```
Start coding or generate with AI.
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from textblob import TextBlob
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectoriz
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
True
```

```
# Load each CSV file
file_paths = [

    "/content/drive/MyDrive/Colab Notebooks/data/B00K8K937I_Puritan'sPrideSuperStren
    "/content/drive/MyDrive/Colab Notebooks/data/B079TD7HG2_NatrolMelatoninSleepAidG
    "/content/drive/MyDrive/Colab Notebooks/data/B07GR9WBFY_CarlyleMelatonin12mgFast
    "/content/drive/MyDrive/Colab Notebooks/data/B07N46LTJJ_ZzzQuilPureZzzsMelatonin
    "/content/drive/MyDrive/Colab Notebooks/data/B07PF1SN5B_vitafusionMaxStrengthMel
    "/content/drive/MyDrive/Colab Notebooks/data/B08451719W_CarlyleMelatonin12mgFast
    "/content/drive/MyDrive/Colab Notebooks/data/B08CGYFB2Q_VitamaticMelatonin20mgTa

]

# Initial loading and inspection
dfs = []
for file_path in file_paths:
    # Extract the dose from the file name
    dose = file_path.split('_')[1].split('Melatonin')[0]
    # Load the CSV file
    df = pd.read_csv(file_path)
    # Add a 'dose' column
    df['dose'] = dose
    dfs.append(df)

# Concatenate all dataframes into a single dataframe
combined_df = pd.concat(dfs, ignore_index=True)

# Display the first few rows and the columns of the combined dataframe
combined_df.head(), combined_df.columns
```

```
                                    author_profile_url  \
0  https://www.amazon.com/gp/profile/amzn1.accoun...
1  https://www.amazon.com/gp/profile/amzn1.accoun...
2  https://www.amazon.com/gp/profile/amzn1.accoun...
3  https://www.amazon.com/gp/profile/amzn1.accoun...
4  https://www.amazon.com/gp/profile/amzn1.accoun...

                                                 url            brand  \
0  https://www.amazon.com/product-reviews/B00K8K9...  Puritan's Pride
1  https://www.amazon.com/product-reviews/B00K8K9...  Puritan's Pride
2  https://www.amazon.com/product-reviews/B00K8K9...  Puritan's Pride
3  https://www.amazon.com/product-reviews/B00K8K9...  Puritan's Pride
4  https://www.amazon.com/product-reviews/B00K8K9...  Puritan's Pride

                                          review_url       input  \
0  https://www.amazon.com/gp/customer-reviews/R34...  B00K8K937I
1  https://www.amazon.com/gp/customer-reviews/R1R...  B00K8K937I
2  https://www.amazon.com/gp/customer-reviews/R1K...  B00K8K937I
3  https://www.amazon.com/gp/customer-reviews/R4D...  B00K8K937I
4  https://www.amazon.com/gp/customer-reviews/R2F...  B00K8K937I

                                             dose
0  Puritan'sPrideSuperStrengthRapidReleaseCapsules
1  Puritan'sPrideSuperStrengthRapidReleaseCapsules
2  Puritan'sPrideSuperStrengthRapidReleaseCapsules
3  Puritan'sPrideSuperStrengthRapidReleaseCapsules
4  Puritan'sPrideSuperStrengthRapidReleaseCapsules

[5 rows x 23 columns],
Index(['asin', 'product_title', 'average_rating', 'total_reviews',
       'review_author', 'author_badge', 'badge', 'reviewed_product_attribute',
       'reviewed_variant_asin', 'variant_review_url', 'review_rating',
       'review_date', 'reviewed_country', 'review_text',
       'review_comment_count', 'review_header',
'no_of_people_reacted_helpful',
       'author_profile_url', 'url', 'brand', 'review_url', 'input', 'dose'],
      dtype='object'))
```

```
import re

# Define a function to extract numeric dose and unit from strings
def extract_dose_info(text):
    match = re.search(r'(\d+mg)', text, re.IGNORECASE)
    return match.group(0) if match else None

# Apply the function to clean up the 'dose' column
combined_df['dose'] = combined_df['dose'].apply(extract_dose_info)

# Remove unnecessary columns. Let's keep 'review_text', 'review_header', and 'dose'
columns_to_keep = ['review_text', 'review_header', 'dose']
cleaned_df = combined_df[columns_to_keep].dropna()

# Checking the first few rows of the cleaned dataframe and the unique doses
cleaned_df.head(), cleaned_df['dose'].unique()
```

```
(Empty DataFrame
 Columns: [review_text, review_header, dose]
 Index: [],
 array([], dtype=object))
```

```
def extract_dose_from_title_or_filename(text):
    # Pattern to find dose information such as '12mg', '10mg', etc.
    match = re.search(r'\b\d+mg\b', text, re.IGNORECASE)
    return match.group(0) if match else 'Unknown'

# Reload the dataframes with corrected 'dose' extraction
corrected_dfs = []
for file_path in file_paths:
    df = pd.read_csv(file_path)
    # Attempt to extract dose from product title first
    df['dose'] = df['product_title'].apply(extract_dose_from_title_or_filename)
    # If dose not found in product title, extract from filename as a fallback
    if df['dose'].iloc[0] == 'Unknown':
        df['dose'] = extract_dose_from_title_or_filename(file_path)
    corrected_dfs.append(df)

# Concatenating all corrected dataframes into a single dataframe
corrected_combined_df = pd.concat(corrected_dfs, ignore_index=True)

# Keeping relevant columns and drop rows with missing values
columns_to_keep = ['review_text', 'review_header', 'dose']
cleaned_corrected_df = corrected_combined_df[columns_to_keep].dropna()

# Checking the first few rows of the cleaned, corrected dataframe and the unique dos
cleaned_corrected_df.head(), cleaned_corrected_df['dose'].unique()
```

```
(                                          review_text  \
 0  These works great, I think the gel capsule dis...
 1                Powerful! Will knock you right out!
 2  Best melatonin I've tried,so much better then ...
 3                   Works like a charm, great price.
 4  I love this product, I use it to sleep at nigh...

                                        review_header     dose
 0  Max dose and it hits faster than other quick r...  Unknown
 1                                        Powerful!  Unknown
 2                                          Amazing  Unknown
 3                                        Five Stars  Unknown
 4   This is one great product, people should try it.  Unknown  ,
 array(['Unknown', '10mg', '1mg', '20mg'], dtype=object))
```

EDA

```
# Manually assign doses based on the filenames as  automatic extraction took longer
manual_dose_assignments = {
    "B00K8K937I_Puritan'sPrideSuperStrengthRapidReleaseCapsules Melatonin.csv": 'Unk
    "B07GR9WBFY_CarlyleMelatonin12mgFastDissolve180Tablets.csv": '12mg',
    "B07N46LTJJ_ZzzQuilPureZzzsMelatoninSleepAidGummies.csv": 'Unknown', # No specif
    "B07PF1SN5B_vitafusionMaxStrengthMelatoninGummySupplements.csv": 'Unknown', # No
    "B08CGYFB2Q_VitamaticMelatonin20mgTablets.csv": '20mg',
    "B079TD7HG2_NatrolMelatoninSleepAidGummy.csv": 'Unknown', # No specific dose men
    "B08451719W_CarlyleMelatonin12mgFastDissolve300Tablets.csv": '12mg'
}

# Applying manual dose assignments
for file_path, dose in manual_dose_assignments.items():
    filename = file_path.split('/')[-1]
    corrected_combined_df.loc[corrected_combined_df['product_title'].str.contains(fi

# Removing entries with 'Unknown' doses
filtered_df = corrected_combined_df[corrected_combined_df['dose'] != 'Unknown']

# So to Keep relevant columns and drop rows with missing values in 'review_text' or
filtered_cleaned_df = filtered_df[columns_to_keep].dropna(subset=['review_text', 're

# To check the first few rows of the filtered, cleaned dataframe and the unique dose
filtered_cleaned_df.head(), filtered_cleaned_df['dose'].unique()
```

```
(                                            review_text  \
 3200  I’m disappointed in the quality of this produc...
 3201  These are the best tasteing and working onez i...
 3202  Best on the market, could be cheaper but I’m f...
 3203  Flavor palatable but kinda flat; definitely no...
 3204  They are actually 5 mg, you have to take 2 gum...

                                          review_header  dose
```

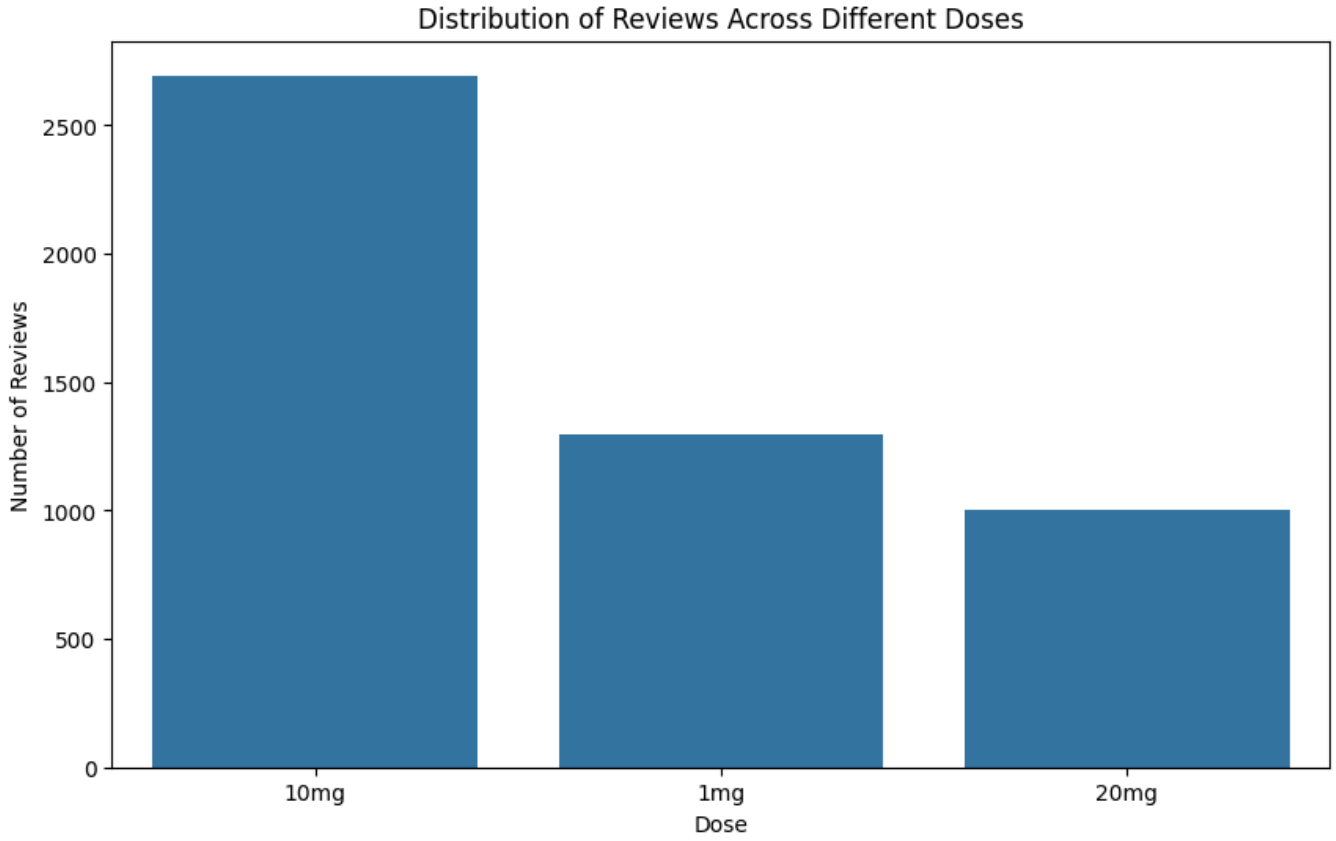
```
3200                                        Not so much  10mg
3201                                        There great  10mg
3202                                         Best brand  10mg
3203  Good price for the quality and quantity of 10mg.  10mg
3204                                         NOT 10 mg  10mg  ,
array(['10mg', '1mg', '20mg'], dtype=object))
```

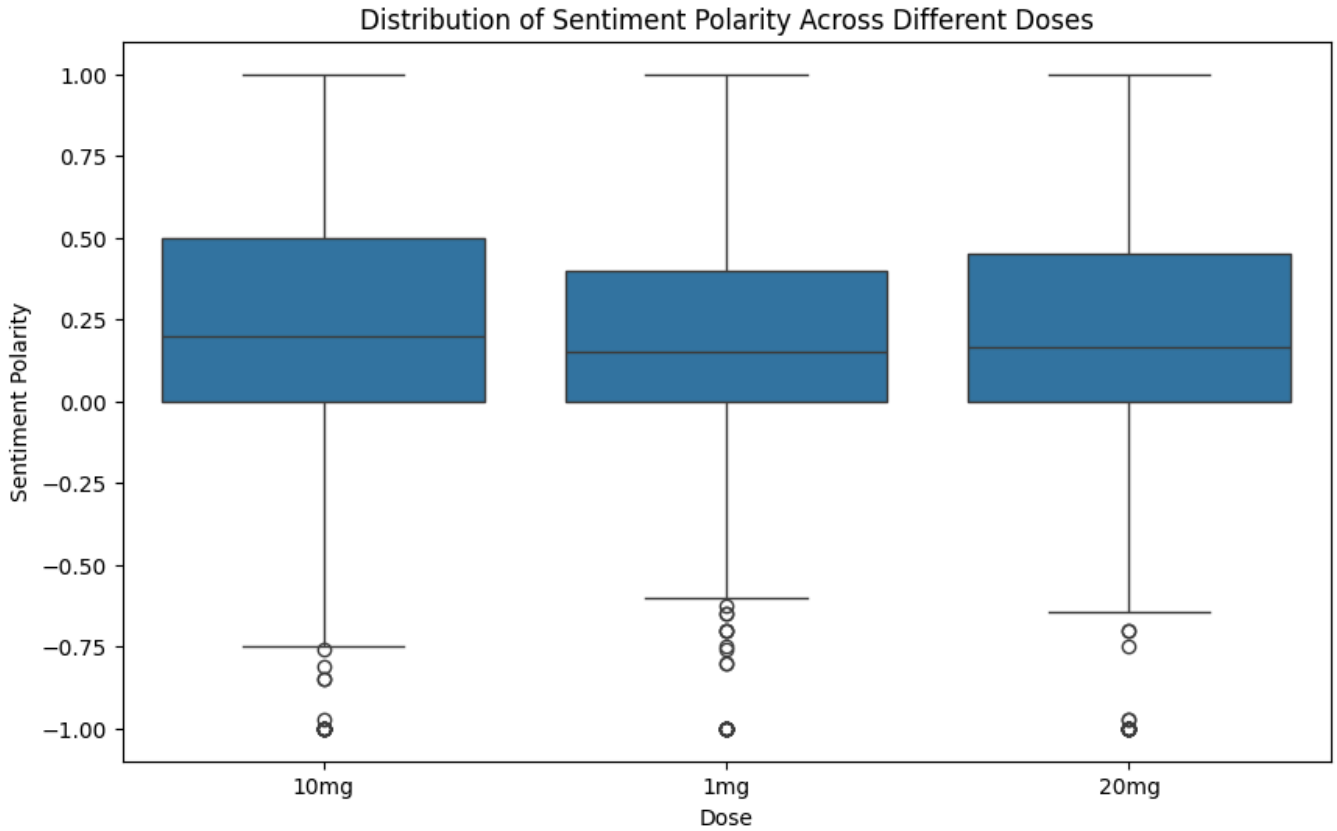
```
# Distribution of reviews across doses...
plt.figure(figsize=(10, 6))
sns.countplot(data=filtered_cleaned_df, x='dose')
plt.title('Distribution of Reviews Across Different Doses')
plt.xlabel('Dose')
plt.ylabel('Number of Reviews')
plt.show()

#  sentiment analysis - so to calculate polarity for each review
filtered_cleaned_df['sentiment_polarity'] = filtered_cleaned_df['review_text'].apply

# Plotting the distribution of sentiment polarity across the diff doses..
plt.figure(figsize=(10, 6))
sns.boxplot(data=filtered_cleaned_df, x='dose', y='sentiment_polarity')
plt.title('Distribution of Sentiment Polarity Across Different Doses')
plt.xlabel('Dose')
plt.ylabel('Sentiment Polarity')
plt.show()

# basic sentiment statistics for each dose
sentiment_stats = filtered_cleaned_df.groupby('dose')['sentiment_polarity'].describe
print(sentiment_stats)
```





dose	count	mean	std	min	25%	50%	75%	max
10mg	2694.0	0.257711	0.354022	-1.0	0.0	0.200000	0.500000	1.0

```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
import string


# Loading dataset
df = filtered_cleaned_df.copy()

# to combine review text and header.
df['full_review'] = df['review_header'] + ' ' + df['review_text']

# Function for preprocessing text
def preprocess_text(text):
    # Normalize text!!
    text = text.lower()

    # Removing punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Tokenize text!!

    tokens = word_tokenize(text)
    # Remove stopwords!!

    tokens = [token for token in tokens if token not in stopwords.words('english')]
    # Lemmatize!!

    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    return ' '.join(tokens)

# Apply preprocessing donee to each review
df['processed_review'] = df['full_review'].apply(preprocess_text)

# Feature Extraction with TF-IDF.
vectorizer = TfidfVectorizer(ngram_range=(1, 2))
X = vectorizer.fit_transform(df['processed_review'])
y = df['dose']

# Splitting datasett
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
```

```
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

```
# Function to find relevant sentences having keywords
def find_relevant_sentences(text, keywords):
    relevant_sentences = []
    sentences = sent_tokenize(text)
    for sentence in sentences:
        for keyword in keywords:
            if keyword in sentence:
                relevant_sentences.append(sentence)
                break  # we match once per sentence
    return relevant_sentences

# Func to extract useful words and phrases that might be relevant to us.
def extract_relevant_phrases(dose_df, keywords):
    lemmatizer = WordNetLemmatizer()
    all_relevant_phrases = []
    for review in dose_df['processed_review']:
        relevant_sentences = find_relevant_sentences(review, keywords)
        for sentence in relevant_sentences:
            words = word_tokenize(sentence)
            words = [lemmatizer.lemmatize(word) for word in words if word not in sto
            tags = nltk.pos_tag(words)
            for word, tag in tags:
                if tag.startswith('JJ') or tag.startswith('NN'):  # Adjectives and N
                    all_relevant_phrases.append(word)
    return Counter(all_relevant_phrases).most_common()

# creating keywords that might depict reactions and sleep quality
keywords_sleep = ['sleep', 'rest', 'insomnia', 'awake', 'night']
keywords_reaction = ['effective', 'useless', 'happy', 'disappointed', 'side effect']

# Analyzing each dose
for dose, group in df.groupby('dose'):
    print(f"Dose: {dose}")
    print("Top Sleep-related Terms:")
    print(extract_relevant_phrases(group, keywords_sleep))
    print("Top Reaction-related Terms:")
    print(extract_relevant_phrases(group, keywords_reaction))
    print("\n")
```

```
Dose: 10mg
Top Sleep-related Terms:
[('sleep', 1154), ('work', 705), ('night', 624), ('great', 588), ('good', 474),
Top Reaction-related Terms:
[('sleep', 114), ('effective', 108), ('work', 97), ('gummies', 95), ('taste', 89


Dose: 1mg
Top Sleep-related Terms:
```

```
[('sleep', 760), ('work', 483), ('night', 360), ('help', 275), ('taste', 261), (
Top Reaction-related Terms:
[('sleep', 69), ('product', 54), ('work', 52), ('effective', 51), ('effect', 38)


Dose: 20mg
Top Sleep-related Terms:
[('sleep', 323), ('work', 216), ('night', 158), ('melatonin', 155), ('help', 144
Top Reaction-related Terms:
[('taste', 34), ('effective', 31), ('work', 29), ('melatonin', 24), ('sleep', 24
```

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

# Predictions
nb_predictions = model.predict(X_test)
```

```
print("Accuracy:", accuracy_score(y_test, nb_predictions))
print("Classification Report:\n", classification_report(y_test, nb_predictions))
```

```
Accuracy: 0.530060120240481
Classification Report:
              precision    recall  f1-score   support

        10mg       0.53      1.00      0.69       528
         1mg       0.00      0.00      0.00       244
        20mg       1.00      0.00      0.01       226

    accuracy                           0.53       998
   macro avg       0.51      0.33      0.23       998
weighted avg       0.51      0.53      0.37       998

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
  _warn_prf(average, modifier, msg_start, len(result))
```

```
from sklearn.linear_model import LogisticRegression

#  Logistic Regression model
log_reg_model = LogisticRegression(max_iter=1000)

# Model training
log_reg_model.fit(X_train, y_train)

# Predictions
log_reg_predictions = log_reg_model.predict(X_test)


print("Logistic Regression Accuracy:", accuracy_score(y_test, log_reg_predictions))
print("Logistic Regression Classification Report:\n", classification_report(y_test,
```

```
Logistic Regression Accuracy: 0.5811623246492986
Logistic Regression Classification Report:
              precision    recall  f1-score   support

        10mg       0.57      0.94      0.71       528
         1mg       0.52      0.15      0.23       244
        20mg       0.81      0.21      0.33       226

    accuracy                           0.58       998
   macro avg       0.63      0.43      0.42       998
weighted avg       0.61      0.58      0.51       998
```

```
from sklearn.svm import SVC

# SVM model
svm_model = SVC(kernel='linear')  # You can experiment with different kernels like '

# model training
svm_model.fit(X_train, y_train)

# Predictions
svm_predictions = svm_model.predict(X_test)


print("SVM Accuracy:", accuracy_score(y_test, svm_predictions))
print("SVM Classification Report:\n", classification_report(y_test, svm_predictions)
```

```
SVM Accuracy: 0.5971943887775552
SVM Classification Report:
              precision    recall  f1-score   support

        10mg       0.59      0.94      0.72       528
         1mg       0.52      0.18      0.27       244
        20mg       0.85      0.25      0.39       226
```

```
          accuracy                           0.60       998
         macro avg       0.65      0.46      0.46       998
      weighted avg       0.63      0.60      0.53       998
```

```
from sklearn.tree import DecisionTreeClassifier

# Decision Tree Classifier
decision_tree_classifier = DecisionTreeClassifier(max_depth=10) # Example max_depth

# Training the model
decision_tree_classifier.fit(X_train, y_train)

#  predictions
dt_predictions = decision_tree_classifier.predict(X_test)


print("Decision Tree Accuracy:", accuracy_score(y_test, dt_predictions))
```

```
Decision Tree Accuracy: 0.5681362725450901
```

```
# on df['processed_review'] and df['dose']

from sklearn.model_selection import train_test_split

X_raw = df['processed_review']
y_raw = df['dose']
X_train_raw, X_test_raw, y_train_raw, y_test_raw = train_test_split(X_raw, y_raw, te

#  X_train_raw and y_train_raw for grid search
grid_search.fit(X_train_raw, y_train_raw)

# After fitting the raw text test data for prediction
rf_predictions = grid_search.predict(X_test_raw)

# accuracy and other metrics using the true labels from your test set
print("\nRandomForest with Grid Search Accuracy:", accuracy_score(y_test_raw, rf_pre
print("RandomForest with Grid Search Classification Report:\n", classification_repor
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits

RandomForest with Grid Search Accuracy: 0.5961923847695391
RandomForest with Grid Search Classification Report:
               precision    recall  f1-score   support

        10mg       0.59      0.91      0.71       528
         1mg       0.53      0.16      0.24       244
        20mg       0.70      0.34      0.46       226
```

```
          accuracy                           0.60       998
         macro avg       0.61      0.47      0.47       998
      weighted avg       0.60      0.60      0.54       998
```

```
#the accuracy scores
accuracy_scores = {
    "Naive Bayes": accuracy_score(y_test_raw, nb_predictions),
    "Decision Tree": accuracy_score(y_test_raw, dt_predictions),
```