In [1]:
```
#!pip install spacy
```

In [2]:
```
#!python -m spacy download en_core_web_sm
```

In [3]:
```
import warnings
warnings.filterwarnings('ignore')

# From Example Code  https://github.com/Swathiu/Detecting-Fake-Reviews/blob/master/Deception_Detection.py
import pandas as pd
import numpy as np

from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer

from datetime import datetime
from time import time

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, pairwise_distances
from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm

import spacy
```

In [4]:
```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', None)
```

In [5]:
```
file_path = "C:/Users/tsaie/OneDrive/Desktop/000 Resumes & Projects/# Projects/DS3 Fake Amazon Reviews/Dataset/"
# apparel = pd.read_csv(file_path + 'amazon_reviews_us_Apparel_v1_00.tsv.gz', compression='gzip', header=0, sep='\t', qu
electronics = pd.read_csv(file_path + 'amazon_reviews_us_Electronics_v1_00.tsv.gz', compression='gzip', header=0, sep='\
print(f"The 'electronics' file has {electronics.shape[0]} rows and {electronics.shape[1]} columns")
electronics.head(3)
```

```
The 'electronics' file has 20000 rows and 15 columns
```

Out[5]:

	marketplace	customer_id	review_id	product_id	product_parent	product_title	product_category	star_rating	helpful_votes	total_vot
0	US	41409413	R2MTG1GCZLR2DK	B00428R89M	112201306	yoomall 5M Antenna WIFI RP-SMA Female to Male Extensionl Cable	Electronics	5	0	
1	US	49668221	R2HBOEM8LE9928	B000068O48	734576678	Hosa GPM-103 3.5mm TRS to 1/4" TRS Adaptor	Electronics	5	0	
2	US	12338275	R1P4RW1R9FDPEE	B000GGKOG8	614448099	Channel Master Titan 2 Antenna Preamplifier	Electronics	5	1	

In [6]:
```
electronics_small = electronics[['verified_purchase', 'review_body']]
electronics_small.head()
```

Out[6]:

	verified_purchase	review_body
0	Y	As described.
1	Y	It works as advertising.
2	Y	Works pissa
3	Y	Did not work at all.
4	Y	Works well. Bass is somewhat lacking but is present. Overall pleased with the item.

Create Balanced Dataset

- have same number of rows of verified and unverified reviews

In [7]:
```
def under_sampling(df):
    print("Under-Sampling Data")
    # Count of Reviews
    print("Verified:", sum(df['verified_purchase'] == 'Y'))
    print("Un-Verified:", sum(df['verified_purchase'] == 'N'))

    sample_size = sum(df['verified_purchase'] == 'N')

    authentic_reviews_df = df[df['verified_purchase'] == 'Y']
    fake_reviews_df = df[df['verified_purchase'] == 'N']

    authentic_reviews_us_df = authentic_reviews_df.sample(sample_size)
    under_sampled_df = pd.concat([authentic_reviews_us_df, fake_reviews_df], axis=0)

    print("Under-Sampled Verified", sum(under_sampled_df['verified_purchase'] == 'Y'))
    print("Under-Sampled Un-Verified", sum(under_sampled_df['verified_purchase'] == 'N'))


    # Graph of Data Distribution
    fig, ax = plt.subplots(figsize=(6, 4))
    sns.countplot(x='verified_purchase', data=under_sampled_df)
    plt.title("Count of Reviews")
    plt.show()
    print("Under-Sampling Complete")
    return under_sampled_df
```

In [8]:
```
electronics_equal_weight = under_sampling(electronics_small)
# electronics_equal_weight
```

```
Under-Sampling Data
Verified: 18309
Un-Verified: 1691
Under-Sampled Verified 1691
Under-Sampled Un-Verified 1691
```


```
Under-Sampling Complete
```

Data Cleaning

In [9]:
```
# Pre-processing Text Reviews
def data_cleaning(df):
    # Removing emtpy cells
    df.dropna(inplace=True)
    df['review_body_cleaned'] = df['review_body']

    # Replace HTML keywords with blank space ("&quot;", "br", "&#34")
    remove_dict = {"<br /><br />": " ", "<br />": " ", "br": " ", "&quot;": " ", "&#34": " "}
    for key, val in remove_dict.items():
        df['review_body_cleaned'] = df['review_body_cleaned'].apply(
            lambda x: x.replace(key, val))

    print("\n######## Remove HTML Keywords Complete ########")

    # Remove Punctuations and numbers
    tokenizer = RegexpTokenizer(r'\w+')
    df['review_body_cleaned'] = df['review_body_cleaned'].apply(
        lambda x: ' '.join([word for word in tokenizer.tokenize(x)]))

    remove_dict = {"0": "", "1": "", "2": "", "3": "", "4": "", "5": "", "6": "", "7": "", "8": "", "9": "",
                   "(": "", ")":""}
    for key, val in remove_dict.items():
        df['review_body_cleaned'] = df['review_body_cleaned'].apply(
            lambda x: x.replace(key, val))

    print("\n######## Remove Punctuation and Numbers Complete ########")

    # Lowercase Words
    df['review_body_cleaned'] = df['review_body_cleaned'].str.lower()

    print("\n######## Lowercase Complete ########")

    # Remove Stop Words.
    stop = stopwords.words('english')
    stop += ["can't", "i'm", "I'm", "i'd", "i've", "i'll", "that's", "there's", "they're"]

    df['review_body_cleaned'] = df['review_body_cleaned'].apply(
        lambda x: ' '.join([word for word in x.split() if word.strip() not in stop]))

    print("\n######## Remove Stop Words Complete ########")

    # Lemmatization using .lemma_
    nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])
    df['review_body_cleaned'] = df['review_body_cleaned'].apply(
        lambda x: ' '.join([token.lemma_ for token in nlp(x)]))

    print("\n######## Data Cleaning Complete ########")

    return df
```

In [10]:
```
# Clean the dataset
electronics_cleaned = data_cleaning(electronics_equal_weight)
electronics_cleaned.head()
```

```
######## Remove HTML Keywords Complete ########

######## Remove Punctuation and Numbers Complete ########

######## Lowercase Complete ########

######## Remove Stop Words Complete ########

######## Data Cleaning Complete ########
```

Out[10]:

	verified_purchase	review_body	review_body_cleaned
13179	Y	Batteries arrived as schedules and were fully charged as needed.	battery arrive schedule fully charge need
5040	Y	ok	ok
14608	Y	Works great!	work great
4935	Y	very good open back gaming headset. there isn't very much bass because it is an open back style of headset.	good open back gaming headset much bass open back style headset
17991	Y	The sound is good, it does get loud for its size. However when I tried to use it with my laptop using the Bluetooth (the reason I bought it) it caused my music player and everything else I was running to lag or freeze up. Using it with the I-pod it also would lag or freeze. However if using an audio jack with either item it worked like a charm.	sound good get loud size however try use laptop use bluetooth reason buy cause music player everything else run lag freeze use pod also would lag freeze however use audio jack either item work like charm

Feature Engineering + Prepare Data for Machine Learning

Add Bigrams

In [11]:
```
# https://stackoverflow.com/questions/48331315/how-to-extract-all-the-ngrams-from-a-text-dataframe-column-in-different-or

from collections import Counter
from nltk import ngrams
from itertools import chain

def find_ngrams(input_list, n):
    return list(zip(*[input_list[i:] for i in range(n)]))

def add_bigram_column(df):
    copy = df.copy()
    copy['bigrams'] = copy['review_body_cleaned'].map(lambda x: find_ngrams(x.split(), 2))
    return copy

electronics_cleaned = add_bigram_column(electronics_cleaned)
electronics_cleaned.head()
```

Out[11]:

	verified_purchase	review_body	review_body_cleaned	bigrams
13179	Y	Batteries arrived as schedules and were fully charged as needed.	battery arrive schedule fully charge need	[(battery, arrive), (arrive, schedule), (schedule, fully), (fully, charge), (charge, need)]
5040	Y	ok	ok	[]
14608	Y	Works great!	work great	[(work, great)]
4935	Y	very good open back gaming headset. there isn't very much bass because it is an open back style of headset.	good open back gaming headset much bass open back style headset	[(good, open), (open, back), (back, gaming), (gaming, headset), (headset, much), (much, bass), (bass, open), (open, back), (back, style), (style, headset)]
17991	Y	The sound is good, it does get loud for its size. However when I tried to use it with my laptop using the Bluetooth (the reason I bought it) it caused my music player and everything else I was running to lag or freeze up. Using it with the I-pod it also would lag or freeze. However if using an audio jack with either item it worked like a charm.	sound good get loud size however try use laptop use bluetooth reason buy cause music player everything else run lag freeze use pod also would lag freeze however use audio jack either item work like charm	[(sound, good), (good, get), (get, loud), (loud, size), (size, however), (however, try), (try, use), (use, laptop), (laptop, use), (use, bluetooth), (bluetooth, reason), (reason, buy), (buy, cause), (cause, music), (music, player), (player, everything), (everything, else), (else, run), (run, lag), (lag, freeze), (freeze, use), (use, pod), (pod, also), (also, would), (would, lag), (lag, freeze), (freeze, however), (however, use), (use, audio), (audio, jack), (jack, either), (either, item), (item, work), (work, like), (like, charm)]

Let's call the bigrams in vertified reviews "gold_bigrams" and the bigrams in unverified reviews "fake_bigrams"

In [12]:
```
electronics_fake = electronics_cleaned[electronics_cleaned['verified_purchase'] == 'N']
electronics_gold = electronics_cleaned[electronics_cleaned['verified_purchase'] == 'Y']

electronics_fake.tail(1)
```

Out[12]:

	verified_purchase	review_body	review_body_cleaned	bigrams
19952	N	I use them at work for my chargers. Keeps cords looking organized and I love it.	use work charger keep cord look organize love	[(use, work), (work, charger), (charger, keep), (keep, cord), (cord, look), (look, organize), (organize, love)]

In [14]:
```
def create_gold_bigrams(df_cleaned):
    global df_gold, gold_bigrams, gold_bigram_counts
    df_gold = df_cleaned[df_cleaned['verified_purchase'] == 'Y']
    gold_bigrams = df_gold['bigrams'].tolist()
    gold_bigrams = list(chain(*gold_bigrams))
    gold_bigram_counts = Counter(gold_bigrams)

create_gold_bigrams(electronics_cleaned)
gold_bigram_counts.most_common(20)
```

Out[14]:
```
[(('work', 'great'), 148),
 (('work', 'well'), 71),
 (('sound', 'quality'), 63),
 (('great', 'sound'), 52),
 (('great', 'product'), 45),
 (('sound', 'great'), 36),
 (('work', 'fine'), 30),
 (('work', 'perfectly'), 28),
 (('easy', 'install'), 27),
 (('sound', 'good'), 26),
 (('good', 'sound'), 25),
 (('highly', 'recommend'), 24),
 (('great', 'price'), 24),
 (('battery', 'life'), 22),
 (('charge', 'charge'), 22),
 (('good', 'product'), 21),
 (('easy', 'use'), 19),
 (('would', 'recommend'), 18),
 (('good', 'price'), 18),
 (('work', 'like'), 17)]
```

In [15]:
```
def create_fake_bigrams(df_cleaned):
    global df_fake, fake_bigrams, fake_bigram_counts
    df_fake = df_cleaned[df_cleaned['verified_purchase'] == 'N']
    fake_bigrams = df_fake['bigrams'].tolist()
    fake_bigrams = list(chain(*fake_bigrams))
    fake_bigram_counts = Counter(fake_bigrams)

create_fake_bigrams(electronics_cleaned)
fake_bigram_counts.most_common(20)
```

Out[15]:
```
[(('sound', 'quality'), 265),
 (('bluetooth', 'speaker'), 133),
 (('good', 'sound'), 99),
 (('work', 'well'), 98),
 (('work', 'great'), 94),
 (('exchange', 'honest'), 93),
 (('receive', 'product'), 88),
 (('listen', 'music'), 88),
 (('battery', 'life'), 84),
 (('honest', 'review'), 83),
 (('sound', 'good'), 80),
 (('sound', 'great'), 74),
 (('great', 'sound'), 69),
 (('easy', 'use'), 66),
 (('unbiased', 'review'), 60),
 (('honest', 'unbiased'), 56),
 (('highly', 'recommend'), 55),
 (('product', 'discount'), 52),
 (('can', 'not'), 50),
 (('quality', 'sound'), 49)]
```

Add Features

count = number of gold/fake_bigrams in a review

percent = number of gold/fake_bigrams as a percentage of total number of bigrams in a review.

simple score = sum of the gold/fake_bigrams' popularity scores (calculated using the bigram's count in the Counter)

normalized score = simple score / total bigram count

In [16]:
```
def get_bigram_count(bigrams, bigram_dict):
    count = 0
    for bigram in bigrams:
        if bigram in bigram_dict.keys():
            count += 1
    return count

def get_bigram_simple_score(bigrams, bigram_dict):
    score = 0
    for bigram in bigrams:
        if bigram in bigram_dict.keys():
            score += bigram_dict[bigram]
    return score
```

In [17]:
```
electronics_cleaned['bigram_count'] = electronics_cleaned['bigrams'].apply(lambda x: len(x))
```

In [19]:
```
_dict = dict(fake_bigram_counts)

_cleaned['fake_bigram_count'] = electronics_cleaned['bigrams'].apply(
: get_bigram_count(x, fake_bigram_dict))

_cleaned['fake_bigram_percent'] = electronics_cleaned['fake_bigram_count'] / electronics_cleaned['bigram_count']

_cleaned['fake_bigram_simple_score'] = electronics_cleaned['bigrams'].apply(
: get_bigram_simple_score(x, fake_bigram_dict))

_cleaned['fake_bigram_normalized_score'] = electronics_cleaned['fake_bigram_simple_score'] / electronics_cleaned['bigram_
```

In [20]:
```
ram_dict = dict(gold_bigram_counts)

ics_cleaned['gold_bigram_count'] = electronics_cleaned['bigrams'].apply(
da x: get_bigram_count(x, gold_bigram_dict))

ics_cleaned['gold_bigram_percent'] = electronics_cleaned['gold_bigram_count'] / electronics_cleaned['bigram_count']

ics_cleaned['gold_bigram_simple_score'] = electronics_cleaned['bigrams'].apply(
da x: get_bigram_simple_score(x, gold_bigram_dict))

ics_cleaned['gold_bigram_normalized_score'] = electronics_cleaned['gold_bigram_simple_score'] / electronics_cleaned['bigr
```

In [21]:
```
# Fill all the NaN values with zero
electronics_cleaned = electronics_cleaned.fillna(0)
electronics_cleaned.tail(1)
```

Out[21]:

	verified_purchase	review_body	review_body_cleaned	bigrams	bigram_count	fake_bigram_count	fake_bigram_percent	fake_bigram_simple_
19952	N	I use them at work for my chargers. Keeps cords looking organized and I love it.	use work charger keep cord look organize love	[(use, work), (work, charger), (charger, keep), (keep, cord), (cord, look), (look, organize), (organize, love)]	7	7	1.0	

Use Machine Learning to Make Predictions for Verified VS. Unverified

LABELS:

- 1 = verified review
- 0 = unverified review

In [22]:
```
def semi_supervised_learning(df, model, algorithm, threshold=0.8, iterations=40):
    df = df.copy()

    df_unlabled = df[['fake_bigram_count', 'fake_bigram_percent', 'fake_bigram_simple_score', 'fake_bigram_normalized_sc
             'gold_bigram_count', 'gold_bigram_percent', 'gold_bigram_simple_score', 'gold_bigram_normalized_score']]

    df['verified_purchase'] = df['verified_purchase'].apply(lambda x: 1 if x == 'Y' else 0)
    print("Training " + algorithm + " Model")
    labels = df['verified_purchase']

    train_data, test_data, train_label, test_label = train_test_split(df_unlabled, labels, test_size=0.25, random_state=
```

```
    test_data_copy = test_data.copy()
    test_label_copy = test_label.copy()

    all_labeled = False

    current_iteration = 0

    pbar = tqdm(total=iterations)

    while not all_labeled and (current_iteration < iterations):
        current_iteration += 1
        model.fit(train_data, train_label)

        probabilities = model.predict_proba(test_data)
        pseudo_labels = model.predict(test_data)
```

```
        indices = np.argwhere(probabilities > threshold)

        for item in indices:
            train_data.loc[test_data.index[item[0]]] = test_data.iloc[item[0]]
            train_label.loc[test_data.index[item[0]]] = pseudo_labels[item[0]]
        test_data.drop(test_data.index[indices[:, 0]], inplace=True)
        test_label.drop(test_label.index[indices[:, 0]], inplace=True)

        print("--" * 20)

        if len(test_data) == 0:
            print("Exiting loop")
            all_labeled = True
        pbar.update(1)

    pbar.close()
    predicted_labels = model.predict(test_data_copy)

    print(algorithm + ' Model Results')
    print('--' * 20)
    print('Accuracy Score : ' + str(accuracy_score(test_label_copy, predicted_labels)))
    print('Precision Score : ' + str(precision_score(test_label_copy, predicted_labels, pos_label=1)))
    print('Recall Score : ' + str(recall_score(test_label_copy, predicted_labels, pos_label=1)))
    print('F1 Score : ' + str(f1_score(test_label_copy, predicted_labels, pos_label=1)))
#     print('Confusion Matrix : \n' + str(confusion_matrix(test_label_copy, predicted_labels)))
    plot_confusion_matrix(test_label_copy, predicted_labels, classes=[1, 0],
                          title=algorithm + ' Confusion Matrix').show()

    return model
```

```
def plot_confusion_matrix(y_true, y_pred, classes, title=None, cmap=plt.cm.Blues):
    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           xticklabels=classes,
           yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
```

```
    fig.tight_layout()

    return plt
```

ML Method #1: RandomForestClassifier

In [23]:
```
start_time = time()
# rf = RandomForestClassifier(random_state=42, criterion='entropy', max_depth=14, max_features='auto', n_estimators=500)
rf = RandomForestClassifier(random_state=42, criterion='entropy', max_depth=2, max_features='auto', n_estimators=500)
rf_model = semi_supervised_learning(electronics_cleaned, model=rf, threshold=0.7, iterations=10, algorithm='Random Fores
end_time = time()

print("Time taken : ", end_time - start_time)
```

```
Training Random Forest Model
 10%|█         | 1/10 [00:02<00:18,  2.09s/it]
----------------------------------------
 20%|██        | 2/10 [00:03<00:11,  1.41s/it]
----------------------------------------
 30%|███       | 3/10 [00:03<00:08,  1.19s/it]
----------------------------------------
 40%|████      | 4/10 [00:04<00:06,  1.10s/it]
----------------------------------------
 50%|█████     | 5/10 [00:05<00:05,  1.04s/it]
----------------------------------------
```

ML Method #2: GaussianNB

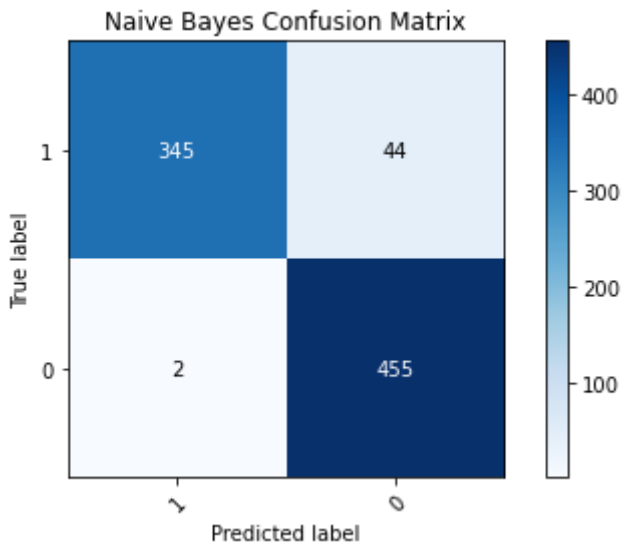
In [24]:
```
start_time = time()
nb = GaussianNB()
nb_model = semi_supervised_learning(electronics_cleaned, model=nb, threshold=0.7, iterations=15, algorithm='Naive Bayes'
end_time = time()

print("Time taken : ", end_time - start_time)
```

```
Training Naive Bayes Model

100%|██████████| 15/15 [00:01<00:00, 10.65it/s]

----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
Naive Bayes Model Results
----------------------------------------
Accuracy Score : 0.9456264775413712
Precision Score : 0.9118236472945892
Recall Score : 0.9956236323851203
F1 Score : 0.9518828451882845
```




```
Time taken :  1.5673511028289795
```

Making Predictions for a User Input

In [25]:
```
def clean_review_and_add_features(review, category_df_cleaned):
    # Create dataframe for the singular review
    df = pd.DataFrame(data={'review_body': review}, index=[0])

    # Generate gold/fake bigrams in order to score the user input
    create_gold_bigrams(category_df_cleaned) # Updates the variables gold_bigrams and gold_bigram_counts
    create_fake_bigrams(category_df_cleaned) # Updates the variables fake_bigrams and fake_bigram_counts

    # Clean the user input
    df = data_cleaning(df)
    df['bigrams'] = df['review_body_cleaned'].map(lambda x: find_ngrams(x.split(), 2))
    df['bigram_count'] = df['bigrams'].apply(lambda x: len(x))

    for gold_or_fake in ['gold', 'fake']:
        exec(f"{gold_or_fake}_bigram_dict = dict({gold_or_fake}_bigram_counts)")

        exec(f"df['{gold_or_fake}_bigram_count'] = \
               df['bigrams'].apply(lambda x: get_bigram_count(x, {gold_or_fake}_bigram_dict))")

        exec(f"df['{gold_or_fake}_bigram_percent'] = \
               df['{gold_or_fake}_bigram_count'] / df['bigram_count']")

        exec(f"df['{gold_or_fake}_bigram_simple_score'] = \
               df['bigrams'].apply(lambda x: get_bigram_simple_score(x, {gold_or_fake}_bigram_dict))")

        exec(f"df['{gold_or_fake}_bigram_normalized_score'] = \
               df['{gold_or_fake}_bigram_simple_score'] / df['bigram_count']")

    return df.fillna(0)
```

Try changing the review and see what the RandomForest model predicts!

In [26]:
```
review = "This is an honest review of the headphone I just bought from Amazon."
category_df_cleaned = electronics_cleaned
user_input_processed_df = clean_review_and_add_features(review, category_df_cleaned)
user_input_processed_df
```

```
######## Remove HTML Keywords Complete ########

######## Remove Punctuation and Numbers Complete ########

######## Lowercase Complete ########

######## Remove Stop Words Complete ########

######## Data Cleaning Complete ########
```

Out[26]:

	review_body	review_body_cleaned	bigrams	bigram_count	gold_bigram_count	gold_bigram_percent	gold_bigram_simple_score	gold_bigram_
0	This is an honest review of the headphone I just bought from Amazon.	honest review headphone buy amazon	[(honest, review), (review, headphone), (headphone, buy), (buy, amazon)]	4	0	0.0	0	

In [27]:
```
def prepare_df_for_prediction(processed_df):
    df = processed_df.copy()
    df = df[['fake_bigram_count', 'fake_bigram_percent', 'fake_bigram_simple_score', 'fake_bigram_normalized_score',
             'gold_bigram_count', 'gold_bigram_percent', 'gold_bigram_simple_score', 'gold_bigram_normalized_score']]
    return df

df_for_prediction = prepare_df_for_prediction(user_input_processed_df)
prediction, probabilities = rf_model.predict(df_for_prediction), rf_model.predict_proba(df_for_prediction)[0]

def interpret_prediction(review, pred, proba):
    proba = [round(proba[0], 3), round(proba[1], 3)]
    if prediction[0] == 1:
        print(f'"{review}" is predicted to be a VERIFIED review, with {proba[1]}% probability of being VERIFIED and {prob
    if prediction[0] == 0:
        print(f'"{review}" is predicted to be an UNVERIFIED review, with {proba[0]}% probability of being UNVERIFIED and

interpret_prediction(review, prediction, probabilities)
```

```
"This is an honest review of the headphone I just bought from Amazon." is predicted to be an UNVERIFIED review, with 0.
766% probability of being UNVERIFIED and 0.234% probability of being VERIFIED
```