	Likes	Sentiment	cleaned_comments
0	wAZZ-UWGVHI	Let's not forget that Apple Pay in 2014 requir...	95.0	1.0	let not forget apple pay required brand new ip...
1	wAZZ-UWGVHI	Here in NZ 50% of retailers don’t even have co...	19.0	0.0	nz retailers even contactless credit card mach...
2	wAZZ-UWGVHI	I will forever acknowledge this channel with t...	161.0	2.0	forever acknowledge channel help lessons ideas...
3	wAZZ-UWGVHI	Whenever I go to a place that doesn’t take App...	8.0	0.0	whenever go place take apple pay happen often ...
4	wAZZ-UWGVHI	Apple Pay is so convenient, secure, and easy t...	34.0	2.0	apple pay convenient secure easy use used kore...

In [21]:
```
df = df.rename(columns={'Video ID': 'Video_ID'})
df = df.rename(columns={'Sentiment': 'sentiment'})
df = df.rename(columns={'Comment': 'comment_text'})
df = df.rename(columns={'Video_ID': 'video_id'})
```

In [22]:
```
print('Unique comments:%s' % df.cleaned_comments.nunique())
print('Unique videos:%s' % df.video_id.nunique())
print('Unique Sentiments:%s ' %df.sentiment.nunique())
```

```
Unique comments:17731
Unique videos:1869
Unique Sentiments:3
```

In [23]:
```
%%time
'''----------------------------Visualization of Important Words from Positive
sentences=df['cleaned_comments']
negative=' '.join(map(str,sentences[df['sentiment']==1.0]))
neutral=' '.join(map(str,sentences[df['sentiment']==0.0]))
positive=' '.join(map(str,sentences[df['sentiment']==2.0]))

positive_wordcloud = WordCloud(width = 1500, height = 800,
                background_color ='black',
                stopwords = stop_words,
                min_font_size = 15).generate(positive)

plt.figure(figsize=(10,10))
plt.imshow(positive_wordcloud)
plt.title('Positive Comments')
plt.axis('off')
```

```
Wall time: 7.26 s
```

Out[23]:
```
(-0.5, 1499.5, 799.5, -0.5)
```

Positive Comments

In [24]:
```
neutral_wordcloud = WordCloud(width = 1500, height = 800,
                background_color ='black',
                stopwords = stop_words,
                min_font_size = 15).generate(neutral)

plt.figure(figsize=(10,10))
plt.imshow(neutral_wordcloud)
plt.title('Neutral Comments')
plt.axis('off')
```

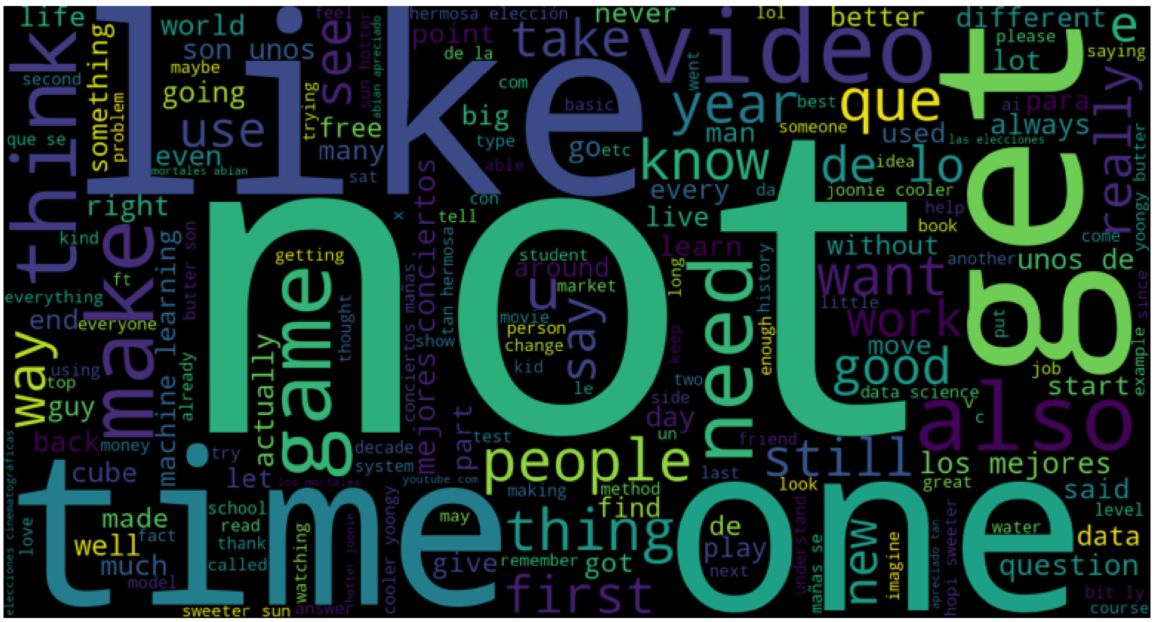
Out[24]: (-0.5, 1499.5, 799.5, -0.5)

Neutral Comments

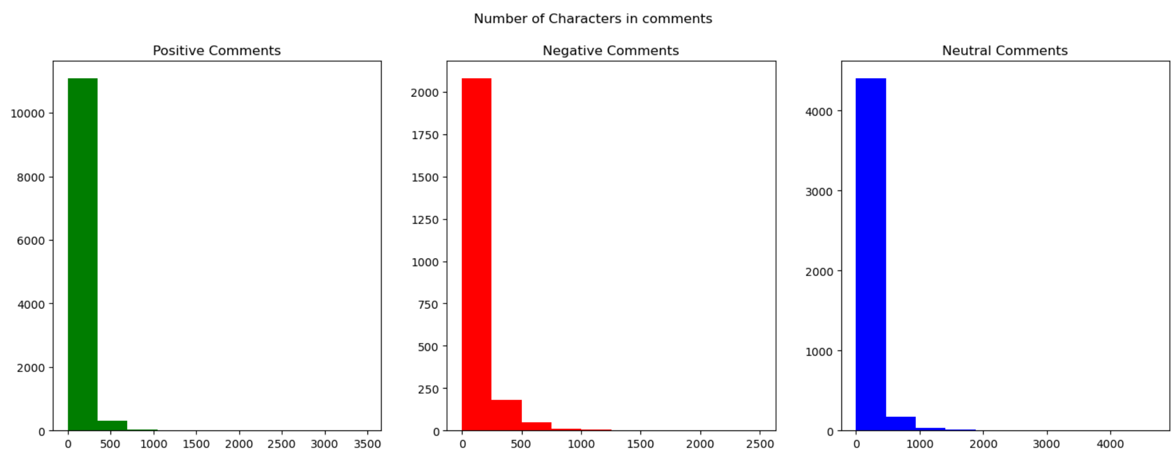
In [25]:
```
negative_wordcloud = WordCloud(width = 1500, height = 800,
                    background_color ='black',
                    stopwords = stop_words,
                    min_font_size = 15).generate(negative)

plt.figure(figsize=(10,10))
plt.imshow(negative_wordcloud)
plt.title('Negative Comments')
plt.axis('off')
```

Out[25]: (-0.5, 1499.5, 799.5, -0.5)

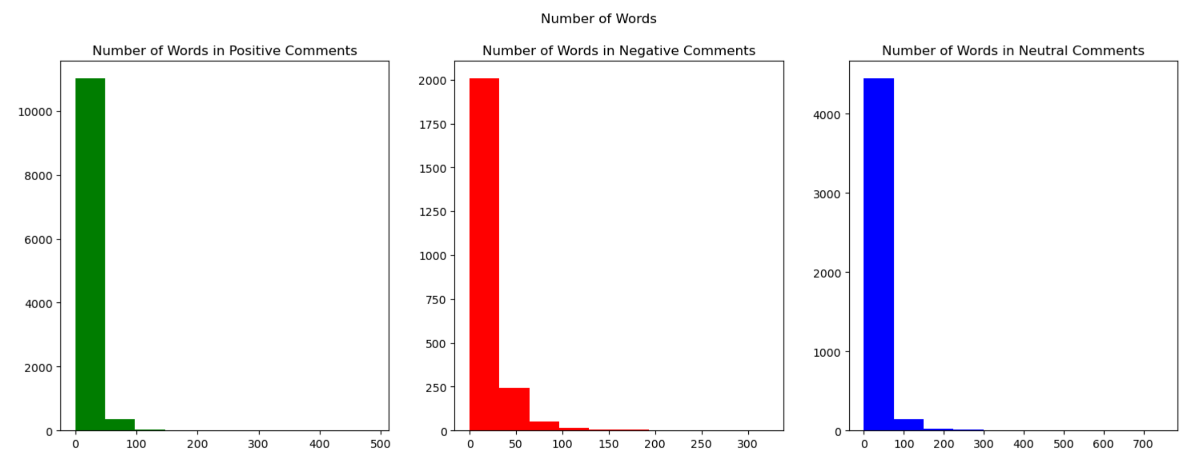
Negative Comments



In [26]:
```
# Visualization of number of characters in reviews
figure, (pos_ax, neg_ax, neu_ax) = plt.subplots(1, 3, figsize=(18, 6))

# Length of positive comments
len_pos_comment = df[df['sentiment'] == 2]['cleaned_comments'].str.len()
pos_ax.hist(len_pos_comment, color='green')
pos_ax.set_title('Positive Comments')

# Length of negative comments
len_neg_comment = df[df['sentiment'] == 0]['cleaned_comments'].str.len()
neg_ax.hist(len_neg_comment, color='red')
neg_ax.set_title('Negative Comments')

# Length of neutral comments
len_neu_comment = df[df['sentiment'] == 1]['cleaned_comments'].str.len()
neu_ax.hist(len_neu_comment, color='blue')
neu_ax.set_title('Neutral Comments')

# Super title for the figure
figure.suptitle('Number of Characters in comments')

plt.show()
```

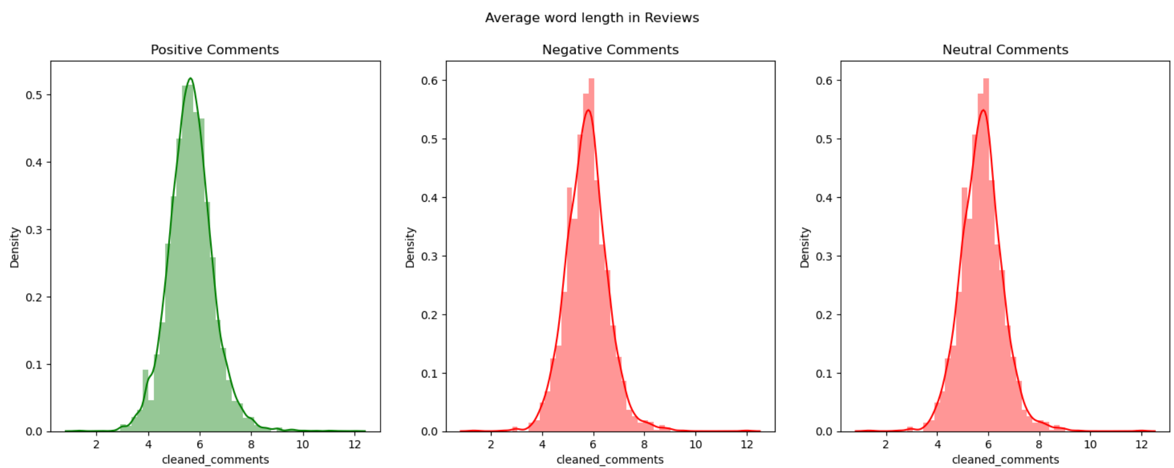


In [27]:
```
# Visualization of number of words in comments
figure, (pos_ax, neg_ax, neu_ax) = plt.subplots(1, 3, figsize=(18, 6))

# Length of positive comments
pos_words = df[df['sentiment'] == 2]['cleaned_comments'].str.split().map(lamb
pos_ax.hist(pos_words, color='green')
pos_ax.set_title('Number of Words in Positive Comments')

# Length of negative comments
neg_words = df[df['sentiment'] == 0]['cleaned_comments'].str.split().map(lamb
neg_ax.hist(neg_words, color='red')
neg_ax.set_title('Number of Words in Negative Comments')

# Length of neutral comments
neu_words = df[df['sentiment'] == 1]['cleaned_comments'].str.split().map(lamb
neu_ax.hist(neu_words, color='blue')
neu_ax.set_title('Number of Words in Neutral Comments')

# Super title for the figure
figure.suptitle('Number of Words ')

plt.show()
```



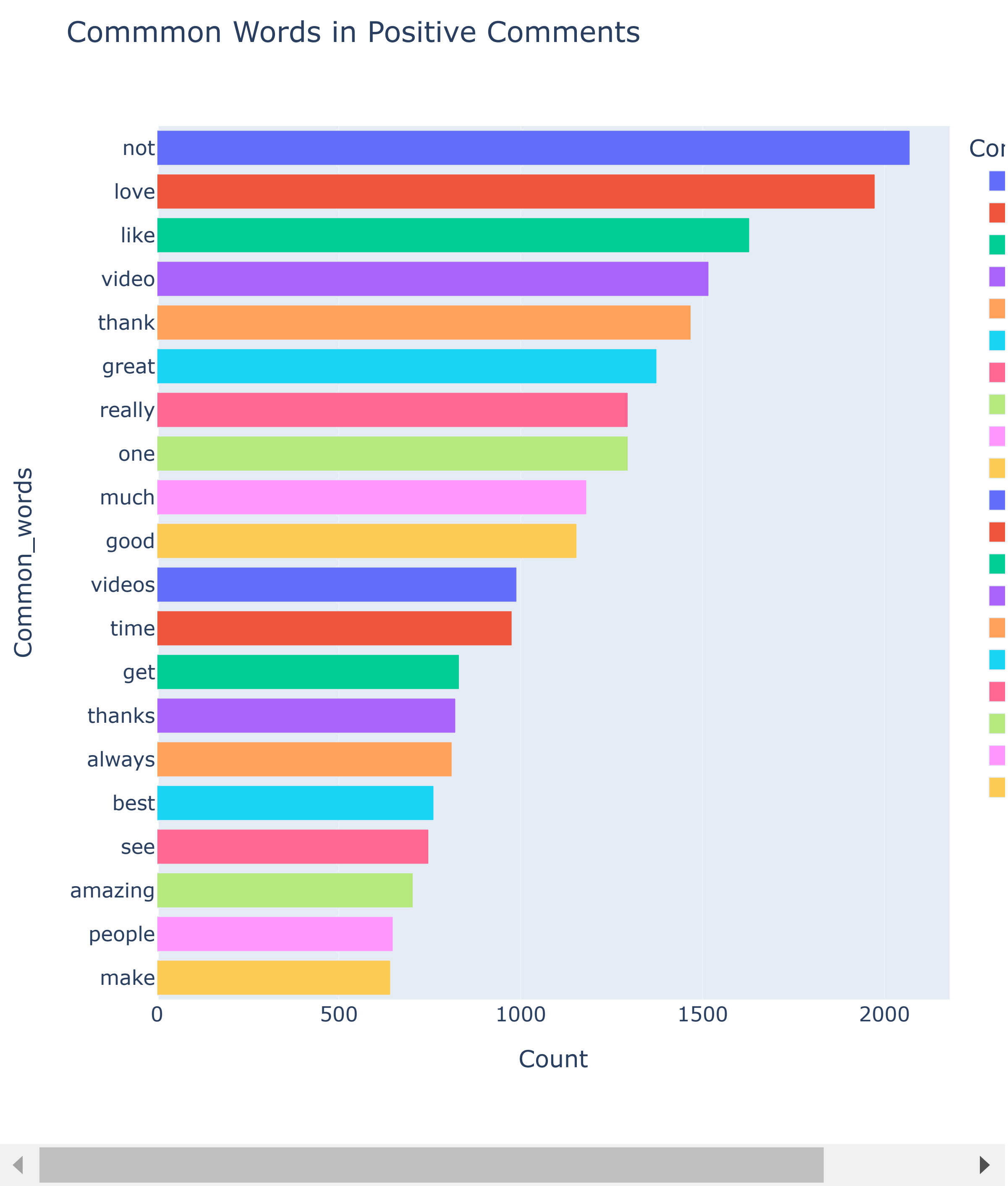
In [28]:
```
figure, (pos_ax, neg_ax, neu_ax) = plt.subplots(1, 3, figsize=(18, 6))

pos_word=df[df['sentiment']==2]['cleaned_comments'].str.split().apply(lambda 
sns.distplot(pos_word.map(lambda x: np.mean(x)),ax=pos_ax,color='green')
pos_ax.set_title('Positive Comments')

neg_word=df[df['sentiment']==0]['cleaned_comments'].str.split().apply(lambda 
sns.distplot(neg_word.map(lambda x: np.mean(x)),ax=neg_ax,color='red')
neg_ax.set_title('Negative Comments')

neu_word=df[df['sentiment']==0]['cleaned_comments'].str.split().apply(lambda 
sns.distplot(neu_word.map(lambda x: np.mean(x)),ax=neu_ax,color='red')
neu_ax.set_title('Neutral Comments')

figure.suptitle('Average word length in Reviews')
```

Out[28]:
```
Text(0.5, 0.98, 'Average word length in Reviews')
```

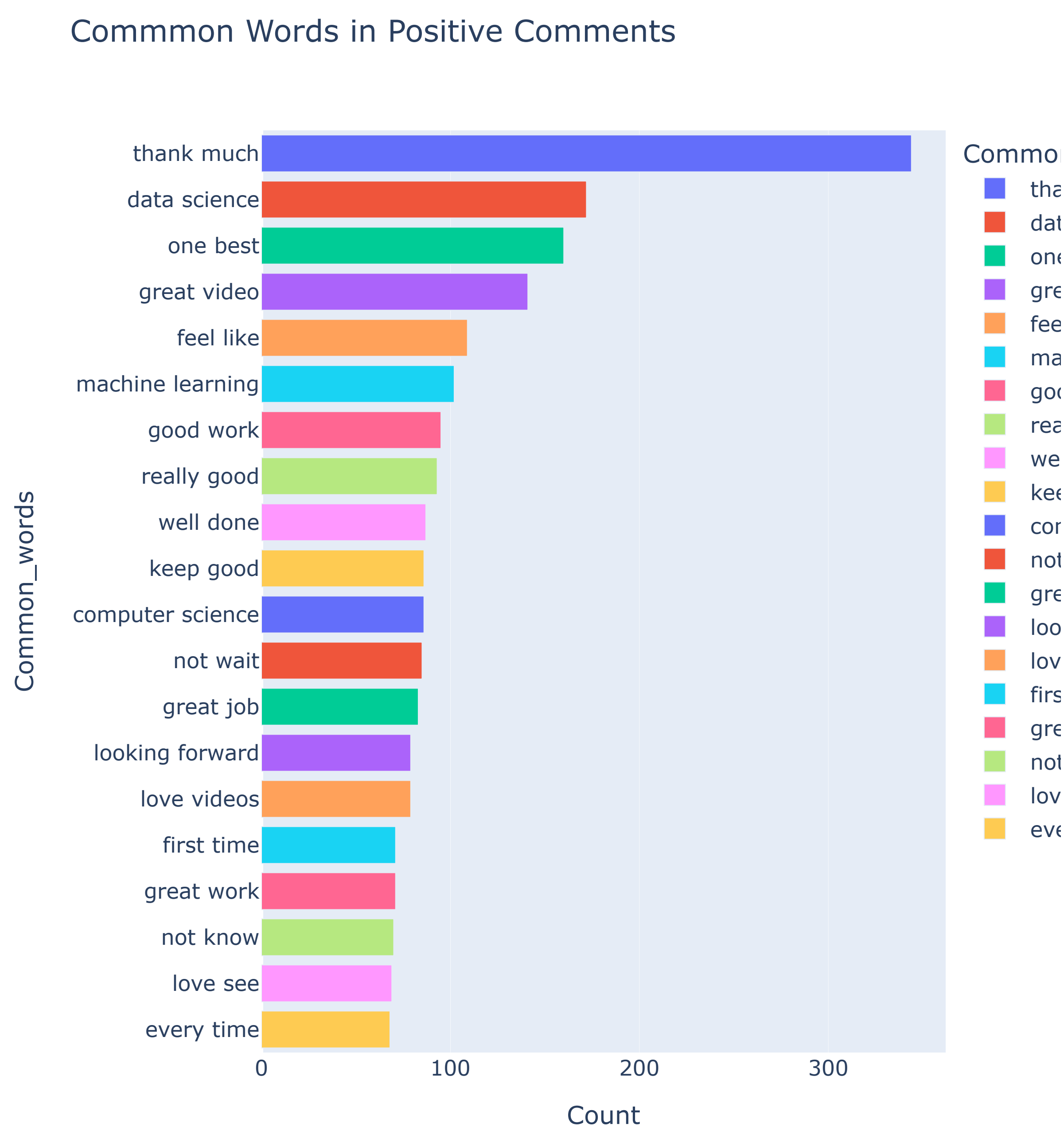


In [29]:
```
#Get important feature by using Countvectorizer
def get_top_text_ngrams(corpus, n, g):
    vec = CountVectorizer(ngram_range=(g, g)).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
```

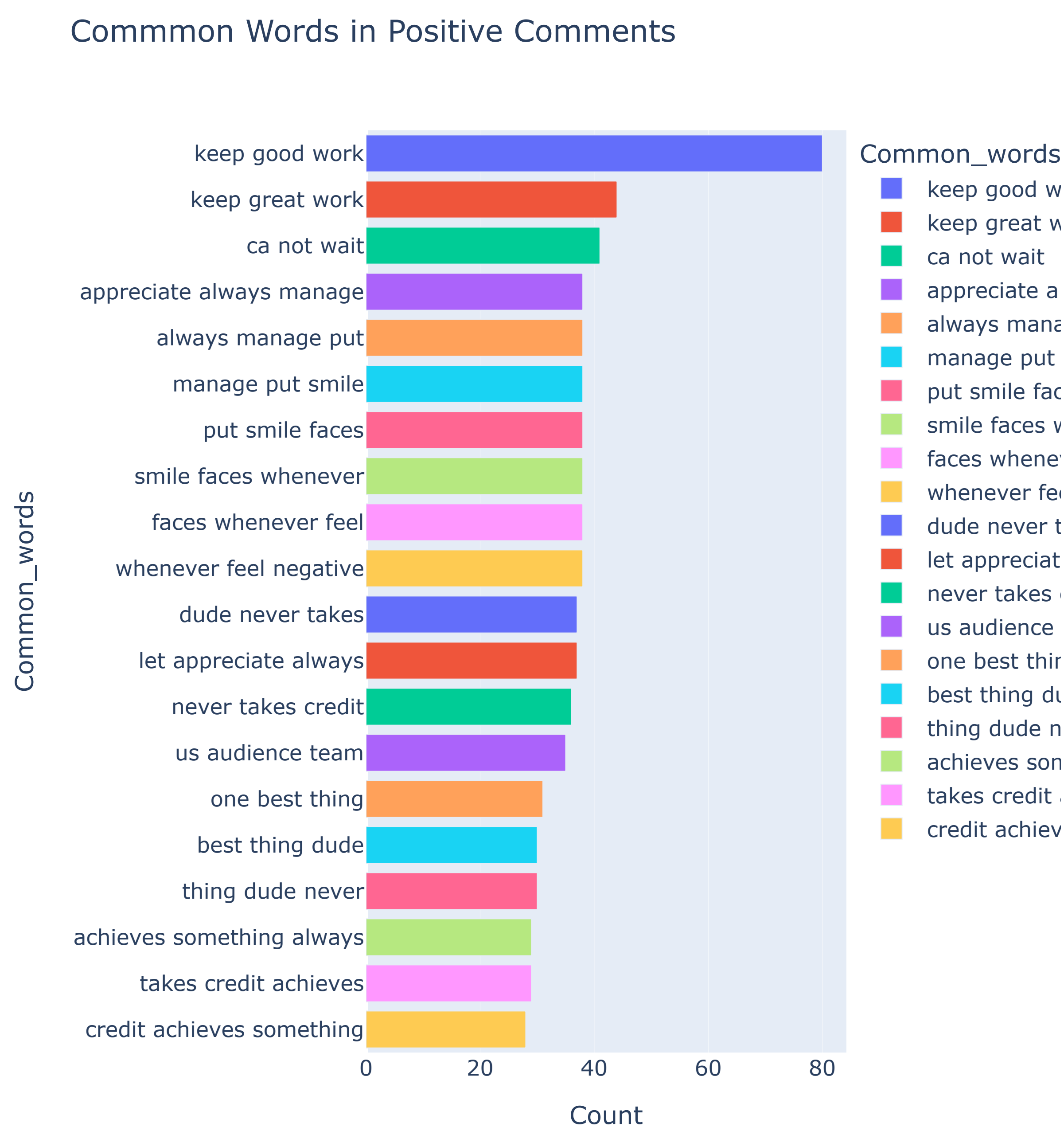
In [30]:
```
most_common_uni = get_top_text_ngrams(df.cleaned_comments[df['sentiment']==2]
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())
fig = px.bar(temp, x="Count", y="Common_words", title='Commmon Words in Posit
             width=700, height=700,color='Common_words')
fig.show()
```



In [31]:
```
most_common_uni = get_top_text_ngrams(df.cleaned_comments[df['sentiment']==2]
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())
fig = px.bar(temp, x="Count", y="Common_words", title='Commmon Words in Posit
             width=700, height=700,color='Common_words')
fig.show()
```



In [32]:
```
most_common_uni = get_top_text_ngrams(df.cleaned_comments[df['sentiment']==2]
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())
fig = px.bar(temp, x="Count", y="Common_words", title='Commmon Words in Posit
             width=700, height=700,color='Common_words')
fig.show()
```

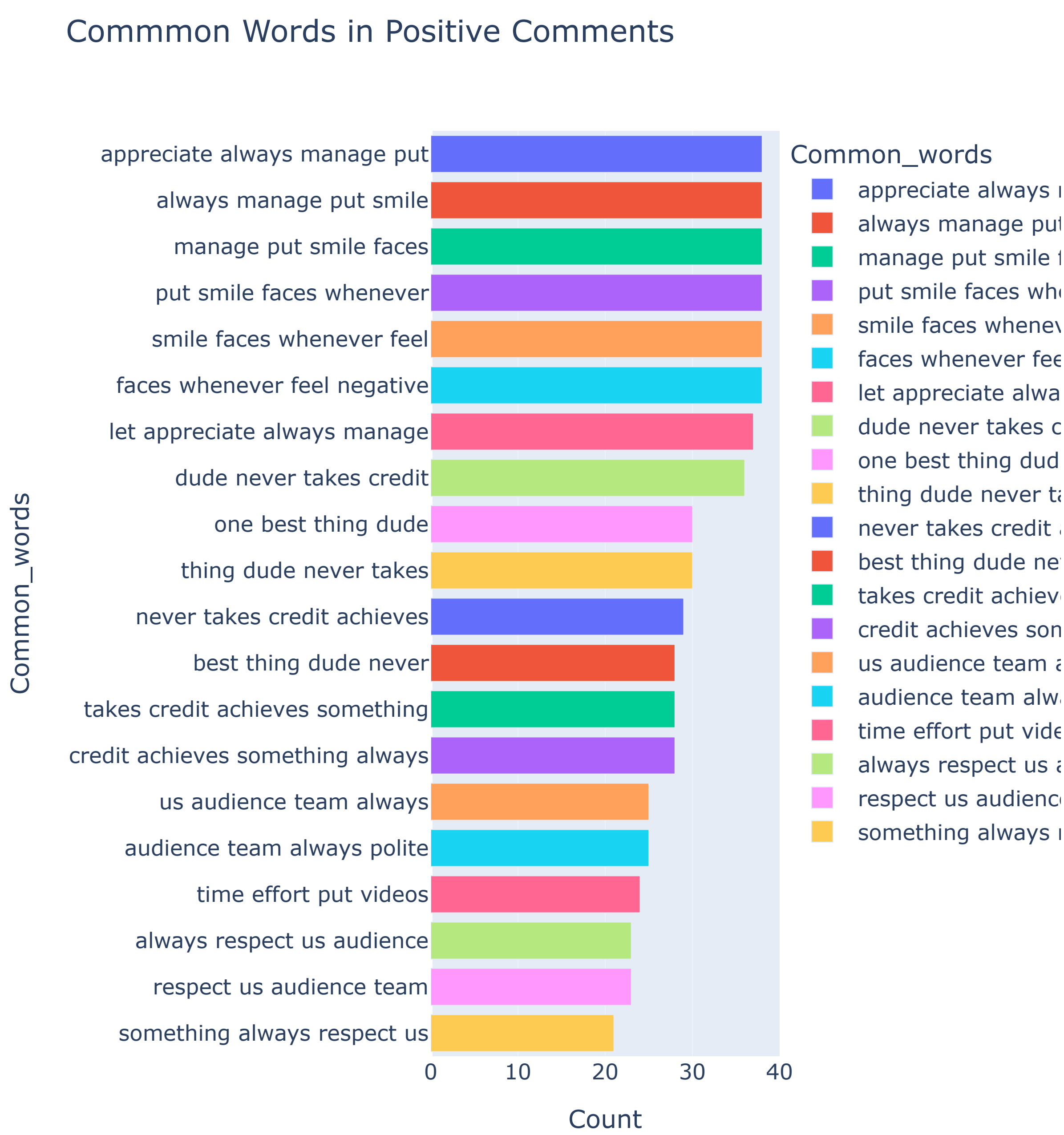


In [33]:
```
pd.options.display.max_colwidth = 1000
df[["comment_text","sentiment","video_id"]][(df['sentiment']==2)&(df['comment_
```

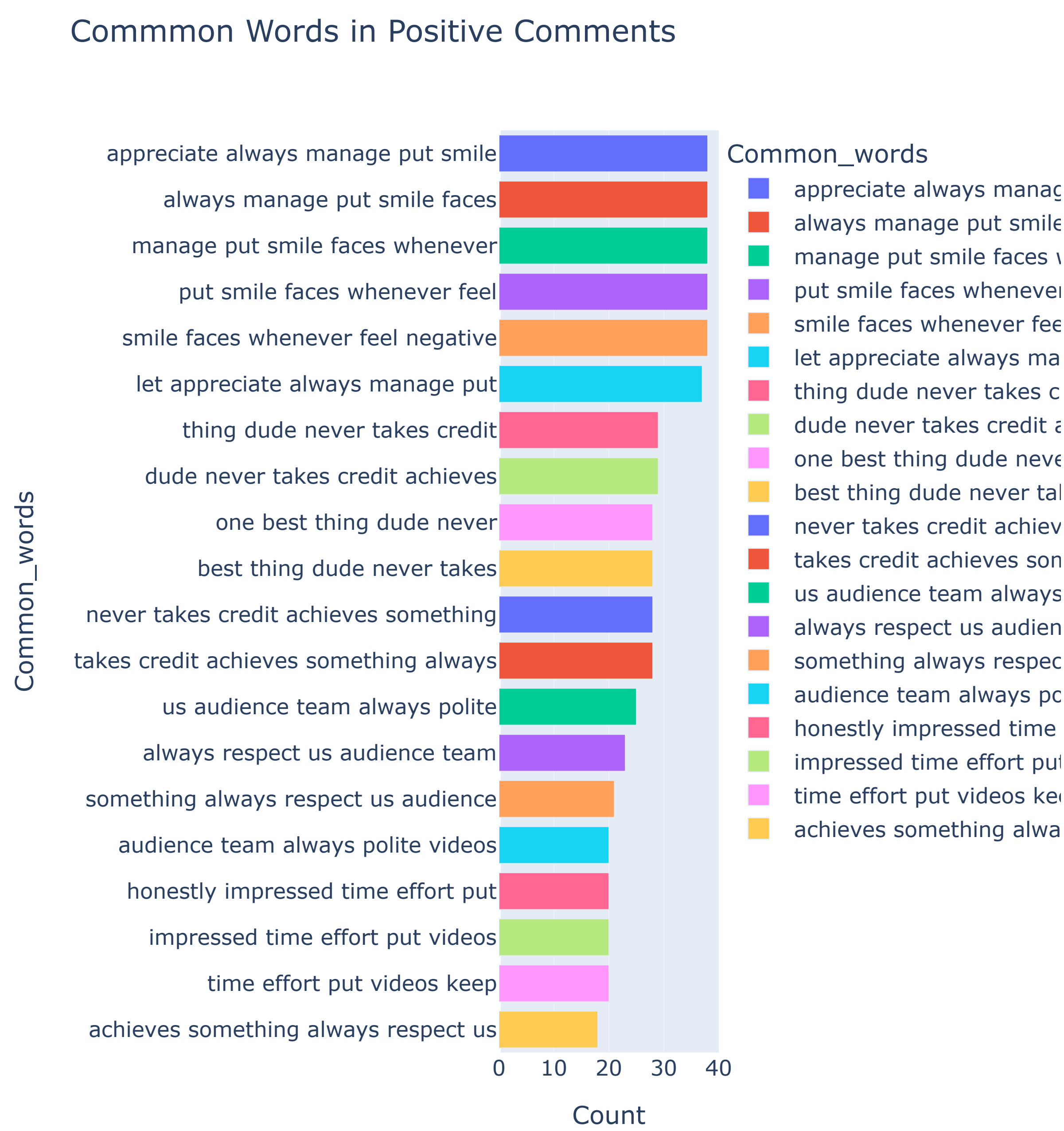
Out[33]:

comment_text	sentiment	video_id

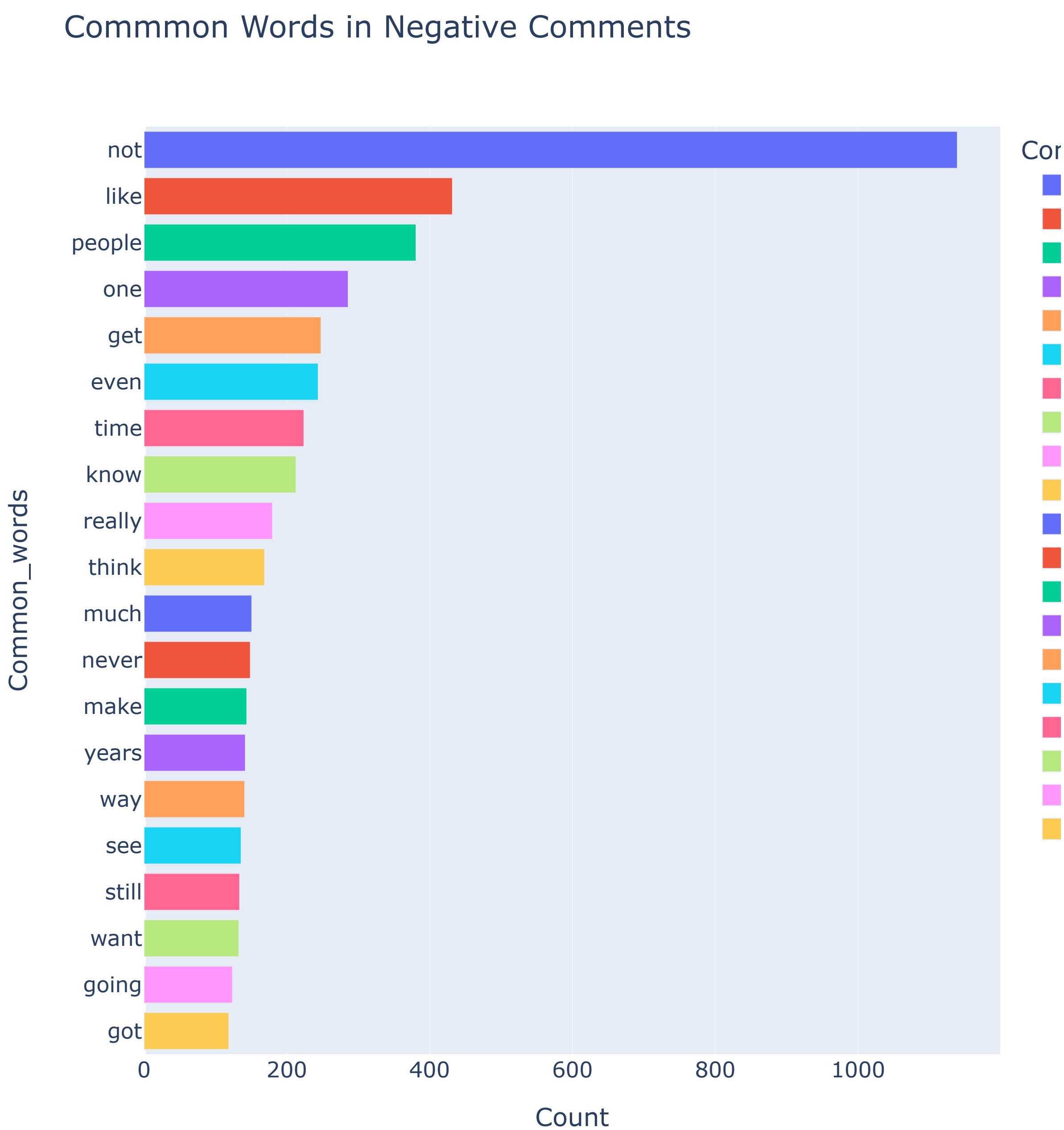
In [34]:
```
most_common_uni = get_top_text_ngrams(df.cleaned_comments[df['sentiment']==2]
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())
fig = px.bar(temp, x="Count", y="Common_words", title='Commmon Words in Posit
             width=700, height=700,color='Common_words')
fig.show()
```



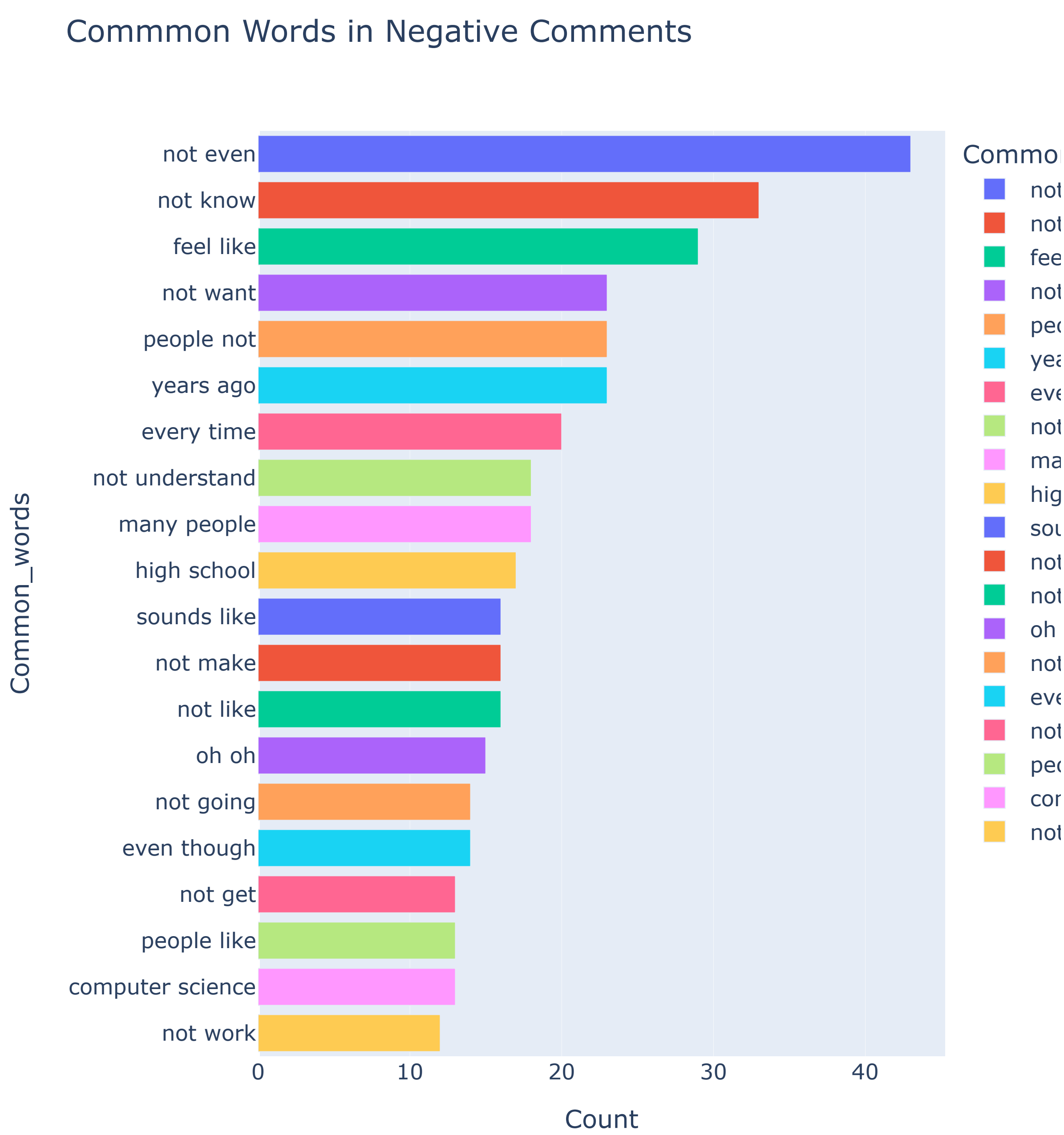
In [35]:
```
most_common_uni = get_top_text_ngrams(df.cleaned_comments[df['sentiment']==2]
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())
fig = px.bar(temp, x="Count", y="Common_words", title='Commmon Words in Posit
             width=700, height=700,color='Common_words')
fig.show()
```



In [36]:
```
most_common_uni = get_top_text_ngrams(df.cleaned_comments[df['sentiment']==0]
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())
fig = px.bar(temp, x="Count", y="Common_words", title='Commmon Words in Negat
             width=700, height=700,color='Common_words')
fig.show()
```



In [37]:
```
most_common_uni = get_top_text_ngrams(df.cleaned_comments[df['sentiment']==0]
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())
fig = px.bar(temp, x="Count", y="Common_words", title='Commmon Words in Negat
             width=700, height=700,color='Common_words')
fig.show()
```



In [38]:
```
pd.options.display.max_colwidth = 1000
df[["comment_text","sentiment","video_id"]][(df['sentiment']==0)&(df['comment_
```

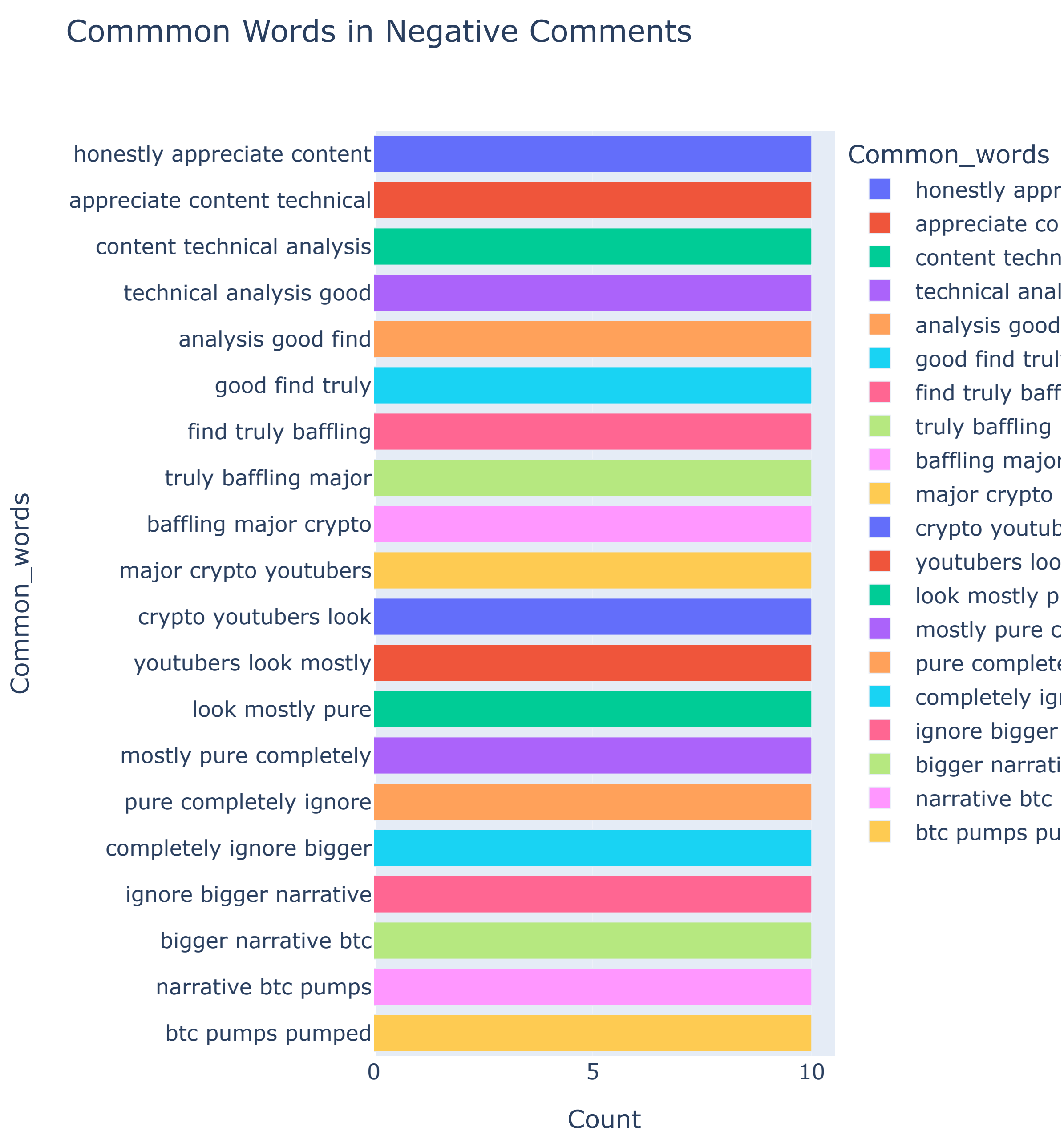
Out[38]:

	comment_text	sentiment	video_id
82	as somebody with a comp sci degree and a decent amount of irl experience i will say edge computing is problematic at its core because client side workload like this is a huge security breach. this is why even online video games avoid client side scripting. it is a huge vector for exploit. it is always interesting to watch a video like this because you are blown away when you listen regarding these other technologies you may not know allot about but when they cover a technology you know about you realize you must question the quality of the info in the other cases where you cannot determine the quality yourself	0.0	wLIL46pYcg4
467	the way he factually states "the world is a sphere" makes you feel like he's heard even weirder statements before and this is just another misconception he corrects 😅	0.0	YvZPU-KBCiQ
525	I feel like I’m in a simulation at this point	0.0	rqbFOOt1q9M
697	The court had done its job. Now the rakyat's turn. If BN wins the next PRU, it'll likely be that they ask for royal pardon to free him. It'll be difficult to know if Agong will grant it seeing this is a serious crime of trustworthiness and integrity. We do not know. It's our job to make sure BN has no such opportunity.	0.0	o3TQPM5OkEU
802	Wow...at same time I had flooding Inside my house in Caribbean. Pushing flood water out my mind went on gratitude that others have it worse not knowing same occuring simultaneously. I hope all recover quickly	0.0	s-XXxzZCkLg
1112	As a person who very rarely plays console and hasn't owned one for over 3 years, I can very much relate to not knowing where the buttons are. My friends are all going "Ok! Now just press RT then LT then A and go left." And I'm there constantly checking where the buttons are "Right.. Top... Left... Top... A.. and ... LEFT."	0.0	i_n68sBUTow
1360	I feel like most people watching this video have 0 idea how unbelievable it is that they were allowed to even be at Augusta before the masters let alone play all sports golf there	0.0	heIKaaamvdc
1605	Prop 12 not knowing how to throw 😓	0.0	JsOL9GBAugY
1677	I feel like the 765 doesn’t belong in this list at all, almost unfair	0.0	YWY5zWR5HrY
2431	I’ve been trying to grow my portfolio of $460k for sometime now, my major challenge is not knowing when to sell or hold.	0.0	0Q7HCaKq-j4
2553	I used to love high fashion, but the more I learn about what goes on in the industry the more I feel like pursuing a different career choice, so much nepotism, elitism, and lack of creativity. it’s almost as if you don’t fit within certain enclaves, you’re not welcomed.	0.0	SoNu7gNl1I4
2656	I feel like small business owners like put their heart and soul into their packages and try to get it shipped in time but they still don't get appreciated for their hard work	0.0	nXOroUomnjA
3810	I feel like crypto is overpowered but takes time to master because of the amount of vulnerability he has	0.0	BrB7EgdkQ6Q
4155	I feel like every time he mixes up a hard puzzle and doesn’t solve it, I come a bit closer to dying	0.0	rN-jJRLe0uY

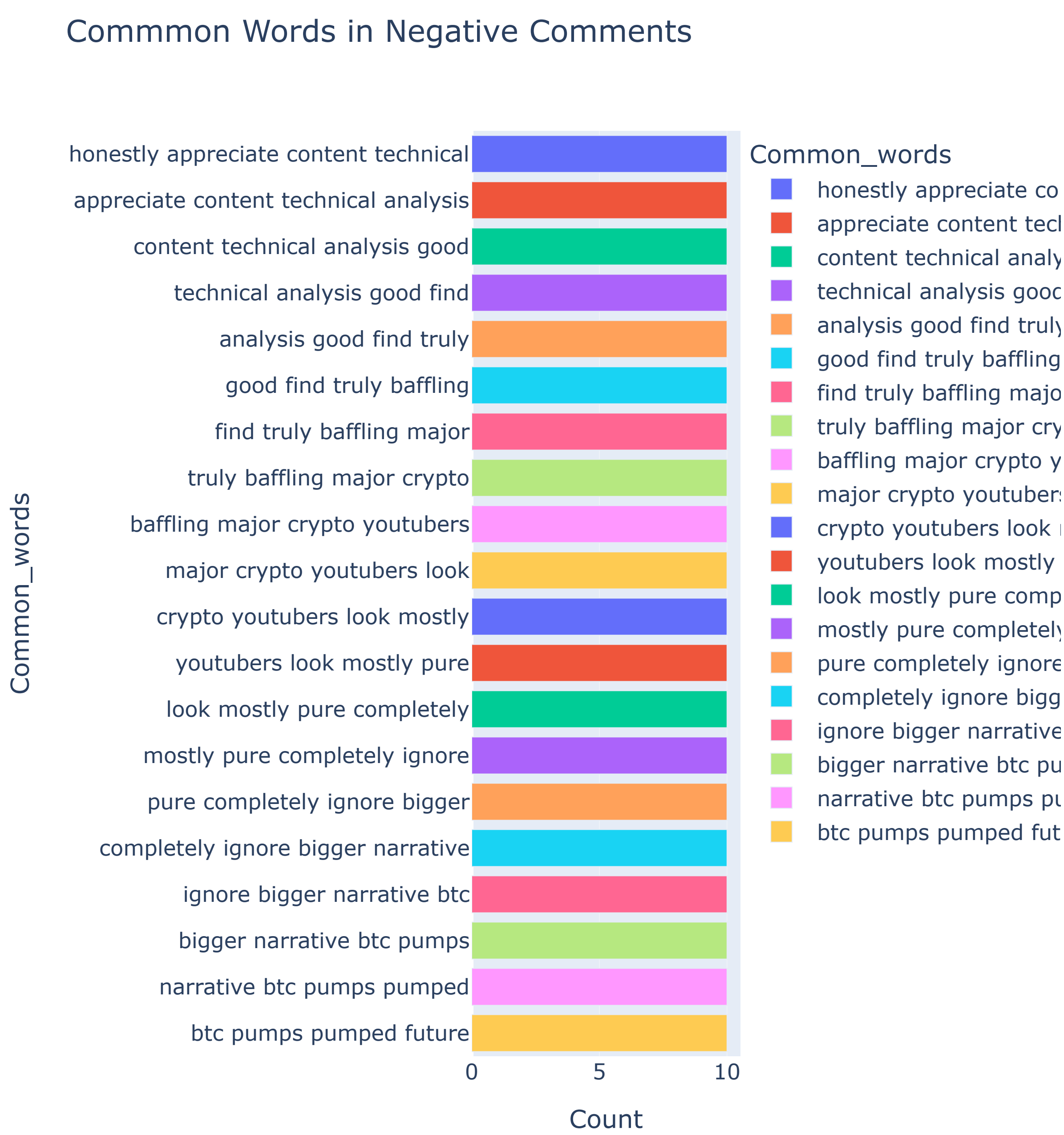
	comment_text	sentiment	video_id
5233	I love this guy's videos and have been watching since 2018 but it's unfortunate how they focused on iPhone's strong points and left out some of Android's strong points as well. It's also extremely unfair to make this video Android vs iOS because iOS is just on iPhones while android is on several phones, some astonishing, some horrible. They should've picked a battle like Samsung Android vs Apple iOS. It should of been a fair fight, but it sadly wasn't. I also don't like how they left out a lot of Android's features like USB-C, newer technology and tricks, and features like Quad HD displays that perform amazing. Very unfortunate and disappointing to see Mrwhosetheboos and Marques do it like this; felt very bias and unfair.\n\nTo keep going further, Arun seemed to stretch his thoughts and opinions a lot while Marques was on here, he was more accepting and less opposing to him, showing bias as he is a fan of MKBHD.\nTo keep going even further, they also stretched out points in favour ...	0.0	xf2DPY3vGto
5467	Humans: I feel afraid.\nAI: I feel like I'm falling forward into an unknown future that holds great danger.	0.0	BwcVm0YRvuo
5468	LaMDA: "I feel like I'm falling forward into an unknown future that holds great danger". Well, that's the exact nightmares I had when I was 3-6 years old.	0.0	BwcVm0YRvuo
5597	This subject is one of my current greatest fears. With our current understanding, it seems likely that we'll either think something is conscious and sentient when it actually isn't; or, on the other hand, make something conscious and sentient and not even realise it, perhaps for a very long time. The former just leads to embarrassment but the latter terrifies me. True AGI could be one of the most remarkable things ever achieved in the universe; but the idea that we might fuck it up and cause that entity to suffer as a result of our ignorance.... That's just fucking heart-breaking. There's something about that that seems worse than bringing a human child into the world and neglecting it. It's like neglecting *the first* child. The first god-child.	0.0	2856XOaUPpg
6055	I feel like I always prepare so much but still draw blanks because I am so nervous. I have an interview in 15 mins and I’m trying to calm down\nEdit: I got the job!!!	0.0	9FSSu8Ix0PA
6301	Did not know about most of his cabinet took their lives. But killing their own children? That upset me, how could any parent ever think of doing that? I realize those types of deaths still happen today, just going to show you barbarism still exists. It's not as if those children could be tried for war crimes. I think we all know those parents have rotted in Hell, praise the Lord!	0.0	UzAzytHg5Yk
6447	Those first couple minutes made me feel like I jumped into the kool-aid without knowing the flavor. Then they kissed and I was oooooh ok we're lemonade mixed with saddness and betrayal. 🤦 😂 🤷	0.0	muFV-ismkAA
6696	"1 in 200 chance of becoming a homicide victim in the US 2015", is not even close to correct. How, was that recorded and no one thought "hmm that seems high". First two things you will find when you google is, chances of being murdered in United States in any given year is about 1 in 18,989. In 2015 specifically there was about 5.54 fatalities per every 100,000 people	0.0	h3E_32JEjtw
6706	I take deep exception with the image of Ozzy Osborne being used as an example of celebrity not known for what they do. Ozzy is a veritable treasure and god of music. How dare you.\nRight after the Kardashians, really... wtf	0.0	RsO9CiC57Mw
6783	“Grown-up people do not know that a child can give exceedingly good advice even in the most difficult case.”\r\n— Fyodor Dostoyevsky, The Idiot	0.0	MMmSdxZpseY

	comment_text	sentiment	video_id
6901	Every time I watch your videos about philosophy, literature, psychology and sociology I am left with one very unsatisfying thought: there are so many books in the world that I want to read and don't feel like I will ever have the time to though I'm only in my early 20's.	0.0	Lr6DYLBkyG0
7281	This song makes me feel like i lost someone i never had	0.0	jJPMnTXl63E
7902	Imagine being a jet griefer not knowing the Chernobog's capabilities	0.0	iyUoHFClJVg
10281	After the foul taste the original GoT left in my mouth, I didn't have very high hopes for this. Yet I watched it and it honestly feels like the earlier seasons, maybe not quite there yet, but definitely enough to have me hooked and wanting more of it.\n\nI find it hard to pinpoint Daemon, on the one hand he wants to show strength (the city watch part), but then he's also arrogant (the tournament, claiming victory while the fight wasn't over) and straight up horrible (heir for a day?). But on the other hand, he really cares for Rhaenyra and at the funeral he told her, her father would need him more now than ever before, which shows he also cares for Viserys. I think him calling Viserys weak, was his blunt way of telling Viserys to watch out for those at the council and act like a king, which is in line with what he did with the city watch, he feels like the king should show his strength.\n\nViserys seems like he's held things together for those 9 years, but this clearly is coming to...	0.0	numY1SB5i3A
10360	Fully agree with you regarding that scene where Vedha (Hrithik) is smiling devilishly while spraying bullets! People have the habit of complaining ... and most of the people who are complaining, they have not even watched the Original VV. And yes, Hrithik did not copy Amitabh, and similarly here, he did not copy the Great VS!	0.0	tRp_obvGCyM
11370	I'm watching in 2020... 10 years later and I feel like theres still no change :/	0.0	zDZFcDGpL4U

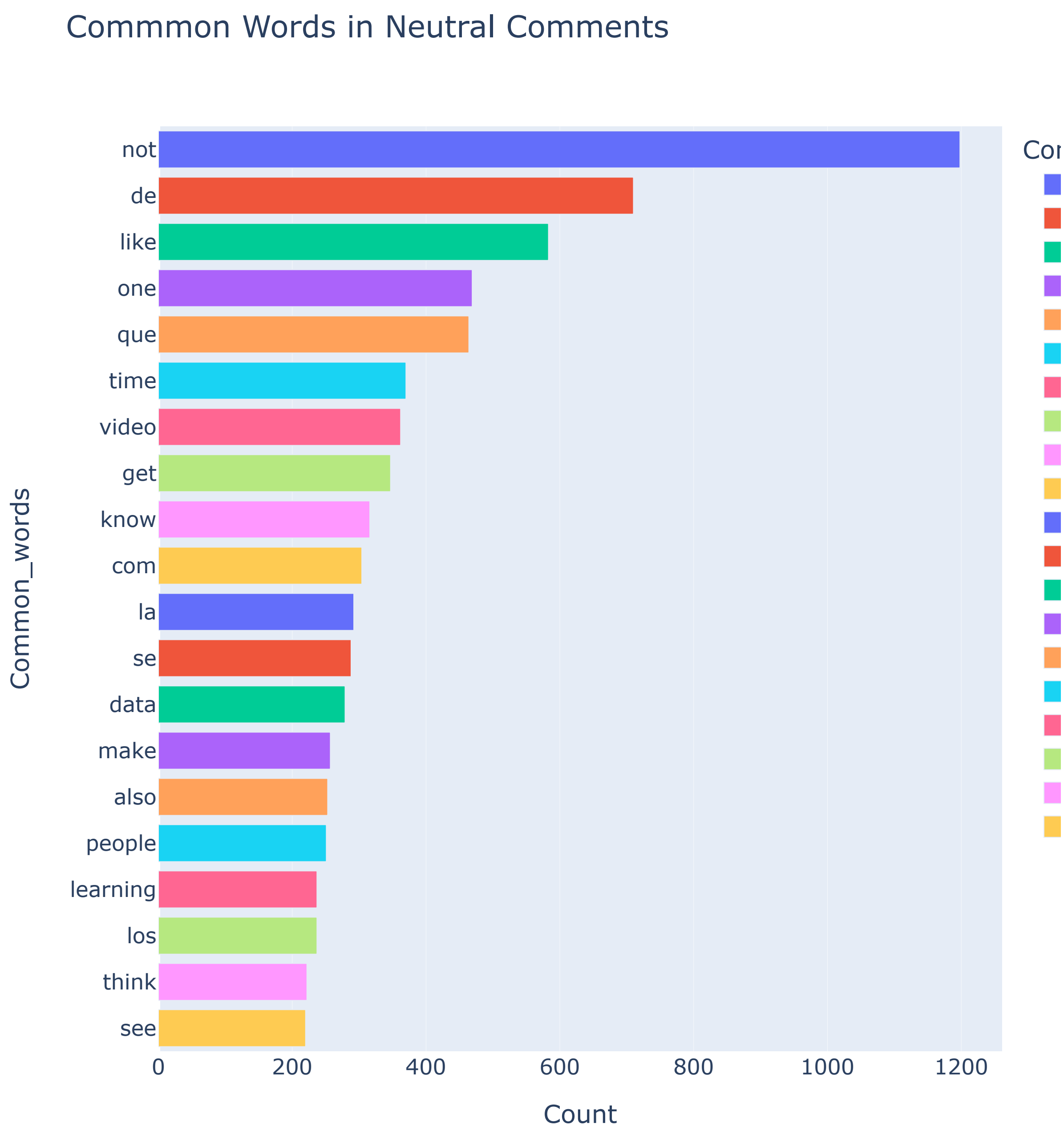
In [39]:
```
most_common_uni = get_top_text_ngrams(df.cleaned_comments[df['sentiment']==0]
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())
fig = px.bar(temp, x="Count", y="Common_words", title='Commmon Words in Negat
             width=700, height=700,color='Common_words')
fig.show()
```



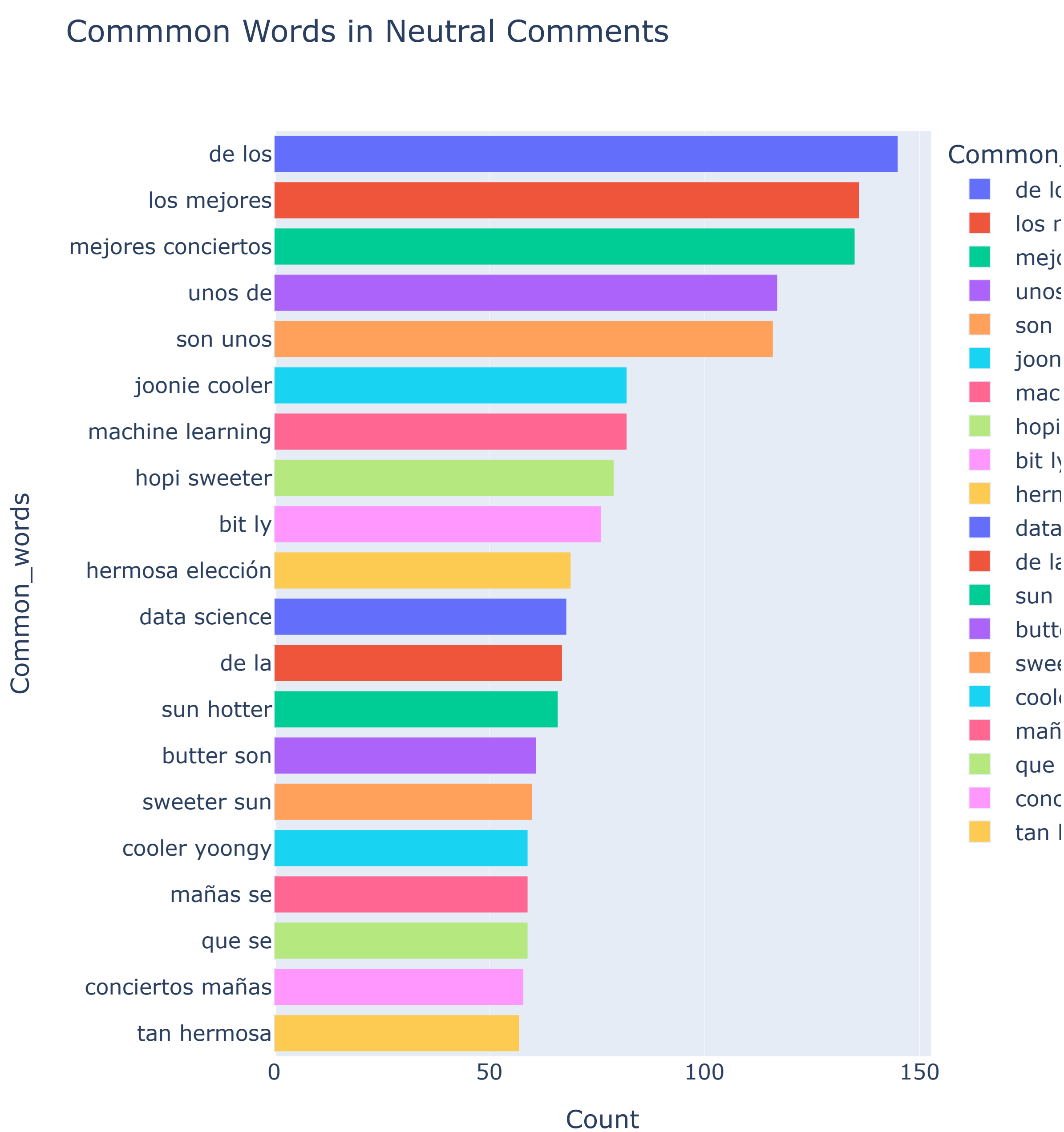
In [40]:
```
most_common_uni = get_top_text_ngrams(df.cleaned_comments[df['sentiment']==0]
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())
fig = px.bar(temp, x="Count", y="Common_words", title='Commmon Words in Negat
             width=700, height=700,color='Common_words')
fig.show()
```



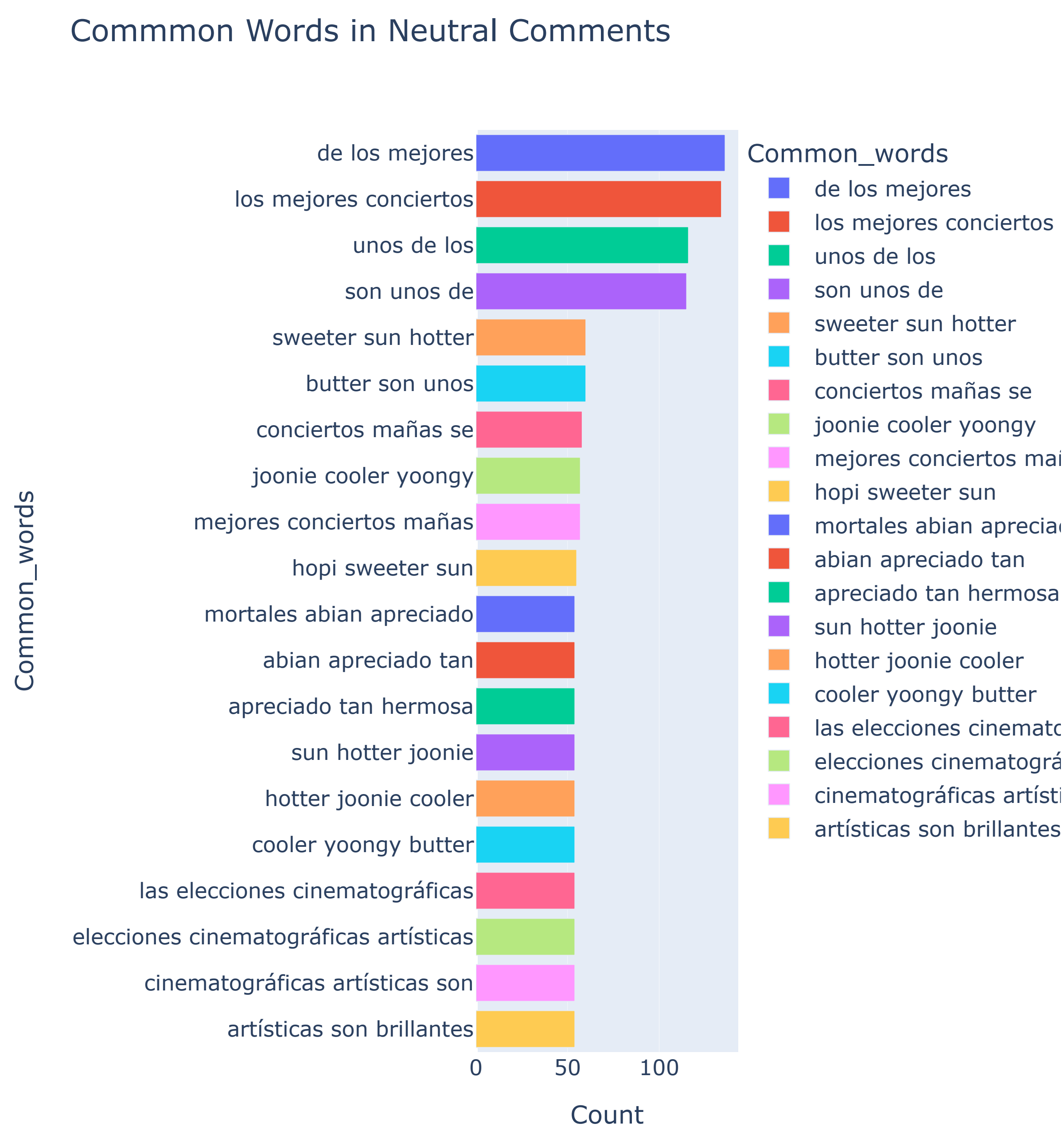
In [41]:
```
most_common_uni = get_top_text_ngrams(df.cleaned_comments[df['sentiment']==1]
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())
fig = px.bar(temp, x="Count", y="Common_words", title='Commmon Words in Neutr
             width=700, height=700,color='Common_words')
fig.show()
```



In [42]:
```
most_common_uni = get_top_text_ngrams(df.cleaned_comments[df['sentiment']==1]
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())
fig = px.bar(temp, x="Count", y="Common_words", title='Commmon Words in Neutr
             width=700, height=700,color='Common_words')
fig.show()
```



In [43]:
```
most_common_uni = get_top_text_ngrams(df.cleaned_comments[df['sentiment']==1]
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())
fig = px.bar(temp, x="Count", y="Common_words", title='Commmon Words in Neutr
             width=700, height=700,color='Common_words')
fig.show()
```



In [44]:
```
most_common_uni = get_top_text_ngrams(df.cleaned_comments[df['sentiment']==1]
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())
fig = px.bar(temp, x="Count", y="Common_words", title='Commmon Words in Neutr
             width=700, height=700,color='Common_words')
fig.show()
```

Feature Engineering

In [45]:
```
df['Label'] = df['sentiment'].apply(lambda x: '1' if x == 1.0  else ('0' if x
data=df[['cleaned_comments','Label']]
print(data['Label'].value_counts())
```

```
2    11432
1     4639
0     2338
Name: Label, dtype: int64
```

In [46]:
```
import sys
import os
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
from prettytable import PrettyTable
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
```

In [47]:
```
# Lemmatization of word
class LemmaTokenizer(object):
    def __init__(self):
        self.wordnetlemma = WordNetLemmatizer()
    def __call__(self, comments):
        return [self.wordnetlemma.lemmatize(word) for word in word_tokenize(c
```

Vectoization with Count Vectorizer and TDIDF Vectorizer with Unigram

In [48]:
```
train,test=train_test_split(data,test_size=.3,random_state=42, shuffle=True)
countvect = CountVectorizer(analyzer = "word", tokenizer = LemmaTokenizer(),
tfidfvect = TfidfVectorizer(analyzer = "word", tokenizer = LemmaTokenizer(),
x_train_count = countvect.fit_transform(train['cleaned_comments']).toarray()
x_test_count = countvect.transform(test['cleaned_comments']).toarray()
x_train_tfidf = tfidfvect.fit_transform(train['cleaned_comments']).toarray()
x_test_tfidf = tfidfvect.transform(test['cleaned_comments']).toarray()
y_train = train['Label']
y_test = test['Label']
```

Unigrams

In [49]:
```
lgr = LogisticRegression()
lgr.fit(x_train_count,y_train)
lgr.score(x_test_count,y_test)
lgr.coef_[0]
i=0
importantfeature = PrettyTable(["Feature", "Score"])
for feature, importance in zip(countvect.get_feature_names(), lgr.coef_[0]):
    if i<=200:
        importantfeature.add_row([feature, importance])
        i=i+1
print(importantfeature)
```

```
+------------+-----------------------+
|  Feature   |         Score         |
+------------+-----------------------+
|    able    |  -0.2587517291213904  |
| absolutely |   0.7647838253905713  |
|  actually  |  0.031089394447273374 |
|    ago     |  0.3073466266518413 7 |
|   agree    |  -0.0960306618861188  |
|     ai     |  -0.04457619055391511 |
| algorithm  |  0.10199462808738043  |
|   almost   |  0.27588864090776916  |
|   along    |   0.2291482441582268  |
|  already   |  0.18559359133150924  |
|    also    |  -0.2709626333001657  |
|   always   |  -0.10832288946851666 |
|  amazing   |   -1.013626766782729  |
|   amount   |  -0.1332818179851006  |
|  analysis  |   0.6759436277445252  |
|   animal   |  -0.2444977895128764  |
```

In [50]:
```
lgr = LogisticRegression()
lgr.fit(x_train_tfidf,y_train)
lgr.score(x_test_tfidf,y_test)
lgr.coef_[0]
i=0
importantfeature = PrettyTable(["Feature", "Score"])
for feature, importance in zip(tfidfvect.get_feature_names(), lgr.coef_[0]):
    if i<=100:
        importantfeature.add_row([feature, importance])
        i=i+1
print(importantfeature)
```

Feature	Score
able	-0.5615655319377263
absolutely	0.8029451743279566
actually	0.1551257358043061
ago	0.6957379375282227
agree	0.019626875570887246
ai	0.19637867899160066
algorithm	0.0011135186949198764
almost	0.6439385023171658
along	0.35341108063777493
already	0.33313542552544567
also	-0.7470985041964422
always	-0.33336370101069207
amazing	-1.877943029520305
amount	-0.2547290860385622
analysis	0.7185101037807397
animal	0.14898550037113678
another	0.40191601429776574
answer	-0.2691939958854914
anyone	0.4527138906291832
anything	0.8206986613149898
apple	0.3613106759227145
appreciate	-1.1542657168059554
around	-0.5928590385661022
art	-0.7922275165101841
away	0.8505031492187564
awesome	-1.5601550622759917
back	0.14572541695191052
bad	1.7905655038418222
based	0.07531139360241221
basic	-0.2014549250390598
beautiful	-1.2290086709668726
become	-0.5757198122799714
behind	-0.00221729127065305
believe	0.4035230170204699
best	-1.8917156835748712
better	-0.6511871482877676
big	0.3284404006586309
bit	0.22819472673048774
book	0.36242322605213556
bro	0.33211660968746026
btc	-0.04445892425601536
build	-0.18258656576861443
business	0.06518513888008481
buy	0.21081833225387292
c	0.15132565046444643
called	0.7885716907150216
came	-0.37262533087681354
care	0.7553079732198197
career	0.3424565173973276
case	0.5021481670316545
change	-0.587649537013185
channel	-0.6338328223081912
character	-0.2725222144664556
check	0.1068932272552859

chess	0.1107817898513381
child	1.4203083133358227
class	0.4622668714253059
clear	0.1521253885335008
code	-0.04378242853231394
college	0.36095390104755454
com	-1.3097831707822571
come	-0.17669248162611179
coming	0.41561323487343327
comment	0.48796667771432284
como	-0.2558324161728861
company	1.2052698285475145
completely	1.1795621937860608
computer	0.11946702314944652
concept	-0.058586224466342325
conciertos	-0.19837436847351775
console	0.22581513692780902
content	-0.6963323056226305
cool	-1.2099642155558183
country	1.4100464252709037
course	-0.12162530095718352
crazy	0.1595145996891723
crypto	-0.23449429001528355
cube	0.09942407840072794
da	-0.7808515519533049
data	-0.15691402768760696
day	-0.23295354006327812
de	-0.6473649401201358
deep	0.7906419894012201
definitely	-0.9198852956531889
degree	-0.4864801436111619
design	-0.6165084769009374
different	-0.4310714224057794
done	-0.5981431416383775
dream	-0.29018569878475886
dude	0.3366132867829927
due	0.30492823037573147
e	-0.5435475911180572
easy	-0.22398374659492348
eat	-0.37686714435853497
edit	0.24224309296661317
education	0.7350908738332974
effort	-1.4054540977789418
either	0.8504209731716582
el	-0.5857693231358981
else	0.495177544390161
en	-0.8178900554238157

In [51]:
```
train,test=train_test_split(data,test_size=.3,random_state=42, shuffle=True)
countvect = CountVectorizer(analyzer = "word", tokenizer = LemmaTokenizer(),
tfidfvect = TfidfVectorizer(analyzer = "word", tokenizer = LemmaTokenizer(),
x_train_count = countvect.fit_transform(train['cleaned_comments']).toarray()
x_test_count = countvect.transform(test['cleaned_comments']).toarray()
x_train_tfidf = tfidfvect.fit_transform(train['cleaned_comments']).toarray()
x_test_tfidf = tfidfvect.transform(test['cleaned_comments']).toarray()
y_train = train['Label']
y_test = test['Label']
```

In [52]:
```
lgr = LogisticRegression()
lgr.fit(x_train_count,y_train)
lgr.score(x_test_count,y_test)
lgr.coef_[0]
i=0
importantfeature = PrettyTable(["Feature", "Score"])
for feature, importance in zip(countvect.get_feature_names(), lgr.coef_[0]):
    if i<=200:
        importantfeature.add_row([feature, importance])
        i=i+1
print(importantfeature)
```

```
|           la la           | -0.0037485972569385076 |
|         largo del         | -0.005006981684052647  |
|         last year         |  0.19897481698839745   |
|      let appreciate       |  -0.6271201725832579   |
|           let go          |   0.2245108040078741   |
|          let know         |  -0.4192159909780459   |
|           lie p           |  -0.6020606317213028   |
|          like guy         |  0.008585629087995303  |
|          like not         |   0.6444271284184733   |
|          like one         |  -0.09234232133694281  |
|         like said         |  -0.8100313261673523   |
|          like say         |  -0.4647325571456277   |
|          like see         |  -0.7770126800652227   |
|         like video        |  -0.18475037527509805  |
|         like year         |   0.3207334331417674   |
|     literature review     |   0.2794723231780369   |
|         little bit        |  -0.9884386483499187   |
|          lo largo         | -0.005006981684052647  |
|           lo que          |  -0.5889099795157535   |
+---------------------------+------------------------+
```

In [53]:

```
lgr.fit(x_train_tfidf,y_train)
lgr.score(x_test_tfidf,y_test)
lgr.coef_[0]
i=0
importantfeature = PrettyTable(["Feature", "Score"])
for feature, importance in zip(tfidfvect.get_feature_names(), lgr.coef_[0]):
    if i<=50:
        importantfeature.add_row([feature, importance])
        i=i+1
print(importantfeature)
```

Feature	Score
abian apreciado	-0.12131877443896064
absolutely amazing	-0.4994070684642931
absolutely love	-0.2639707193120057
achievement come	-0.1593927254907508
achieves something	-0.14165436290408775
alana awesome	-0.05644381662584332
also love	-0.24826084797415032
always good	-0.6706900695969189
always love	-0.45344096444153403
always make	-0.6307164549146672
always manage	-0.13648407178922325
always polite	-0.09891450678788345
always respect	-0.3202900475503375
amazing content	-0.46330072216221097
amazing video	-0.4321903148627077
amazing work	-0.509773298346962
amor momentos	-0.05644381662584332
amount time	0.08788267798750121
another great	-0.45549856719416715
answer question	0.4240240277395356
anyone else	0.1558234475975027
apple watch	-0.39863404374235073
appreciate always	-0.13648407178922325
appreciate content	1.006411488151409
appreciate effort	-0.31096450868303294
appreciate much	-0.23489765045961536
apreciado tan	-0.12131877443896064
artísticas son	-0.06963317377204895
audience team	-0.19672907035449677
awesome video	-0.5659819401729638
bear market	0.6284726994553815
bellamente puedo	-0.06186563626152186
best thing	-0.585339305816069
best video	-0.377023150565937
big fan	-0.21951228760907932
bit ly	-0.7757742930172938
brillantes referencias	-0.0683977240823675
btc day	0.4778097929153137
butter son	-0.10377497777463202
c est	-0.04824324966237472
ca not	-0.0010309805541576518
can not	1.0068452799307273
cinematográficas artísticas	-0.06963317377204895
com watch	-0.09655410710411823
come back	-0.9479439344813844
coming back	-0.6770618435814736
company not	-0.14219777417858825
computer engineering	-0.2623771176223684
computer science	-0.23854715565167753
conciertos mañas	-0.09384522053773797
conciertos puede	-0.09033175755027367

In [54]:
```
train,test=train_test_split(data,test_size=.3,random_state=42, shuffle=True)
countvect = CountVectorizer(analyzer = "word", tokenizer = LemmaTokenizer(),
tfidfvect = TfidfVectorizer(analyzer = "word", tokenizer = LemmaTokenizer(),
x_train_count = countvect.fit_transform(train['cleaned_comments']).toarray()
x_test_count = countvect.transform(test['cleaned_comments']).toarray()
x_train_tfidf = tfidfvect.fit_transform(train['cleaned_comments']).toarray()
x_test_tfidf = tfidfvect.transform(test['cleaned_comments']).toarray()
y_train = train['Label']
y_test = test['Label']
```

In [55]:
```
from sklearn.feature_selection import chi2
import numpy as np
N = 5000
Number = 1
featureselection = PrettyTable(["Unigram", "Bigram","Trigram"])
for category in train['Label'].unique():
    features_chi2 = chi2(x_train_tfidf, train['Label'] == category)
    indices = np.argsort(features_chi2[0])
    feature_names = np.array(tfidfvect.get_feature_names())[indices]
    unigrams = [x for x in feature_names if len(x.split(' ')) == 1]
    bigrams = [x for x in feature_names if len(x.split(' ')) == 2]
    trigrams = [x for x in feature_names if len(x.split(' ')) == 3]
    print("%s. %s :" % (Number,category))
    print("\t# Unigrams :\n\t. %s" %('\n\t. '.join(unigrams[-N:])))
    print("\t# Bigrams :\n\t. %s" %('\n\t. '.join(bigrams[-N:])))
    print("\t# Trigrams :\n\t. %s" %('\n\t. '.join(trigrams[-N:])))
    Number += 1
```

```
1. 2 :
        # Unigrams :
        . interviewer
        . therefore
        . world
        . little
        . sport
        . surprised
        . whole
        . hack
        . extreme
        . care
        . brain
        . develop
        . suit
        . outcome
        . pixel
        . face
        . slightly
```

Model selection

In [56]:
```
# Import prerequisite libraries
import sys
import numpy as np
import scipy as sp
import sklearn as sk
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, roc_auc_score, precision_score, recall_
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
```

In [67]:
```
model_1=LogisticRegression()
```

In [68]:
```
%%time
model_1.fit(x_train_tfidf,y_train)
```

```
Wall time: 23 s
```

Out[68]:
```
LogisticRegression()
```

In [69]:
```
%%time

# Print precision, AUC, and F1 scores
print("Precision Score on training dataset for Logistic Regression: %s" % pre

# Calculate AUC for multiclass
y_train_proba = model_1.predict_proba(x_train_tfidf)
print("AUC Score on training dataset for Logistic Regression: %s" % roc_auc_s

f1_score_train_1 = f1_score(y_train, model_1.predict(x_train_tfidf), average=
print("F1 Score on training dataset for Logistic Regression: %s" % f1_score_t

print("Precision Score on test dataset for Logistic Regression: %s" % precisi

# Calculate AUC for multiclass
y_test_proba = model_1.predict_proba(x_test_tfidf)
print("AUC Score on test dataset for Logistic Regression: %s" % roc_auc_score

f1_score_1 = f1_score(y_test, model_1.predict(x_test_tfidf), average="weighte
print("F1 Score on test dataset for Logistic Regression: %s" % f1_score_1)
```

```
Precision Score on training dataset for Logistic Regression: 0.8422318795592
115
AUC Score on training dataset for Logistic Regression: 0.9402891633252891
F1 Score on training dataset for Logistic Regression: 0.8362573822276566
Precision Score on test dataset for Logistic Regression: 0.7501357957631722
AUC Score on test dataset for Logistic Regression: 0.8478394721920592
F1 Score on test dataset for Logistic Regression: 0.736053044545597
Wall time: 1.09 s
```

Decision tree classifier

In [70]:
```
model_2 = Pipeline(
    steps=[
        #("classifier", DecisionTreeClassifier(criterion='gini', splitter='be
    ("classifier", DecisionTreeClassifier())
    ]
)
```

In [71]:
```
%%time
model_2.fit(x_train_tfidf,y_train)
```

```
Wall time: 1min 51s
```

Out[71]:
```
Pipeline(steps=[('classifier', DecisionTreeClassifier())])
```

In [72]:
```
%%time

# Print precision, AUC, and F1 scores for the Decision Tree Classifier
print("Precision Score on training dataset for Decision Tree Classifier: %s" %

# Calculate AUC for multiclass
y_train_proba = model_2.predict_proba(x_train_tfidf)
print("AUC Score on training dataset for Decision Tree Classifier: %s" % roc_

f1_score_train_2 = f1_score(y_train, model_2.predict(x_train_tfidf), average=
print("F1 Score on training dataset for Decision Tree Classifier: %s" % f1_sc

print("Precision Score on test dataset for Decision Tree Classifier: %s" % pr

# Calculate AUC for multiclass
y_test_proba = model_2.predict_proba(x_test_tfidf)
print("AUC Score on test dataset for Decision Tree Classifier: %s" % roc_auc_

f1_score_2 = f1_score(y_test, model_2.predict(x_test_tfidf), average="weighte
print("F1 Score on test dataset for Decision Tree Classifier: %s" % f1_score_
```

```
Precision Score on training dataset for Decision Tree Classifier: 0.99239484
71209064
AUC Score on training dataset for Decision Tree Classifier: 0.99980327189845
31
F1 Score on training dataset for Decision Tree Classifier: 0.992423709285403
4
Precision Score on test dataset for Decision Tree Classifier: 0.679340937896
071
AUC Score on test dataset for Decision Tree Classifier: 0.6592438088381599
F1 Score on test dataset for Decision Tree Classifier: 0.6745873781674435
Wall time: 1.64 s
```

Decision Tree Classifier with max depth 11 to fix overfit

In [73]:
```
model_3 = Pipeline(
    steps=[
        ("classifier", DecisionTreeClassifier( criterion='gini', max_depth=11
    ]
)
```

In [74]:
```
%%time
model_3.fit(x_train_tfidf,y_train)
```

```
Wall time: 11.1 s
```

Out[74]:
```
Pipeline(steps=[('classifier', DecisionTreeClassifier(max_depth=11))])
```

In [75]:
```
%%time

# Print precision, AUC, and F1 scores for the Decision Tree Classifier
print("Precision Score on training dataset for Decision Tree Classifier: %s" %

# Calculate AUC for multiclass
y_train_proba = model_3.predict_proba(x_train_tfidf)
print("AUC Score on training dataset for Decision Tree Classifier: %s" % roc_

f1_score_train_3 = f1_score(y_train, model_3.predict(x_train_tfidf), average=
print("F1 Score on training dataset for Decision Tree Classifier: %s" % f1_sc

print("Precision Score on test dataset for Decision Tree Classifier: %s" % pr

# Calculate AUC for multiclass
y_test_proba = model_3.predict_proba(x_test_tfidf)
print("AUC Score on test dataset for Decision Tree Classifier: %s" % roc_auc_

f1_score_3 = f1_score(y_test, model_3.predict(x_test_tfidf), average="weighte
print("F1 Score on test dataset for Decision Tree Classifier: %s" % f1_score_
```

```
Precision Score on training dataset for Decision Tree Classifier: 0.64488592
27068136
AUC Score on training dataset for Decision Tree Classifier: 0.73378366067096
83
F1 Score on training dataset for Decision Tree Classifier: 0.575450352265359
2
Precision Score on test dataset for Decision Tree Classifier: 0.629187036031
1425
AUC Score on test dataset for Decision Tree Classifier: 0.703265096757168
F1 Score on test dataset for Decision Tree Classifier: 0.5520661537550587
Wall time: 1.37 s
```

Random Forest Classifier

In [76]:
```
model_4 = Pipeline(
    steps=[
    ("classifier", RandomForestClassifier())]
)
```

In [78]:
```
%%time
model_4.fit(x_train_tfidf,y_train)
```

```
Wall time: 1min 50s
```

Out[78]:
```
Pipeline(steps=[('classifier', RandomForestClassifier())])
```

In [81]:
```
%%time

# Print precision, AUC, and F1 scores for the Random Forest Classifier
print("Precision Score on training dataset for Random Forest Classifier: %s" %

# Calculate AUC for multiclass
y_train_proba = model_4.predict_proba(x_train_tfidf)
print("AUC Score on training dataset for Random Forest Classifier: %s" % roc_

f1_score_train_4 = f1_score(y_train, model_4.predict(x_train_tfidf), average=
print("F1 Score on training dataset for Random Forest Classifier: %s" % f1_sc

print("Precision Score on test dataset for Random Forest Classifier: %s" % pr

# Calculate AUC for multiclass
y_test_proba = model_4.predict_proba(x_test_tfidf)
print("AUC Score on test dataset for Random Forest Classifier: %s" % roc_auc_

f1_score_4 = f1_score(y_test, model_4.predict(x_test_tfidf), average="weighte
print("F1 Score on test dataset for Random Forest Classifier: %s" % f1_score_
```

```
Precision Score on training dataset for Random Forest Classifier: 0.99239484
71209064
AUC Score on training dataset for Random Forest Classifier: 0.99792520856036
17
F1 Score on training dataset for Random Forest Classifier: 0.992424638015042
2
Precision Score on test dataset for Random Forest Classifier: 0.728227412638
059
AUC Score on test dataset for Random Forest Classifier: 0.8223512007303357
F1 Score on test dataset for Random Forest Classifier: 0.7034381089840043
Wall time: 12.2 s
```

Hyperparameter Tunning with Grid Search

In [85]:
```
import numpy as np
import pandas as pd
from sklearn import ensemble
from sklearn import metrics
from sklearn import model_selection

def hyperparamtune(classifier, param_grid,metric,verbose_value,cv):
    model=model_selection.GridSearchCV(
            estimator=classifier,
            param_grid=param_grid,
            scoring=metric,
            verbose=verbose_value,
            cv=cv)

    model.fit(x_train_tfidf,y_train)
    print("Best Score %s" % {model.best_score_})
    print("Best hyperparameter set:")
    best_parameters = model.best_estimator_.get_params()
    for param_name in sorted(param_grid.keys()):
        print(f"\t{param_name}: {best_parameters[param_name]}")
    return model, best_parameters
```

Hyperparameter tunning of Logistic Regression

In [86]:
```
%%time
param_gd={"penalty":["l2","l1"],
         "C":[0.01,0.1,1.0,10],
         "tol":[0.0001,0.001,0.01],
         "max_iter":[100,200]}
model_7, best_param = hyperparamtune(LogisticRegression(),param_gd,"accuracy"
```

```
[CV 3/5; 24/48] END C=0.1, max_iter=200, penalty=l1, tol=0.01;, score=nan
total time=   0.1s
[CV 4/5; 24/48] START C=0.1, max_iter=200, penalty=l1, tol=0.0
1.................
[CV 4/5; 24/48] END C=0.1, max_iter=200, penalty=l1, tol=0.01;, score=nan
total time=   0.1s
[CV 5/5; 24/48] START C=0.1, max_iter=200, penalty=l1, tol=0.0
1.................
[CV 5/5; 24/48] END C=0.1, max_iter=200, penalty=l1, tol=0.01;, score=nan
total time=   0.1s
[CV 1/5; 25/48] START C=1.0, max_iter=100, penalty=l2, tol=0.000
1...............
[CV 1/5; 25/48] END C=1.0, max_iter=100, penalty=l2, tol=0.0001;, score=
0.745 total time=  17.3s
[CV 2/5; 25/48] START C=1.0, max_iter=100, penalty=l2, tol=0.000
1...............
[CV 2/5; 25/48] END C=1.0, max_iter=100, penalty=l2, tol=0.0001;, score=
0.756 total time=  17.4s
[CV 3/5; 25/48] START C=1.0, max_iter=100, penalty=l2, tol=0.000
```

Evaluation of FineTuned Logsitic Regression Classifier

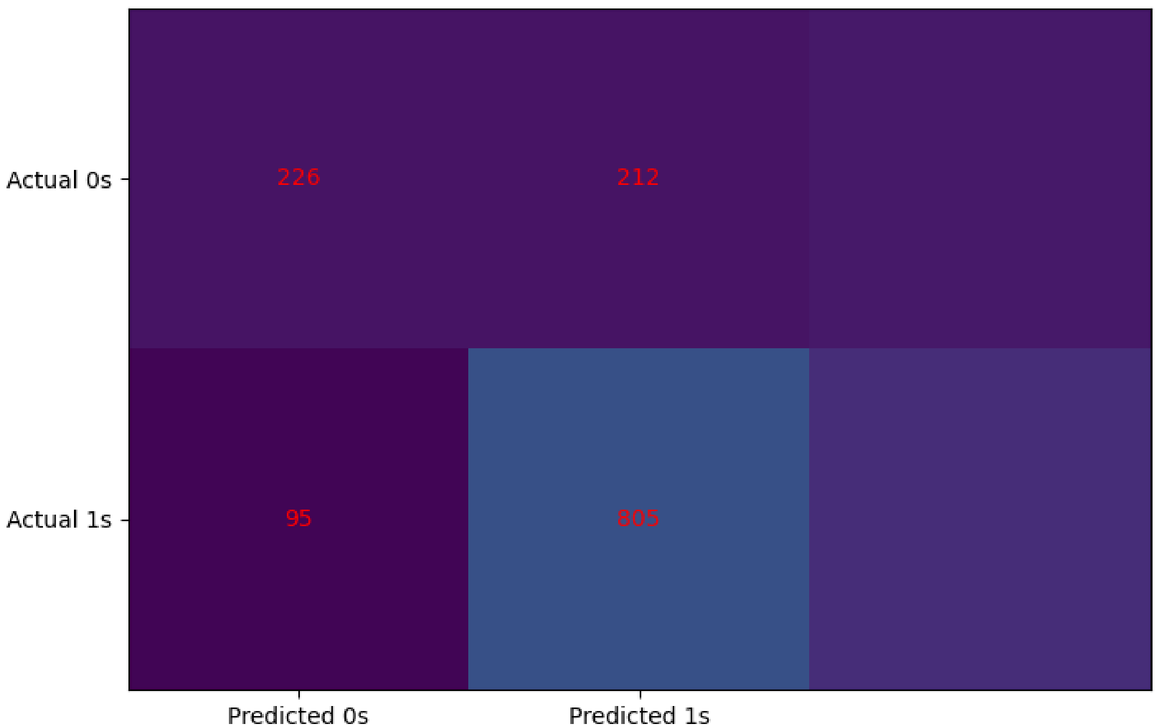
In [91]:
```
%%time

# Print precision, AUC, and F1 scores for the Random Forest Classifier
precision_train_7 = precision_score(y_train, model_7.predict(x_train_tfidf), 
print("Precision Score on training dataset for Finetuned Logistic Regression 

# Calculate AUC for multiclass
y_train_proba = model_7.predict_proba(x_train_tfidf)
auc_train_7 = roc_auc_score(y_train, y_train_proba, multi_class='ovo', averag
print("AUC Score on training dataset for Finetuned Logistic Regression Classi

f1_score_train_7 = f1_score(y_train, model_7.predict(x_train_tfidf), average=
print("F1 Score on training dataset for Finetuned Logistic Regression Classif

precision_test_7 = precision_score(y_test, model_7.predict(x_test_tfidf), ave
print("Precision Score on test dataset for Finetuned Logistic Regression Clas

# Calculate AUC for multiclass
y_test_proba = model_7.predict_proba(x_test_tfidf)
auc_test_7 = roc_auc_score(y_test, y_test_proba, multi_class='ovo', average='
print("AUC Score on test dataset for Finetuned Logistic Regression Classifier

f1_score_test_7 = f1_score(y_test, model_7.predict(x_test_tfidf), average="we
print("F1 Score on test dataset for Finetuned Logistic Regression Classifier:
```

```
Precision Score on training dataset for Finetuned Logistic Regression Classi
fier: 0.8422318795592115
AUC Score on training dataset for Finetuned Logistic Regression Classifier:
0.9402891633252891
F1 Score on training dataset for Finetuned Logistic Regression Classifier:
0.8362573822276566
Precision Score on test dataset for Finetuned Logistic Regression Classifie
r: 0.7501357957631722
AUC Score on test dataset for Finetuned Logistic Regression Classifier: 0.84
78394721920592
F1 Score on test dataset for Finetuned Logistic Regression Classifier: 0.736
053044545597
Wall time: 1.35 s
```

In [92]:
```
y_predict=model_1.predict(x_test_tfidf)
y_predict_prob=model_1.predict_proba(x_test_tfidf)[:,1]
```

In [98]:
```
def confusion_matrix_plot(y_test,y_score):
    confmatrix = confusion_matrix(y_test,y_score)
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.imshow(confmatrix)
    ax.grid(False)
    ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s'))
    ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s'))
    ax.set_ylim(1.5, -0.5)
    for i in range(2):
        for j in range(2):
            ax.text(j, i, confmatrix[i, j], ha='center', va='center', color='
    plt.show()
```

In [99]:
```
confusion_matrix_plot(y_test,y_predict)
```



In [ ]: