Analysis of Reddit Comments

Importing Relevant Modules

In [1]:
```
import re
import nltk
import pickle
import pandas as pd
import plotly.express as px
import matplotlib.pyplot as plt

from tqdm import tqdm
from textblob import TextBlob
from collections import Counter

from nltk.util import ngrams
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from sklearn.metrics import plot_confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer

# nltk.download('punkt')
# nltk.download('wordnet')
# nltk.download('stopwords')
# nltk.download('averaged_perceptron_tagger')
```

Reading Dataset

In [2]:
```
reddit_df = pd.read_pickle('redditDataset.pkl')
reddit_df.drop_duplicates(inplace=True)
reddit_df.reset_index(inplace=True,drop=True)
reddit_df
```

Out[2]:

	text	subreddit
0	that sounds like the answer of a man who does ...	humor
1	i always wonder what people get out of reposti...	humor
2	fuck that downvote them for stalking you in t...	humor
3	seeing as how much this has been around and a...	humor
4	it took me a while to even get my legally requ...	humor
...	...	...
3646	i would recommend anyone looking for a more de...	news
3647	so much innuendo so little facts why so coy i...	news
3648	well mlk was black and bernie is white also ...	news
3649	a man intentionally kills animals and gets 50 ...	news
3650	he is right he was part of the obstruction bac...	news

3651 rows × 2 columns

Text Preprocessing

In [3]:
```
def clean(x):

    WL = WordNetLemmatizer()
    web = ["http","https","imgur","com","org","edu","www"]
    stop_words = stopwords.words('english')

    # Removing unnecessary characters
    x = x.lower().strip()
    x = re.sub(r'[^a-z\s]', '',x)

    w = word_tokenize(x)
    text = []

    # Removing Stop Words, Web Links and Lemmatizing
    for i in w:

        valid = True
        word = WL.lemmatize(i)

        for j in web:
            if j in i: valid = False

        if i in stop_words:
            valid = False

        if len(i)<3:
            valid = False

        if valid:
            text.append(word)

    text = " ".join(text)

    return text
```

In [4]:
```
reddit_df['text'] = [clean(x) for x in tqdm(list(reddit_df.text),position=0,leave=True)]
```

```
100%|██████████| 3651/3651 [00:04<00:00, 774.06it/s]
```

Calculating Polarity & Subjectivity

In [5]:
```
reddit_df['polarity'] = [TextBlob(x).sentiment.polarity for x in reddit_df.text]
reddit_df['subjectivity'] = [TextBlob(x).sentiment.subjectivity for x in reddit_df.text]
```

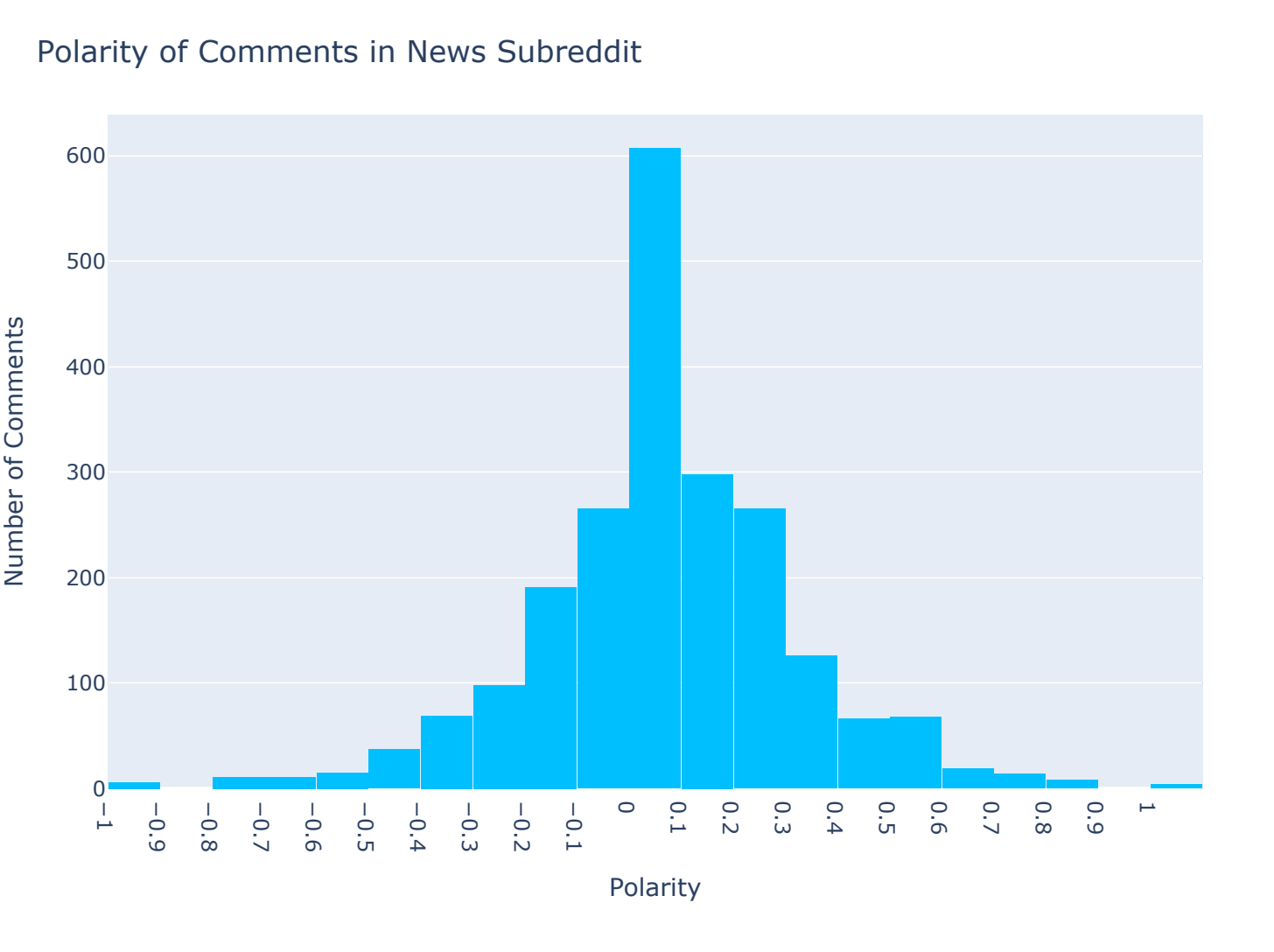
Splitting into Categories

In [6]:
```
news_df = reddit_df[reddit_df.subreddit.str.contains("news")].copy()
news_df.reset_index(inplace=True,drop=True)

humor_df = reddit_df[reddit_df.subreddit.str.contains("humor")].copy()
humor_df.reset_index(inplace=True,drop=True)
```

News Subreddit

Polarity of Comments

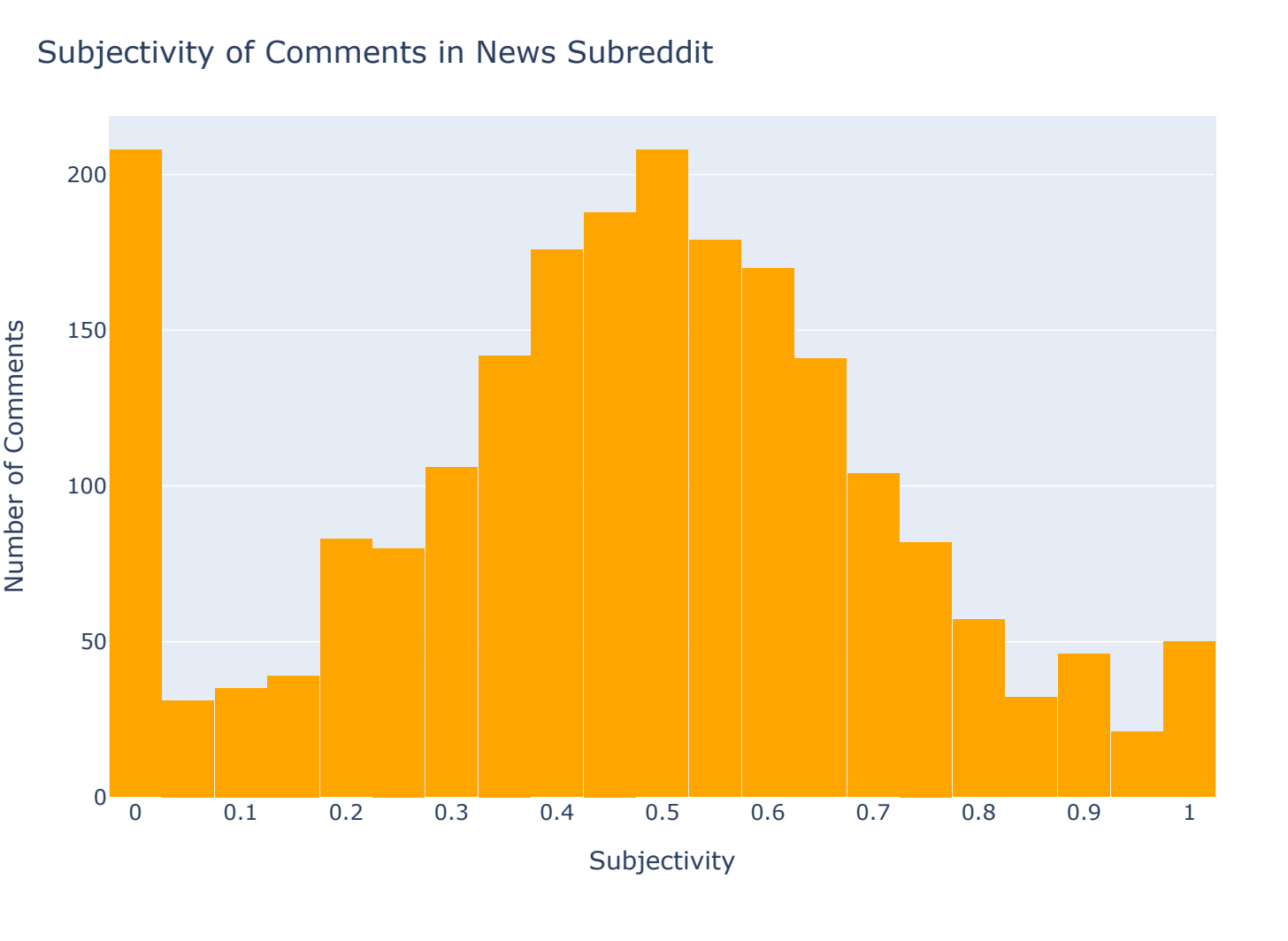
In [7]:
```
fig = px.histogram(news_df, x="polarity",color_discrete_sequence=['deepskyblue'],nbins=40)
fig.update_xaxes(title='Polarity',tickmode = 'array',tickvals = [i/10 for i in range(-10,11)])
fig.update_yaxes(title='Number of Comments')
fig.update_layout(title="Polarity of Comments in News Subreddit",bargap=0.01)
fig.show()

print("Mean Polarity in Comments of News Subreddit:",round(news_df.polarity.mean(),3))
print("Standard Deviation of Polarity in Comments of News Subreddit :",round(news_df.polarity.std(),3))
```




```
Mean Polarity in Comments of News Subreddit: 0.053
Standard Deviation of Polarity in Comments of News Subreddit : 0.251
```

Subjectivity of Comments

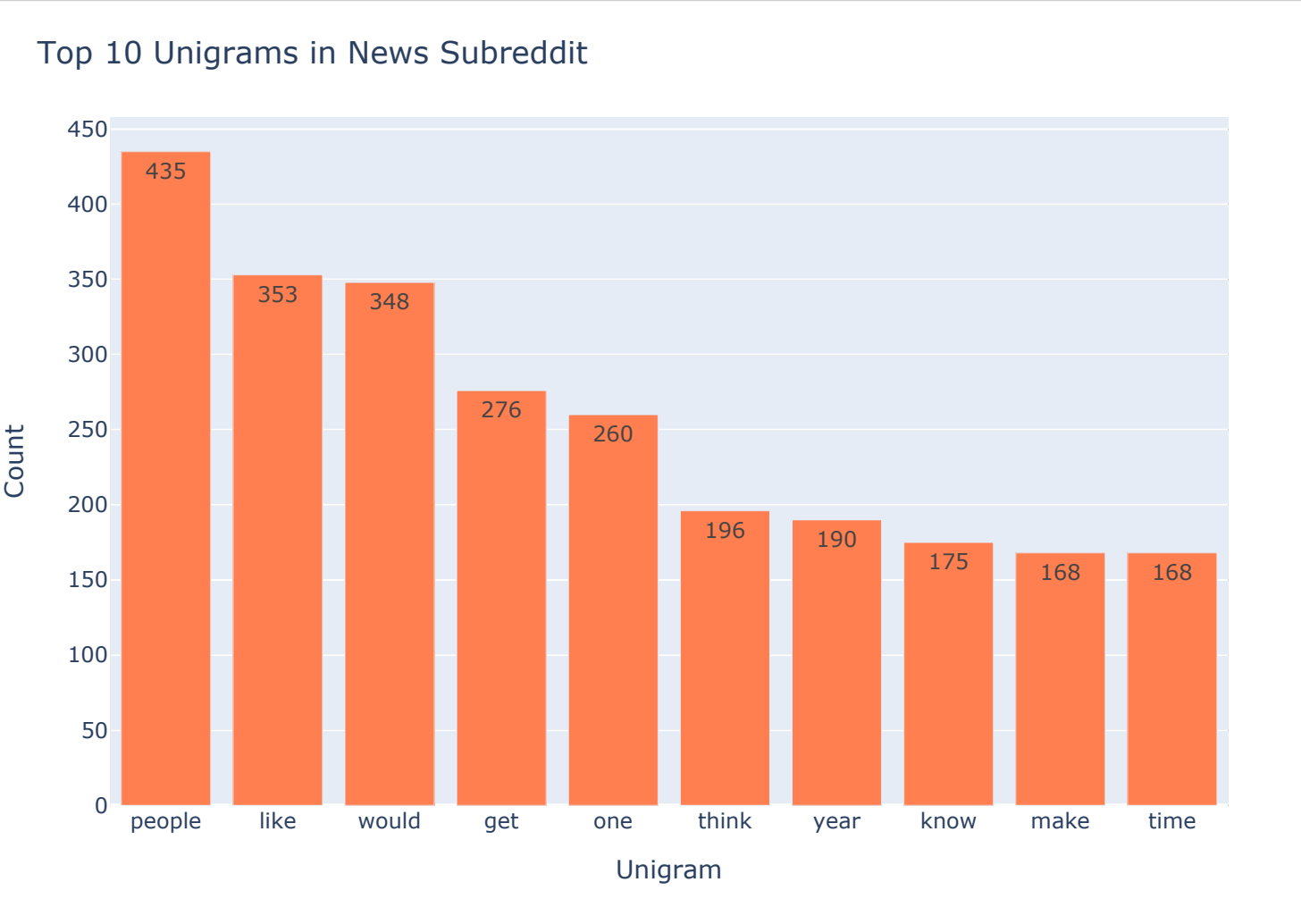
In [8]:
```
fig = px.histogram(news_df, x="subjectivity",color_discrete_sequence=['orange'],nbins=40)
fig.update_xaxes(title='Subjectivity',tickmode = 'array',tickvals = [i/10 for i in range(11)])
fig.update_yaxes(title='Number of Comments')
fig.update_layout(title="Subjectivity of Comments in News Subreddit",bargap=0.01)
fig.show()

print("Mean Subjectivity in Comments of News Subreddit :",round(news_df.subjectivity.mean(),3))
print("Standard Deviation of Subjectivity in Comments of News Subreddit:",round(news_df.subjectivity.st
d(),3))
```




```
Mean Subjectivity in Comments of News Subreddit : 0.457
Standard Deviation of Subjectivity in Comments of News Subreddit: 0.248
```

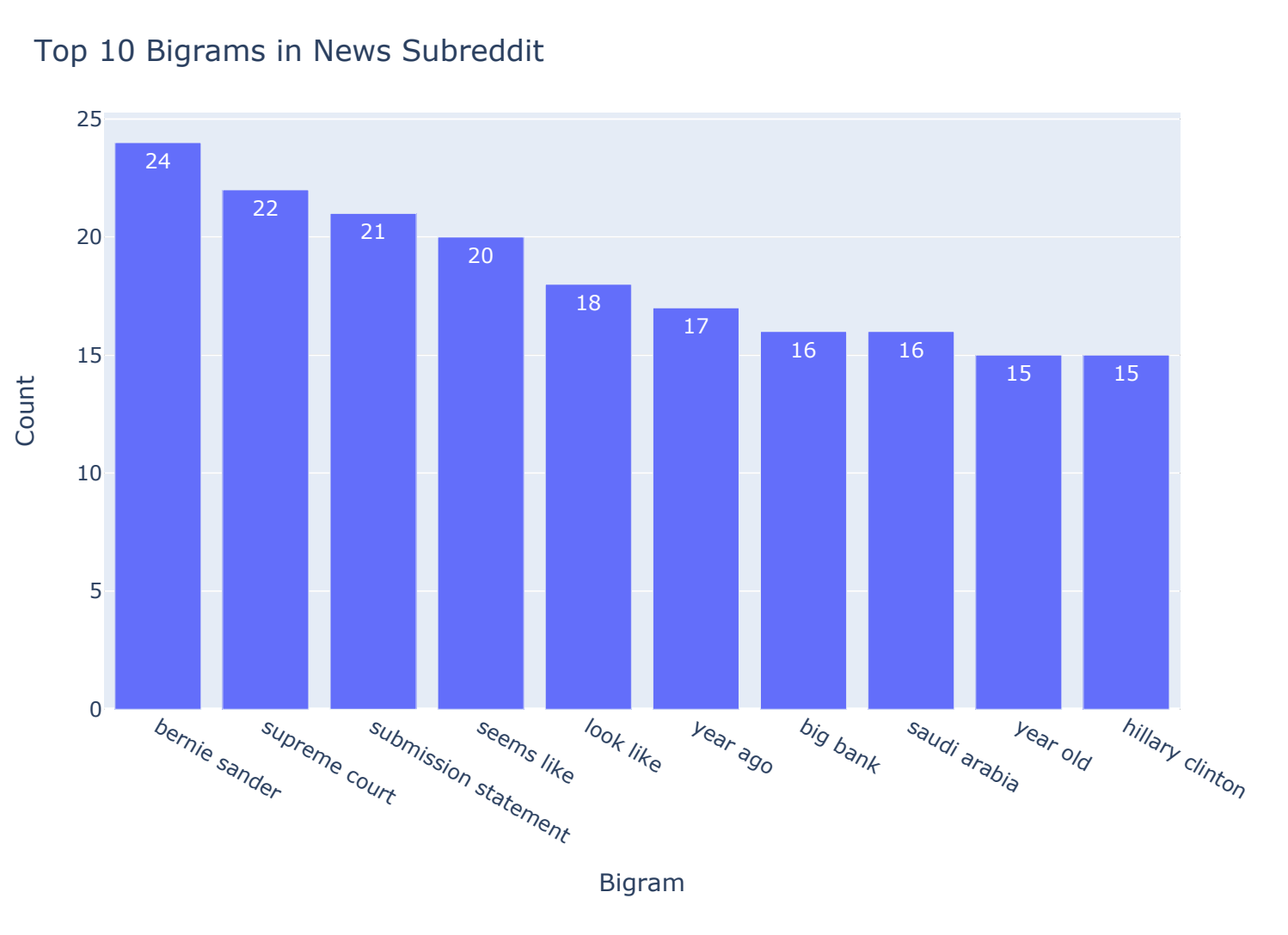
Unigrams

In [9]:
```
# Resolving Unigrams
tokens = word_tokenize(" ".join(list(news_df.text)))
unigrams = ngrams(tokens,1)
unigram_counter = Counter(unigrams)
news_unigram_df = pd.DataFrame()
news_unigram_df['Unigram'] = [i[0][0] for i in unigram_counter.most_common(10)]
news_unigram_df['Count'] = [i[1] for i in unigram_counter.most_common(10)]

# Plotting
fig = px.bar(news_unigram_df, x="Unigram",y='Count',text="Count",color_discrete_sequence=['coral'])
fig.update_layout(title="Top 10 Unigrams in News Subreddit")
fig.show()
```



Bigrams

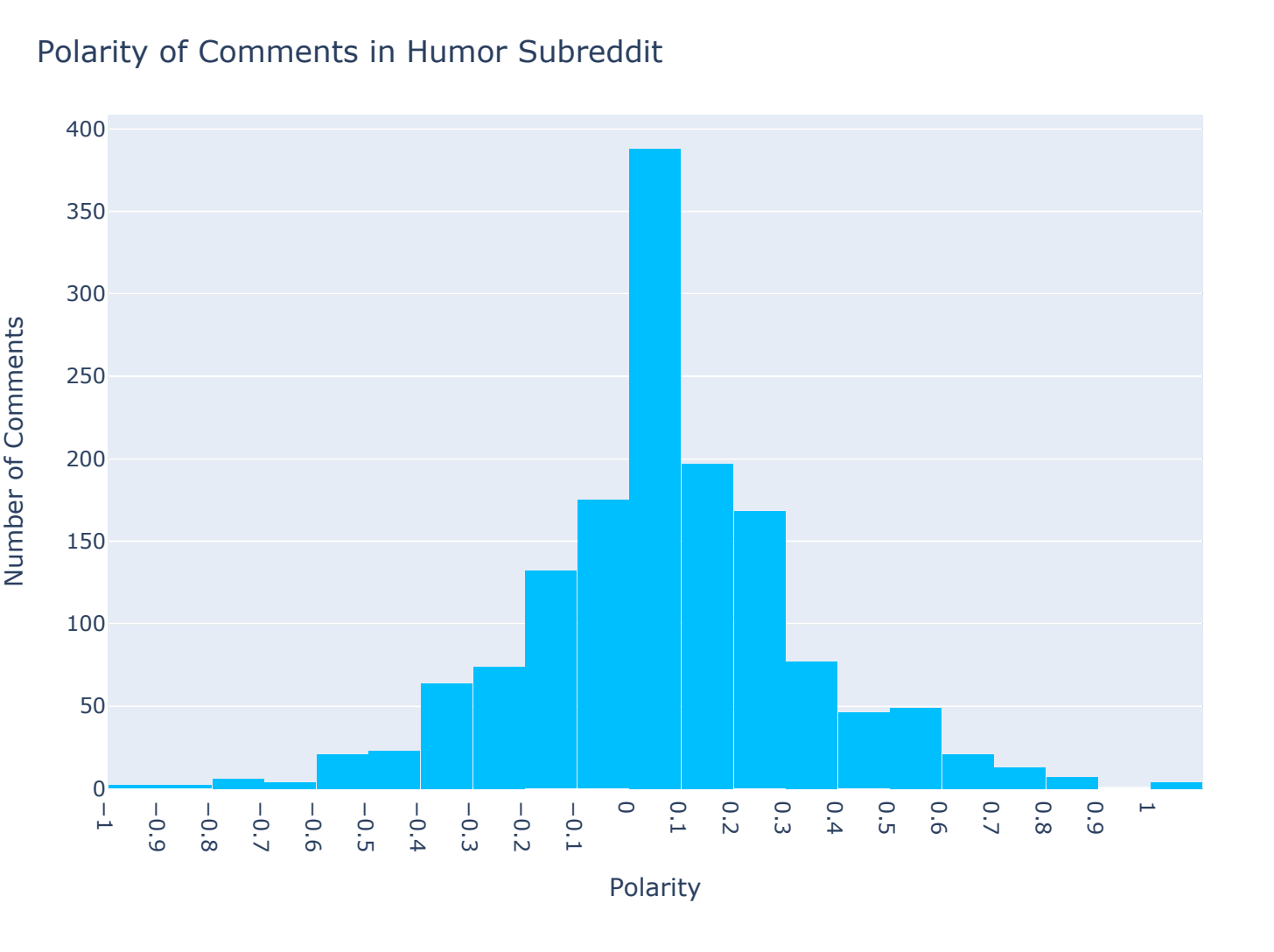
In [10]:
```
# Resolving Bigrams
tokens = word_tokenize(" ".join(list(news_df.text)))
bigrams = ngrams(tokens,2)
bigram_counter = Counter(bigrams)
news_bigram_df = pd.DataFrame()
news_bigram_df['Bigram'] = [i[0][0]+" "+i[0][1] for i in bigram_counter.most_common(10)]
news_bigram_df['Count'] = [i[1] for i in bigram_counter.most_common(10)]

# Plotting
fig = px.bar(news_bigram_df, x="Bigram",y='Count',text="Count")
fig.update_layout(title="Top 10 Bigrams in News Subreddit")
fig.show()
```



Humor Subreddit

Polarity of Comments

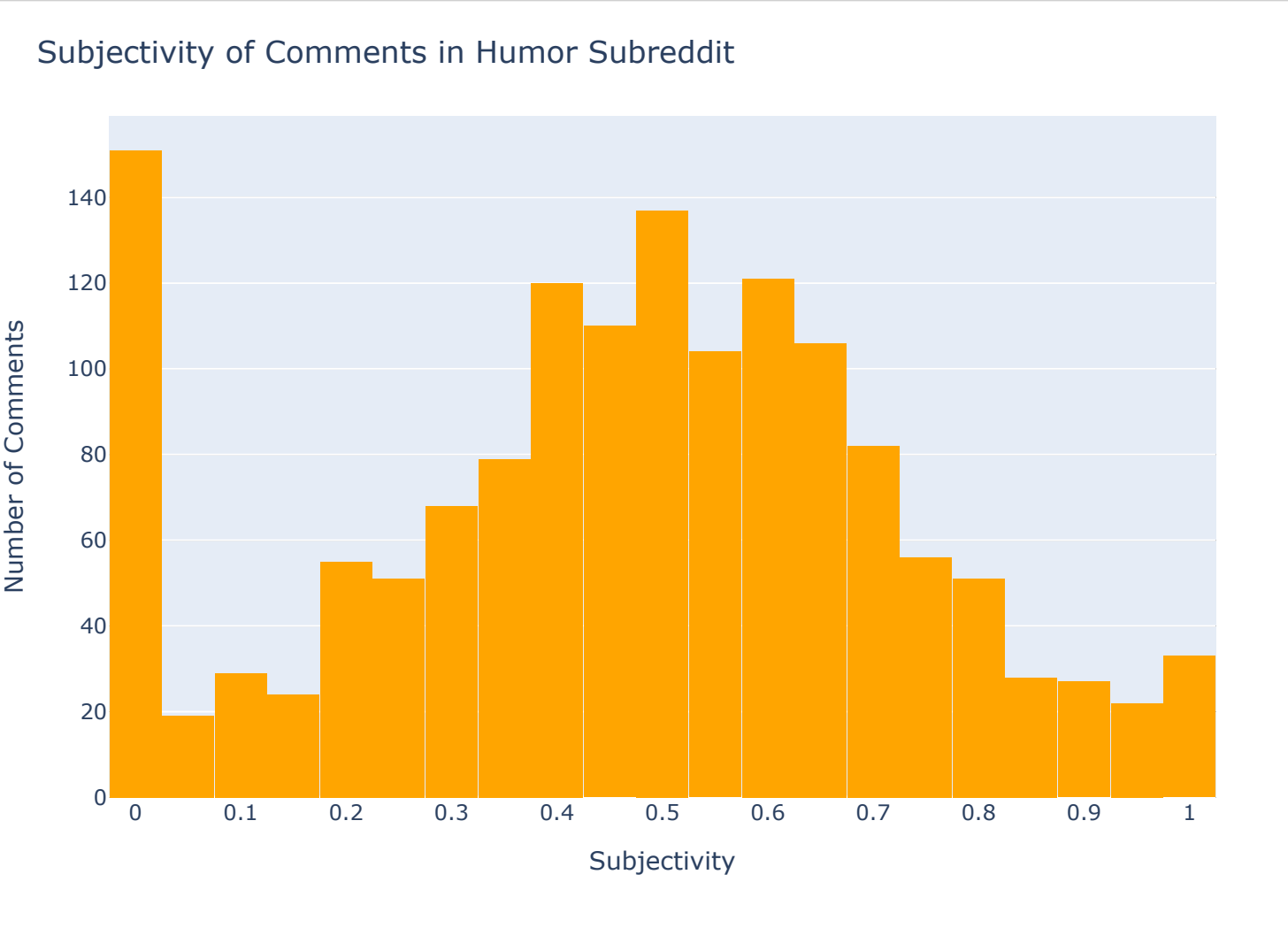
In [11]:
```
fig = px.histogram(humor_df, x="polarity",color_discrete_sequence=['deepskyblue'],nbins=40)
fig.update_xaxes(title='Polarity',tickmode = 'array',tickvals = [i/10 for i in range(-10,11)])
fig.update_yaxes(title='Number of Comments')
fig.update_layout(title=" Polarity of Comments in Humor Subreddit",bargap=0.01)
fig.show()

print("Mean Polarity in Comments of Humor Subreddit:",round(humor_df.polarity.mean(),3))
print("Standard Deviation of Polarity in Comments of Humor Subreddit :",round(humor_df.polarity.std(),3
))
```




```
Mean Polarity in Comments of Humor Subreddit: 0.049
Standard Deviation of Polarity in Comments of Humor Subreddit : 0.265
```

Subjectivity of Comments

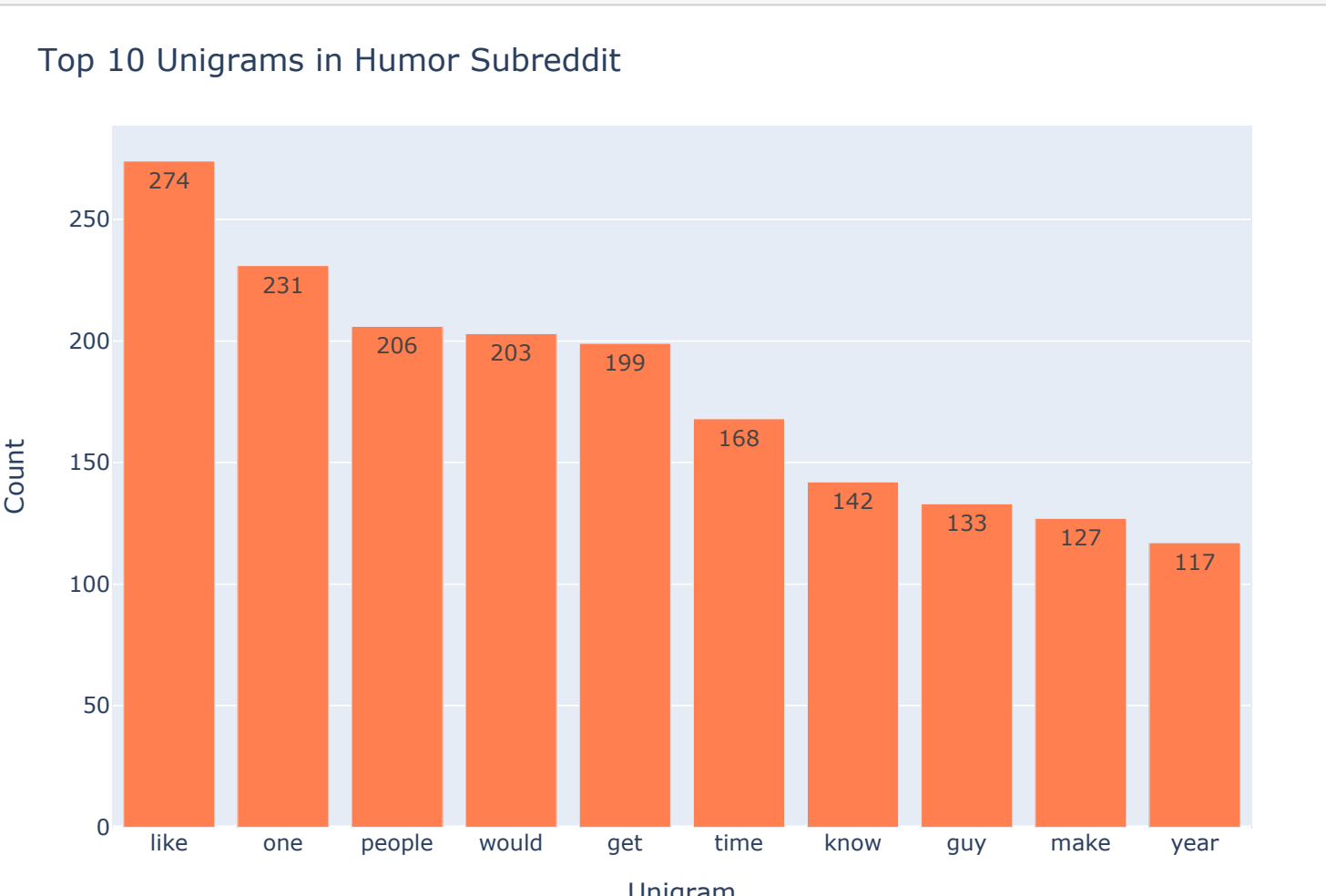
In [12]:
```
fig = px.histogram(humor_df, x="subjectivity",color_discrete_sequence=['orange'],nbins=40)
fig.update_xaxes(title='Subjectivity',tickmode = 'array',tickvals = [i/10 for i in range(11)])
fig.update_yaxes(title='Number of Comments')
fig.update_layout(title="Subjectivity of Comments in Humor Subreddit",bargap=0.01)
fig.show()

print("Mean Subjectivity in Comments of Humor Subreddit :",round(humor_df.subjectivity.mean(),3))
print("Standard Deviation of Subjectivity in Comments of Humor Subreddit:",round(humor_df.subjectivity.
std(),3))
```




```
Mean Subjectivity in Comments of Humor Subreddit : 0.464
Standard Deviation of Subjectivity in Comments of Humor Subreddit: 0.256
```

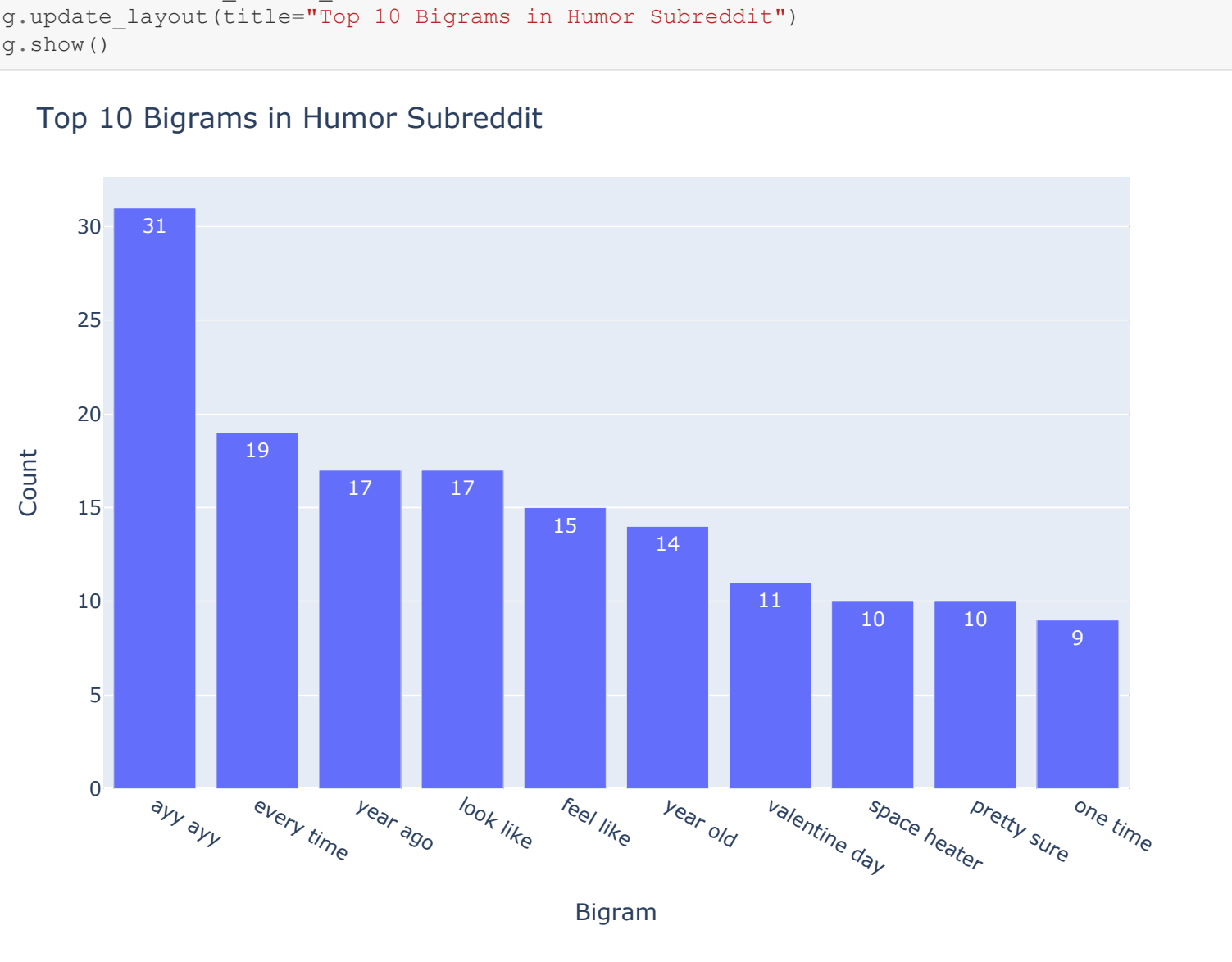
Unigrams

In [13]:
```
# Resolving Unigrams
tokens = word_tokenize(" ".join(list(humor_df.text)))
unigrams = ngrams(tokens,1)
unigram_counter = Counter(unigrams)
humor_unigram_df = pd.DataFrame()
humor_unigram_df['Unigram'] = [i[0][0] for i in unigram_counter.most_common(10)]
humor_unigram_df['Count'] = [i[1] for i in unigram_counter.most_common(10)]

# Plotting
fig = px.bar(humor_unigram_df, x="Unigram",y='Count',text="Count",color_discrete_sequence=['coral'])
fig.update_layout(title="Top 10 Unigrams in Humor Subreddit")
fig.show()
```



Bigrams

In [14]:
```
# Resolving Bigrams
tokens = word_tokenize(" ".join(list(humor_df.text)))
bigrams = ngrams(tokens,2)
bigram_counter = Counter(bigrams)
humor_bigram_df = pd.DataFrame()
humor_bigram_df['Bigram'] = [i[0][0]+" "+i[0][1] for i in bigram_counter.most_common(10)]
humor_bigram_df['Count'] = [i[1] for i in bigram_counter.most_common(10)]

#Plotting
fig = px.bar(humor_bigram_df, x="Bigram",y='Count',text="Count")
fig.update_layout(title="Top 10 Bigrams in Humor Subreddit")
fig.show()
```



Category Classification

Preprocessing

In [15]:
```
# Vectorizing Text
vectorizer = CountVectorizer()
vectorizer.fit(reddit_df.text)

# Train-Test Split
train, test = train_test_split(reddit_df.copy(),test_size=0.3,stratify=reddit_df['subreddit'],random_st
ate=10)
train.reset_index(inplace=True,drop=True)
test.reset_index(inplace=True,drop=True)

x_train = train.text
y_train = train.subreddit.replace({'humor': 0,'news': 1})
x_test = test.text
y_test = test.subreddit.replace({'humor': 0,'news': 1})

x_train = vectorizer.transform(x_train)
x_test = vectorizer.transform(x_test)
```

Multinomial Naive Bayes

In [16]:
```
model = MultinomialNB()
model.fit(x_train,y_train)
print("Accuracy :",round(model.score(x_test,y_test),4))
```

```
Accuracy : 0.8148
```

Logistic Regression

In [17]:
```
model = LogisticRegression(random_state=1)
model.fit(x_train,y_train)
print("Accuracy :",round(model.score(x_test,y_test),4))
```

```
Accuracy : 0.7883
```

Random Forest Classifier

In [ ]:
```
model = RandomForestClassifier(n_estimators=100,random_state=1)
model.fit(x_train,y_train)
print("Accuracy :",round(model.score(x_test,y_test),4))
```

Best Model : Multinomial Naive Bayes

In [ ]:
```
model = MultinomialNB()
model.fit(x_train,y_train)

# Storing Model as Pickle
with open('reddit_model.pkl','wb') as file:
    pickle.dump(model,file)

y_pred = model.predict(x_test)

# Accuracy, Precision, Recall, F1-Score
print(classification_report(y_test,y_pred,target_names=["Humor","News"]))
```

In [ ]:
```
# Confusion Matrix
plot_confusion_matrix(model,x_test,y_test,display_labels=["Humor","News"],cmap="Blues",colorbar=False)
plt.show()
```

In [ ]:
```
# Predicting Probabilities
y_prob = model.predict_proba(x_test)
test['label'] = test.subreddit.replace({'humor': 0,'news': 1})
test['probability'] = [y_prob[i][test.label[i]] for i in range(len(test))]
test.sort_values('probability',inplace=True,ignore_index=True)
```

Comments on which model performed well

In [ ]:
```
good = test.tail()[::-1]
print("----------------------------------------------------------------------------------------------------
-------------------")
print("Comments on which model performed well :- ")
print("----------------------------------------------------------------------------------------------------
-------------------")
for i in good.index:
    print()
    print(">> Correctly Predicted as "+["Humor","News"][good.label[i]]+" <<")
    print("--> Predicted Probability of "+["Humor","News"][good.label[i]]+" : "+str(good.probability[i]
*100)+"%")
    print()
    print(good.text[i])
    print("------------------------------------------------------------------------------------------------
------------------------")
```

Comments on which model failed

In [ ]:
```
bad = test.head()
print("----------------------------------------------------------------------------------------------------
-------------------")
print("Comments on which model performed bad :- ")
print("----------------------------------------------------------------------------------------------------
-------------------")

for i in bad.index:
    print()
    print(">> Wrongly Predicted as "+["Humor","News"][1-bad.label[i]]+" <<")
    print("--> Predicted Probability of "+["Humor","News"][bad.label[i]]+" : "+str(bad.probability[i]*1
00)+"%")
    print()
    print(bad.text[i])
    print("------------------------------------------------------------------------------------------------
------------------------")
```