

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.stem.porter import PorterStemmer
nltk.download('stopwords')
from nltk.corpus import stopwords
STOPWORDS = set(stopwords.words('english'))

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score
from wordcloud import WordCloud
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
import pickle
import re
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
%pip install wordcloud
```

```
Requirement already satisfied: wordcloud in /usr/local/lib/python3.10/dist-packages (1.9.3)
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.10/dist-packages (from wordcloud) (1.25.2)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from wordcloud) (9.4.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from wordcloud) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (4.49.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (23.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->wordcloud)
```

```
data = pd.read_csv(r"/content/amazon_alexa.tsv", delimiter = '\t', quoting = 3)
```

```
print(f"Dataset shape : {data.shape}")
```

```
Dataset shape : (3150, 5)
```

```
data.head()
```

	rating	date	variation	verified_reviews	feedback
0	5	31-Jul-18	Charcoal Fabric	Love my Echo!	1
1	5	31-Jul-18	Charcoal Fabric	Loved it!	1
2	4	31-Jul-18	Walnut Finish	"Sometimes while playing a game, you can answe...	1
3	5	31-Jul-18	Charcoal Fabric	"I have had a lot of fun with this thing. My 4...	1
4	5	31-Jul-18	Charcoal Fabric	Music	1

```
print(f"Feature names : {data.columns.values}")
```

```
Feature names : ['rating' 'date' 'variation' 'verified_reviews' 'feedback']
```

```
data.isnull().sum()
```

```
rating      0
date        0
variation    0
verified_reviews  0
feedback     0
dtype: int64
```

```
#Getting the record where 'verified_reviews' is null
```

```
data[data['verified_reviews'].isna() == True]
```

```
rating date variation verified_reviews feedback
```

```
#We will drop the null record
```

```
data.dropna(inplace=True)
```

```
print(f"Dataset shape after dropping null values : {data.shape}")
```

```
Dataset shape after dropping null values : (3150, 5)
```

```
#Creating a new column 'length' that will contain the length of the string in 'verified_reviews' column
```

```
data['length'] = data['verified_reviews'].apply(len)
```

```
data.head()
```

	rating	date	variation	verified_reviews	feedback	length
0	5	31-Jul-18	Charcoal Fabric	Love my Echo!	1	13
1	5	31-Jul-18	Charcoal Fabric	Loved it!	1	9
2	4	31-Jul-18	Walnut Finish	"Sometimes while playing a game, you can answe...	1	197
3	5	31-Jul-	Charcoal	"I have had a lot of fun with this thing.	1	100

```
#Randomly checking for 10th record
```

```
print(f"'verified_reviews' column value: {data.iloc[10]['verified_reviews']}") #Original value
```

```
print(f"Length of review : {len(data.iloc[10]['verified_reviews'])}") #Length of review using len()
```

```
print(f"'length' column value : {data.iloc[10]['length']}") #Value of the column 'length'
```

```
'verified_reviews' column value: "I sent it to my 85 year old Dad, and he talks to it constantly."
Length of review : 65
'length' column value : 65
```

```
data.dtypes
```

```
rating      int64
date        object
variation    object
verified_reviews  object
feedback     int64
length      int64
dtype: object
```

```
len(data)
```

```
3150
```

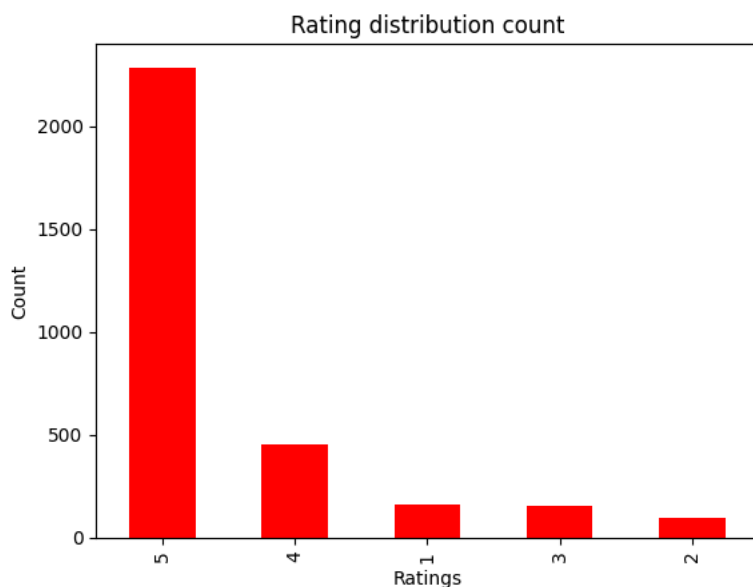
```
#Distinct values of 'rating' and its count
```

```
print(f"Rating value count: \n{data['rating'].value_counts()}")
```

```
Rating value count:
5    2286
4     455
1     161
3     152
2      96
Name: rating, dtype: int64
```

```
#Bar plot to visualize the total counts of each rating
```

```
data['rating'].value_counts().plot.bar(color = 'red')
plt.title('Rating distribution count')
plt.xlabel('Ratings')
plt.ylabel('Count')
plt.show()
```



#Finding the percentage distribution of each rating - we'll divide the number of records for each rating by total number of records

```
print(f"Rating value count - percentage distribution: \n{round(data['rating'].value_counts()/data.shape[0]*100,2)}")
```

```
Rating value count - percentage distribution:
```

```
5    72.57
```

```
4    14.44
```

```
1     5.11
```

```
3     4.83
```

```
2     3.05
```

```
Name: rating, dtype: float64
```

```
fig = plt.figure(figsize=(7,7))
```

```
colors = ('red', 'green', 'blue','orange','yellow')
```

```
wp = {'linewidth':1, "edgecolor":'black'}
```

```
tags = data['rating'].value_counts()/data.shape[0]
```

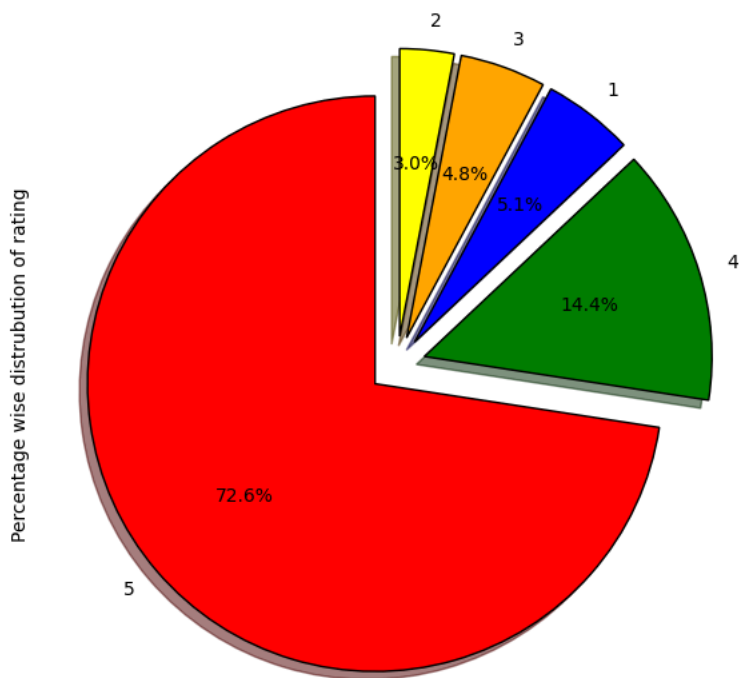
```
explode=(0.1,0.1,0.1,0.1,0.1)
```

```
tags.plot(kind='pie', autopct="%1.1f%%", shadow=True, colors=colors, startangle=90, wedgeprops=wp, explode=explode, label='Percentage w:
```

```
from io import BytesIO
```

```
graph = BytesIO()
```

```
fig.savefig(graph, format="png")
```



#Distinct values of 'feedback' and its count

```
print(f"Feedback value count: \n{data['feedback'].value_counts()}")
```

```
Feedback value count:
1    2893
0     257
Name: feedback, dtype: int64
```

#Extracting the 'verified_reviews' value for one record with feedback = 0

```
review_0 = data[data['feedback'] == 0].iloc[1]['verified_reviews']
print(review_0)
```

Sound is terrible if u want good music too get a bose

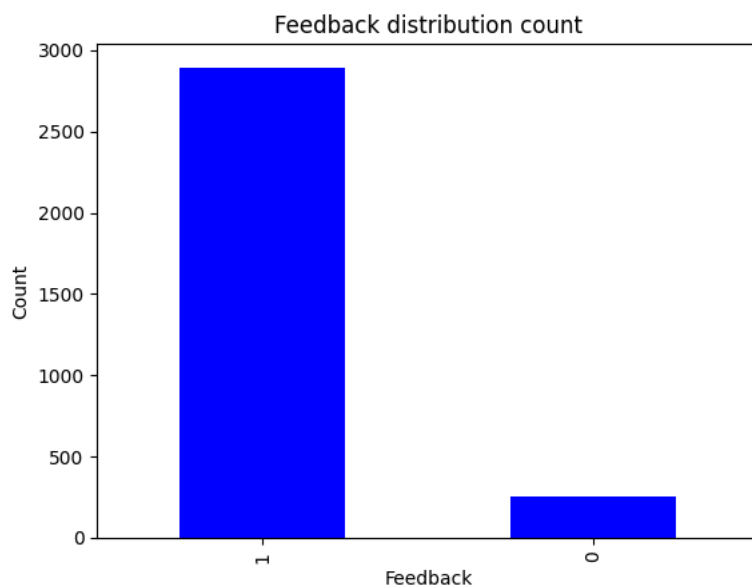
#Extracting the 'verified_reviews' value for one record with feedback = 1

```
review_1 = data[data['feedback'] == 1].iloc[1]['verified_reviews']
print(review_1)
```

Loved it!

#Bar graph to visualize the total counts of each feedback

```
data['feedback'].value_counts().plot.bar(color = 'blue')
plt.title('Feedback distribution count')
plt.xlabel('Feedback')
plt.ylabel('Count')
plt.show()
```



#Finding the percentage distribution of each feedback - we'll divide the number of records for each feedback by total number of records

```
print(f"Feedback value count - percentage distribution: \n{round(data['feedback'].value_counts()/data.shape[0]*100,2)}")
```

```
Feedback value count - percentage distribution:
```

```
1    91.84
```

```
0     8.16
```

```
Name: feedback, dtype: float64
```

```
#Feedback = 0
```

```
data[data['feedback'] == 0]['rating'].value_counts()
```

```
1    161
```

```
2     96
```

```
Name: rating, dtype: int64
```

```
#Feedback = 1
```

```
data[data['feedback'] == 1]['rating'].value_counts()
```

```
5    2286
```

```
4     455
```

```
3     152
```

```
Name: rating, dtype: int64
```

```
#Distinct values of 'variation' and its count
```

```
print(f"Variation value count: \n{data['variation'].value_counts()}")
```

```
Variation value count:
```

```
Black Dot    516
```

```
Charcoal Fabric    430
```

```
Configuration: Fire TV Stick    350
```

```
Black Plus    270
```

```
Black Show    265
```

```
Black    261
```

```
Black Spot    241
```

```
White Dot    184
```

```
Heather Gray Fabric    157
```

```
White Spot    109
```

```
White    91
```

```
Sandstone Fabric    90
```

```
White Show    85
```

```
White Plus    78
```

```
Oak Finish    14
```

```
Walnut Finish    9
```

```
Name: variation, dtype: int64
```

```
#Bar graph to visualize the total counts of each variation
```

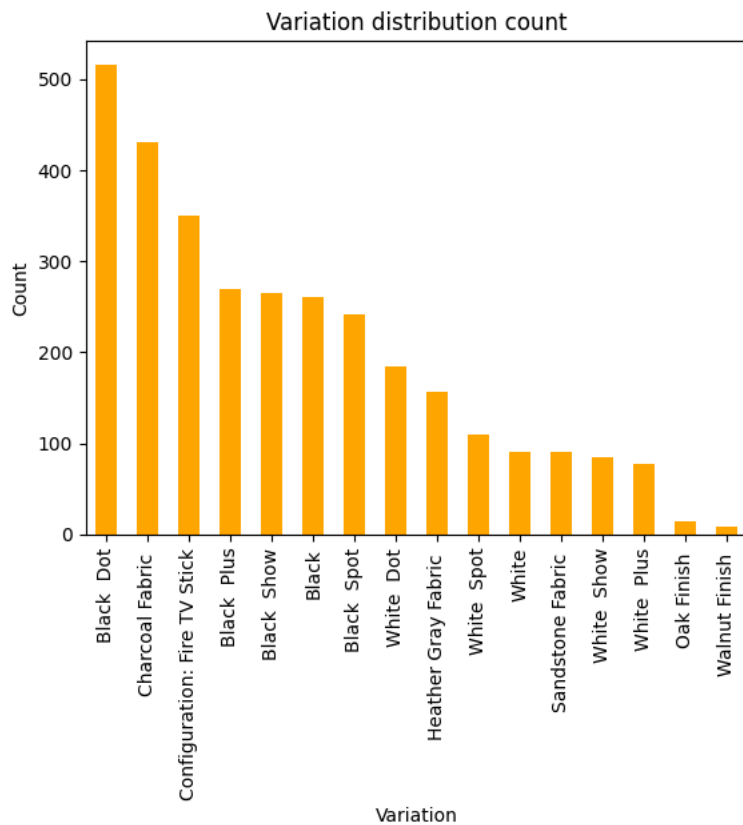
```
data['variation'].value_counts().plot.bar(color = 'orange')
```

```
plt.title('Variation distribution count')
```

```
plt.xlabel('Variation')
```

```
plt.ylabel('Count')
```

```
plt.show()
```



#Finding the percentage distribution of each variation - we'll divide the number of records for each variation by total number of records

```
print(f"Variation value count - percentage distribution: \n{round(data['variation'].value_counts()/data.shape[0]*100,2)}")
```

Variation value count - percentage distribution:

Black Dot	16.38
Charcoal Fabric	13.65
Configuration: Fire TV Stick	11.11
Black Plus	8.57
Black Show	8.41
Black	8.29
Black Spot	7.65
White Dot	5.84
Heather Gray Fabric	4.98
White Spot	3.46
White	2.89
Sandstone Fabric	2.86
White Show	2.70
White Plus	2.48
Oak Finish	0.44
Walnut Finish	0.29

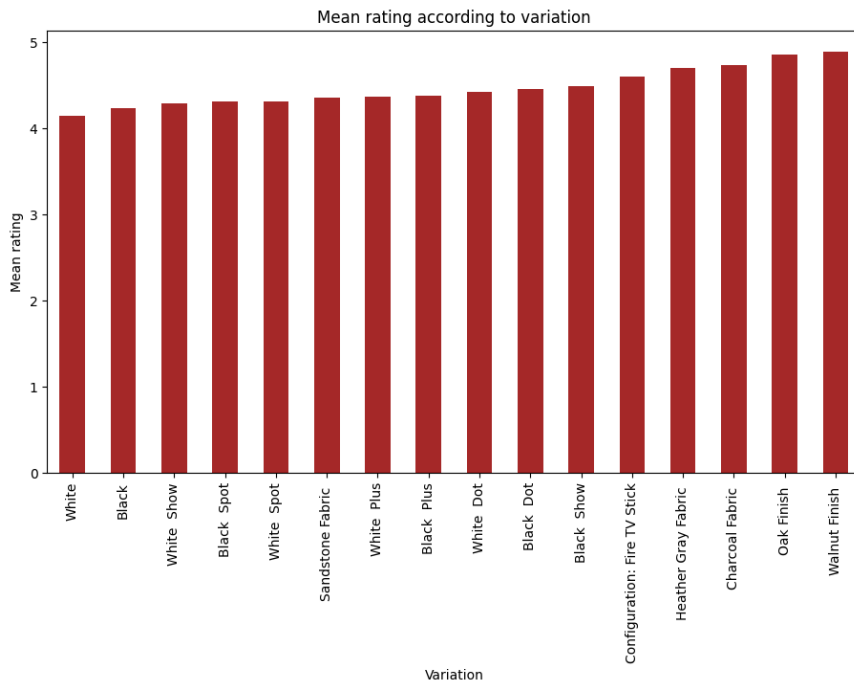
Name: variation, dtype: float64

```
data.groupby('variation')['rating'].mean()
```

variation	
Black	4.233716
Black Dot	4.453488
Black Plus	4.370370
Black Show	4.490566
Black Spot	4.311203
Charcoal Fabric	4.730233
Configuration: Fire TV Stick	4.591429
Heather Gray Fabric	4.694268
Oak Finish	4.857143
Sandstone Fabric	4.355556
Walnut Finish	4.888889
White	4.142857
White Dot	4.423913
White Plus	4.358974
White Show	4.282353
White Spot	4.311927

Name: rating, dtype: float64

```
data.groupby('variation')['rating'].mean().sort_values().plot.bar(color = 'brown', figsize=(11, 6))
plt.title("Mean rating according to variation")
plt.xlabel('Variation')
plt.ylabel('Mean rating')
plt.show()
```

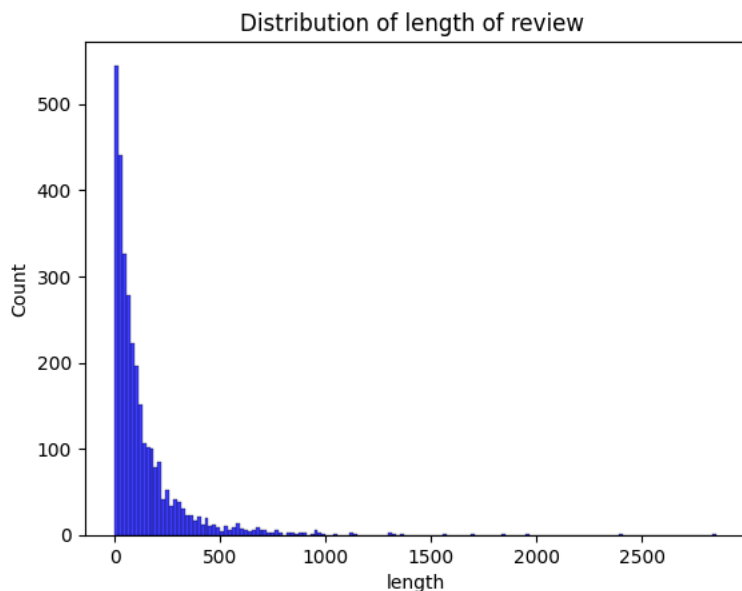


```
data['length'].describe()
```

```
count    3150.000000
mean      132.673651
std       182.526953
min         1.000000
25%       30.000000
50%       74.000000
75%      166.000000
max      2853.000000
Name: length, dtype: float64
```

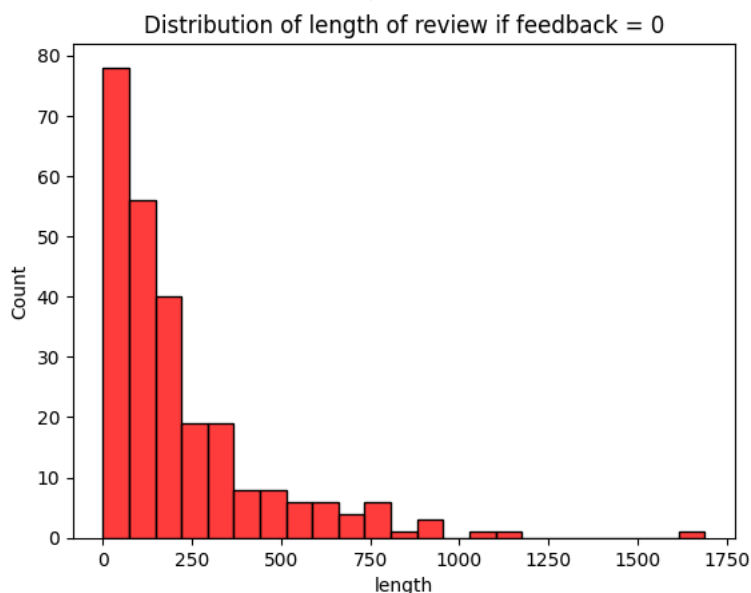
```
sns.histplot(data['length'],color='blue').set(title='Distribution of length of review ')
```

```
[Text(0.5, 1.0, 'Distribution of length of review ')]
```



```
sns.histplot(data[data['feedback']==0]['length'],color='red').set(title='Distribution of length of review if feedback = 0')
```

```
[Text(0.5, 1.0, 'Distribution of length of review if feedback = 0')]
```



```
cv = CountVectorizer(stop_words='english')
words = cv.fit_transform(data.verified_reviews)
```

```
# Combine all reviews
reviews = " ".join([review for review in data['verified_reviews']])
```

```
# Initialize wordcloud object
wc = WordCloud(background_color='white', max_words=50)
```

```
# Generate and plot wordcloud
plt.figure(figsize=(10,10))
plt.imshow(wc.generate(reviews))
plt.title('Wordcloud for all reviews', fontsize=10)
plt.axis('off')
plt.show()
```




```
# Combine all reviews for each feedback category and splitting them into individual words
neg_reviews = " ".join([review for review in data[data['feedback'] == 0]['verified_reviews']])
neg_reviews = neg_reviews.lower().split()

pos_reviews = " ".join([review for review in data[data['feedback'] == 1]['verified_reviews']])
pos_reviews = pos_reviews.lower().split()

#Finding words from reviews which are present in that feedback category only
unique_negative = [x for x in neg_reviews if x not in pos_reviews]
unique_negative = " ".join(unique_negative)

unique_positive = [x for x in pos_reviews if x not in neg_reviews]
unique_positive = " ".join(unique_positive)

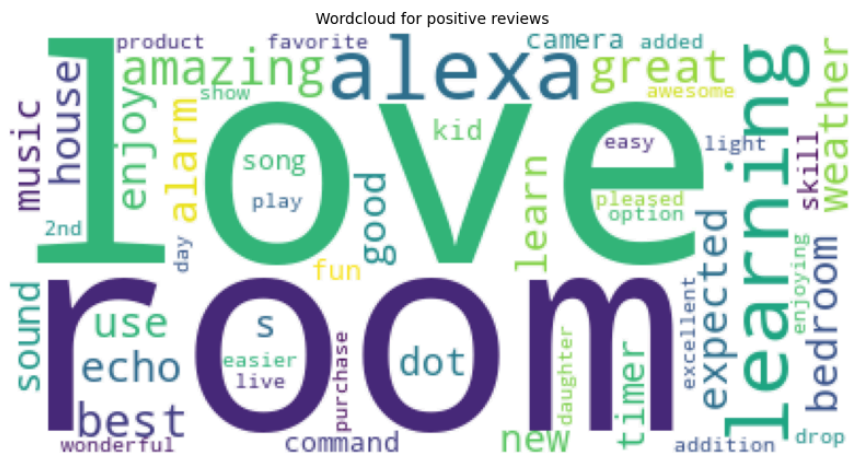
wc = WordCloud(background_color='white', max_words=50)

# Generate and plot wordcloud
plt.figure(figsize=(10,10))
plt.imshow(wc.generate(unique_negative))
plt.title('Wordcloud for negative reviews', fontsize=10)
plt.axis('off')
plt.show()
```



```
wc = WordCloud(background_color='white', max_words=50)

# Generate and plot wordcloud
plt.figure(figsize=(10,10))
plt.imshow(wc.generate(unique_positive))
plt.title('Wordcloud for positive reviews', fontsize=10)
plt.axis('off')
plt.show()
```



```
corpus = []
stemmer = PorterStemmer()
for i in range(0, data.shape[0]):
    review = re.sub('[^a-zA-Z]', ' ', data.iloc[i]['verified_reviews'])
    review = review.lower().split()
    review = [stemmer.stem(word) for word in review if not word in STOPWORDS]
    review = ' '.join(review)
    corpus.append(review)

cv = CountVectorizer(max_features = 2500)

#Storing independent and dependent variables in X and y
X = cv.fit_transform(corpus).toarray()
y = data['feedback'].values

print(f"X shape: {X.shape}")
print(f"y shape: {y.shape}")

X shape: (3150, 2500)
y shape: (3150,)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 15)

print(f"X train: {X_train.shape}")
print(f"y train: {y_train.shape}")
print(f"X test: {X_test.shape}")
print(f"y test: {y_test.shape}")

X train: (2205, 2500)
y train: (2205,)
X test: (945, 2500)
y test: (945,)

print(f"X train max value: {X_train.max()}")
print(f"X test max value: {X_test.max()}")

X train max value: 12
X test max value: 10

scaler = MinMaxScaler()

X_train_scl = scaler.fit_transform(X_train)
X_test_scl = scaler.transform(X_test)

#Fitting scaled X_train and y_train on Random Forest Classifier
model_rf = RandomForestClassifier()
model_rf.fit(X_train_scl, y_train)

▼ RandomForestClassifier
RandomForestClassifier()
```

```
#Accuracy of the model on training and testing data

print("Training Accuracy :", model_rf.score(X_train_scl, y_train))
print("Testing Accuracy :", model_rf.score(X_test_scl, y_test))

Training Accuracy : 0.9941043083900227

#Predicting on the test set
y_preds = model_rf.predict(X_test_scl)

#Confusion Matrix
cm = confusion_matrix(y_test, y_preds)

cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model_rf.classes_)
cm_display.plot()
plt.show()
```

