```python
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.
```

```python
%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) (1.26.4)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from wordcloud) (10.4.0)
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) (1.3.0)
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) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (24.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (3.2.0)
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->wordclouc
```

```python
#Load the data

data = pd.read_csv(r"/content/2111.txt", delimiter = '\t', quoting = 3)

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

```

```
Dataset shape : (2769, 5)
```

```python
data.head()
```

	feedback	variation	date	rating	verified_reviews
0	0	Mamaearth-Onion-Growth-Control-Redensyl	2019-09-06 00:00:00	1	I bought this hair oil after viewing so many g...
1	1	Mamaearth-Onion-Growth-Control-Redensyl	2019-08-14 00:00:00	5	"Used This Mama Earth Newly Launched Onion Oil...
2	0	Mamaearth-Onion-Growth-Control-Redensyl	2019-10-19 00:00:00	1	So bad product...My hair falling increase too ...
3	0	Mamaearth-Onion-Growth-Control-Redensyl	2019-09-16 00:00:00	1	Product just smells similar to navarathna hair...
4	1	Mamaearth-Onion-Growth-Control-Redensyl	2019-08-18 00:00:00	5	I have been trying different onion oil for my ...

Next steps: Generate code with data View recommended plots New interactive sheet

```python
#Column names

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

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

```python
#Check for null values

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

	0
feedback	0
variation	0
date	0
rating	0
verified_reviews	6

```

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

	0
feedback	0
variation	0
date	0
rating	0
verified_reviews	6

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

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

	feedback	variation	date	rating	verified_reviews
2238	1	Tata-Tea-Gold-500g	2018-03-03 00:00:00	4	NaN
2248	1	Tata-Tea-Gold-500g	2018-03-03 00:00:00	4	NaN
2621	1	Mysore-Sandal-Bathing-Soap-125g	2020-05-22 00:00:00	4	NaN
2624	1	Mysore-Sandal-Bathing-Soap-125g	2020-09-24 00:00:00	5	NaN
2631	1	Mysore-Sandal-Bathing-Soap-125g	2020-05-22 00:00:00	4	NaN
2634	1	Mysore-Sandal-Bathing-Soap-125g	2020-09-24 00:00:00	5	NaN

```
#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 : (2763, 5)
```

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

```
data.head()
```

	feedback	variation	date	rating	verified_reviews	length
0	0	Mamaearth-Onion-Growth-Control-Redensyl	2019-09-06 00:00:00	1	I bought this hair oil after viewing so many g...	477
1	1	Mamaearth-Onion-Growth-Control-Redensyl	2019-08-14 00:00:00	5	"Used This Mama Earth Newly Launched Onion Oil...	497
2	0	Mamaearth-Onion-Growth-Control-Redensyl	2019-10-19 00:00:00	1	So bad product...My hair falling increase too ...	149

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
#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 bought this hair oil after viewing so many good comments. But this product is not good enough.Fir
Length of review : 477
'length' column value : 477
```

```
data.dtypes
```

	0
feedback	int64
variation	object
date	object
rating	int64
verified_reviews	object
length	int64

```
dtype: object
```

analysing rating column

```
len(data)
```

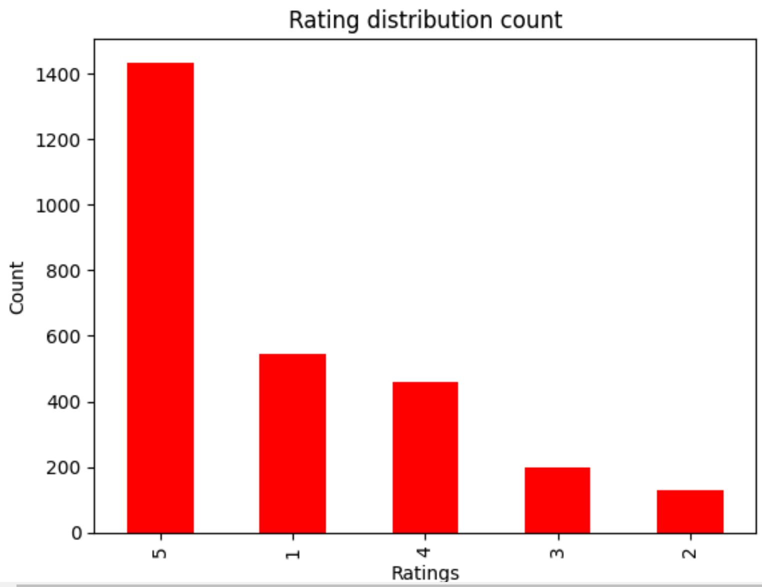
```
2763
```

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

```
Rating value count:
rating
5    1433
1     544
4     460
3     198
2     128
Name: count, dtype: int64
```

Let's plot the above values in a bar graph

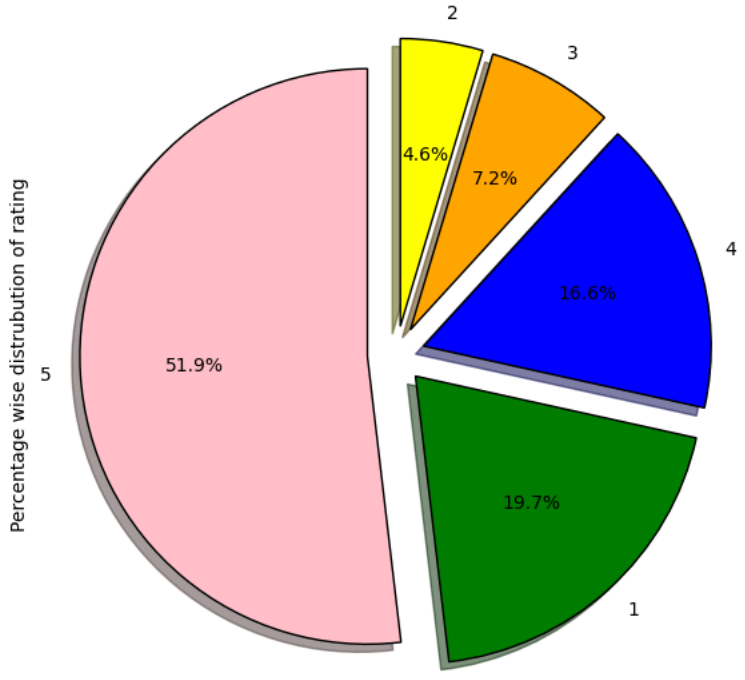
```
#Bar plot to visualize the total counts of each rating

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:
rating
5    51.86
1    19.69
4    16.65
3     7.17
2     4.63
Name: count, dtype: float64
```

```

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

colors = ('pink', '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

from io import  BytesIO

graph = BytesIO()

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



Analyzing 'feedback' column

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

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

```
Feedback value count:
feedback
1    2091
0     672
Name: count, 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)
```

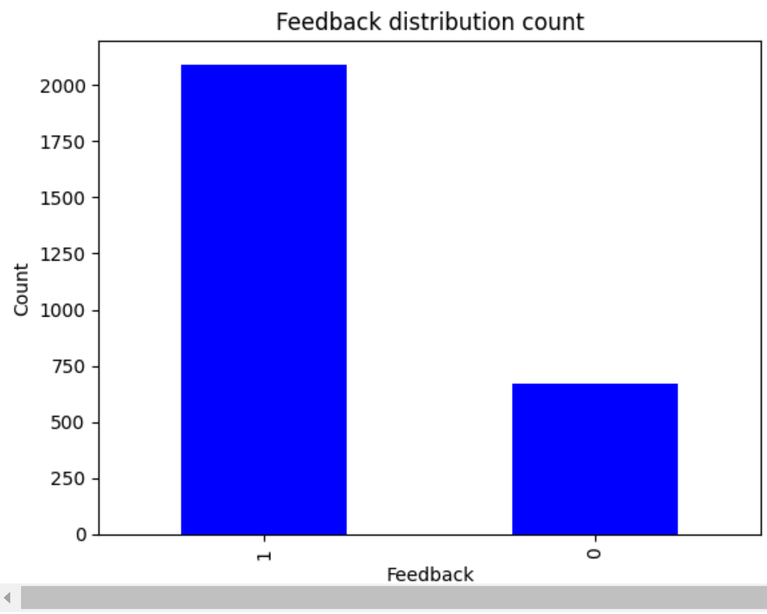
```
So bad product...My hair falling increase too much..I order shampoo mask and oil.. nothing stop hairfallAfter 3 to 4 wash my hair fa
```

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

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

```
I have been trying different onion oil for my hair as my hair is not very healthy. This product has literally changed the texture of
```

From the above 2 examples we can see that feedback 0 is negative review and 1 is positive review

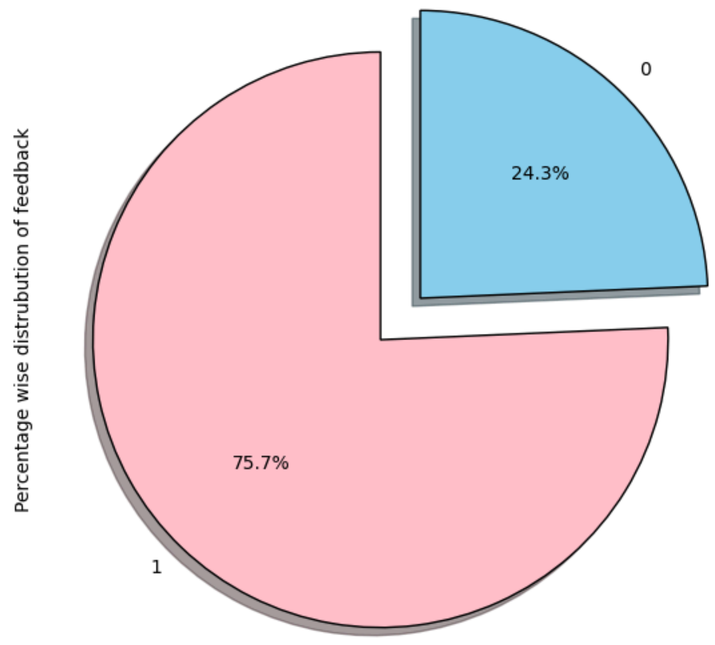
```
#Bar graph to visualize the total counts of each feedback

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 recor

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

```
Feedback value count - percentage distribution:
feedback
1    75.68
0    24.32
Name: count, dtype: float64
```

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

colors = ('pink', 'skyblue')

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

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

explode=(0.1,0.1)

tags.plot(kind='pie', autopct="%1.1f%%", shadow=True, colors=colors, startangle=90, wedgeprops=wp, explode=explode, label='Percentage
<Axes: ylabel='Percentage wise distrubution of feedback'>
```



```

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

	count
rating	
1	544
2	128

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

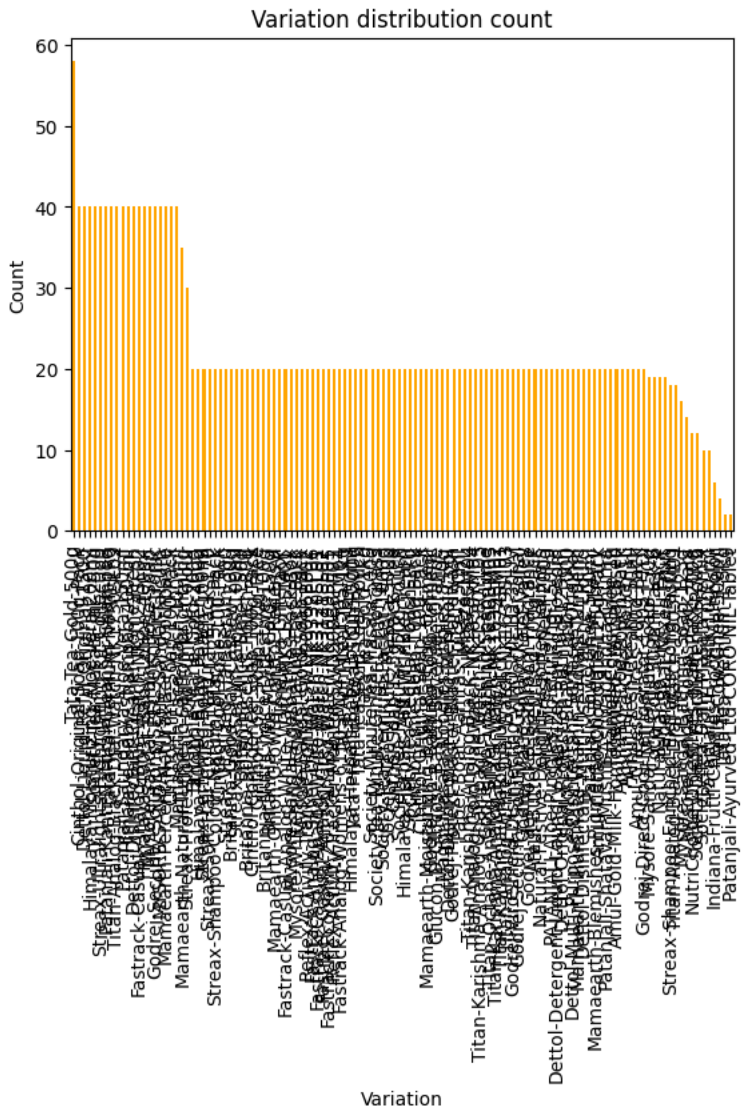
	count
rating	
5	1433
4	460
3	198

If rating of a review is 1 or 2 then the feedback is 0 (negative) and if the rating is 3, 4 or 5 then the feedback is 1 (positive).

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

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

```
Variation value count:
variation
Tata-Tea-Gold-500g                          58
Cinthol-Original-Soap-100g-Pack             40
Dettol-Liquid-Refill-Original-1500          40
Himalaya-Moisturizing-Aloe-Vera-200ml       40
Society-Tea-Masala-Jar-250g                 40
                                            ..
Patanjali-UHT-Milk-1000-ml                  10
Indiana-Frutti-Cherries-Frooti-Multicolor    6
Amul-Cow-Ghee-500ml                          4
Tata-Tea-Premium-1-5kg                       2
Patanjali-Ayurved-Ltd-CORO-NIL-Tablet        2
Name: count, Length: 122, 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 rec

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

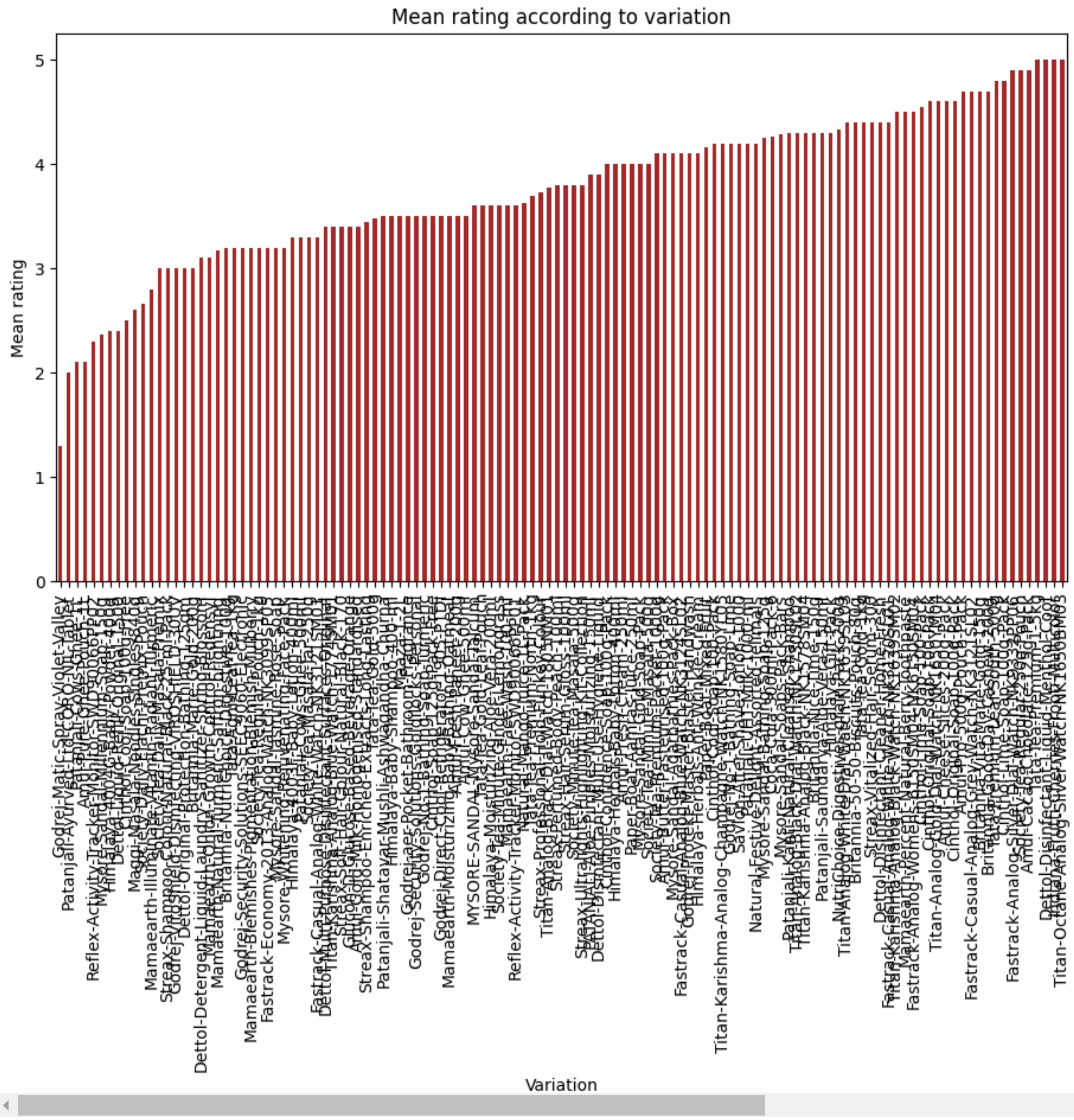
```
Variation value count - percentage distribution:
variation
Tata-Tea-Gold-500g                          2.10
Cinthol-Original-Soap-100g-Pack             1.45
Dettol-Liquid-Refill-Original-1500          1.45
Himalaya-Moisturizing-Aloe-Vera-200ml       1.45
Society-Tea-Masala-Jar-250g                 1.45
                                            ...
Patanjali-UHT-Milk-1000-ml                  0.36
Indiana-Frutti-Cherries-Frooti-Multicolor   0.22
Amul-Cow-Ghee-500ml                         0.14
Tata-Tea-Premium-1-5kg                      0.07
Patanjali-Ayurved-Ltd-CORO-NIL-Tablet       0.07
Name: count, Length: 122, dtype: float64
```

```
#Mean rating according to variation
data.groupby('variation')['rating'].mean()
```

variation	rating
Amul-Butter-Pasteurised-100g-Pack	4.1
Amul-Cacao-Chocolate-125g-Pack	4.9
Amul-Cheese-Slices-200g-Pack	4.6
Amul-Cow-Ghee-500ml	3.5
Amul-Fresh-Cream-250ml	4.0
...	...
Titan-Karishma-Analog-Black-NK1578SM04	4.3
Titan-Karishma-Analog-Black-Watch-NK1639SM02	4.5
Titan-Karishma-Analog-Blue-Watch-1774SM01	3.4
Titan-Karishma-Analog-Champagne-Watch-NK1580YL05	4.2
Titan-Octane-Analog-Silver-Watch-NK1650BM03	5.0

122 rows × 1 columns

```
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()
```



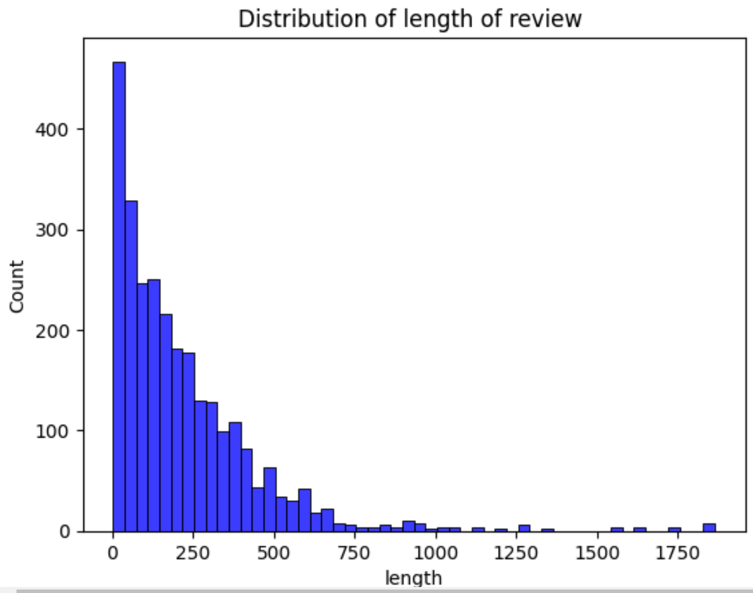
Analyzing 'verified_reviews' column This column contains the textual review given by the user for a variation for the product.

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

```
              length
count    2763.000000
mean      222.631560
std       238.461988
min         2.000000
25%        57.500000
50%       159.000000
75%       312.000000
max      1866.000000
```

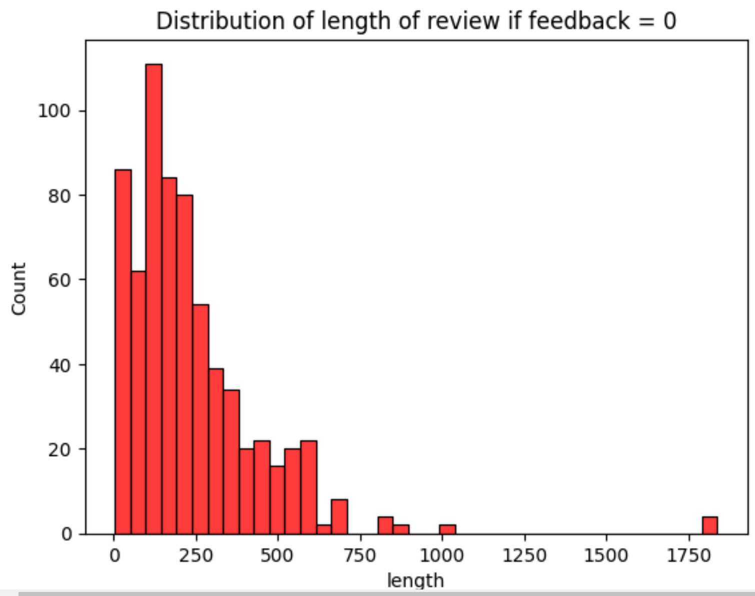
```
sns.histplot(data['length'],color='blue').set(title='Distribution of length of review ')
```

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



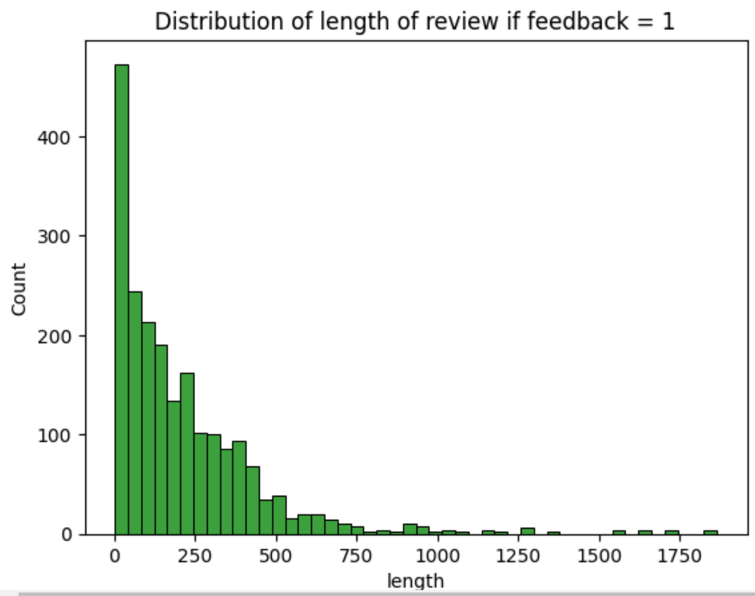
```

#Length analysis when feedback is 0 (negative)

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')]
```

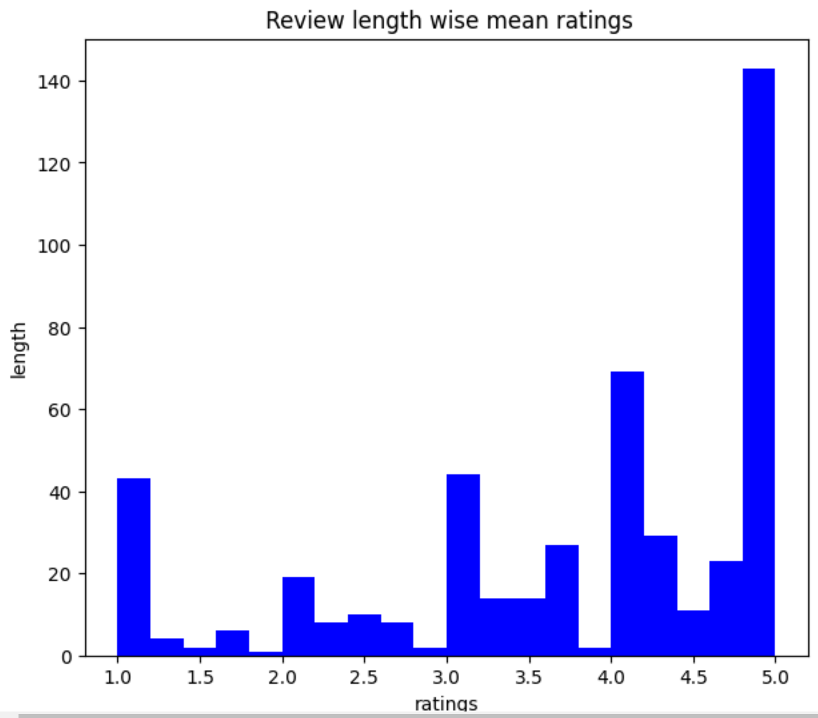


```
sns.histplot(data[data['feedback']==1]['length'],color='green').set(title='Distribution of length of review if feedback = 1')
```

```
[Text(0.5, 1.0, 'Distribution of length of review if feedback = 1')]
```

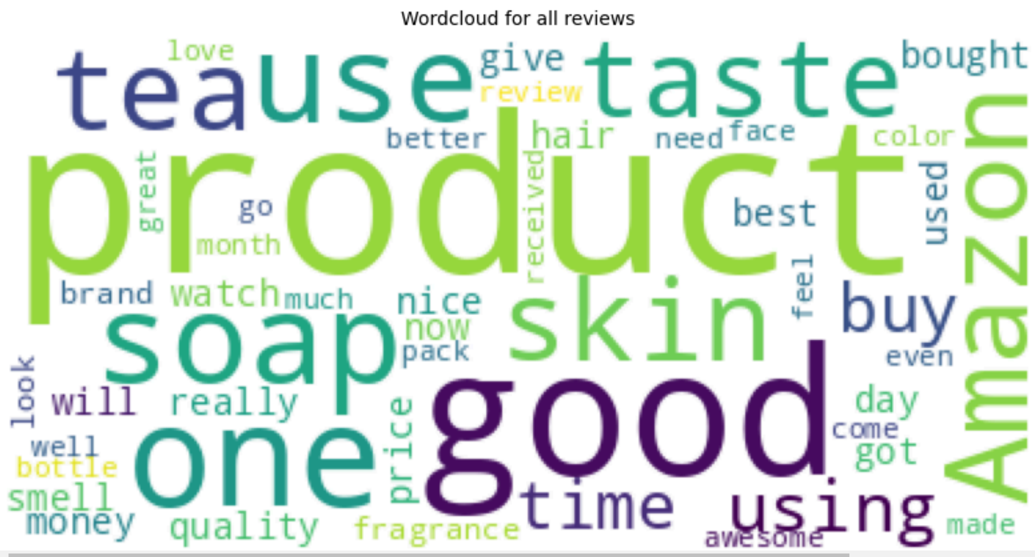


```
#Lengthwise mean rating

data.groupby('length')['rating'].mean().plot.hist(color = 'blue', figsize=(7, 6), bins = 20)
plt.title(" Review length wise mean ratings")
plt.xlabel('ratings')
plt.ylabel('length')
plt.show()
```



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

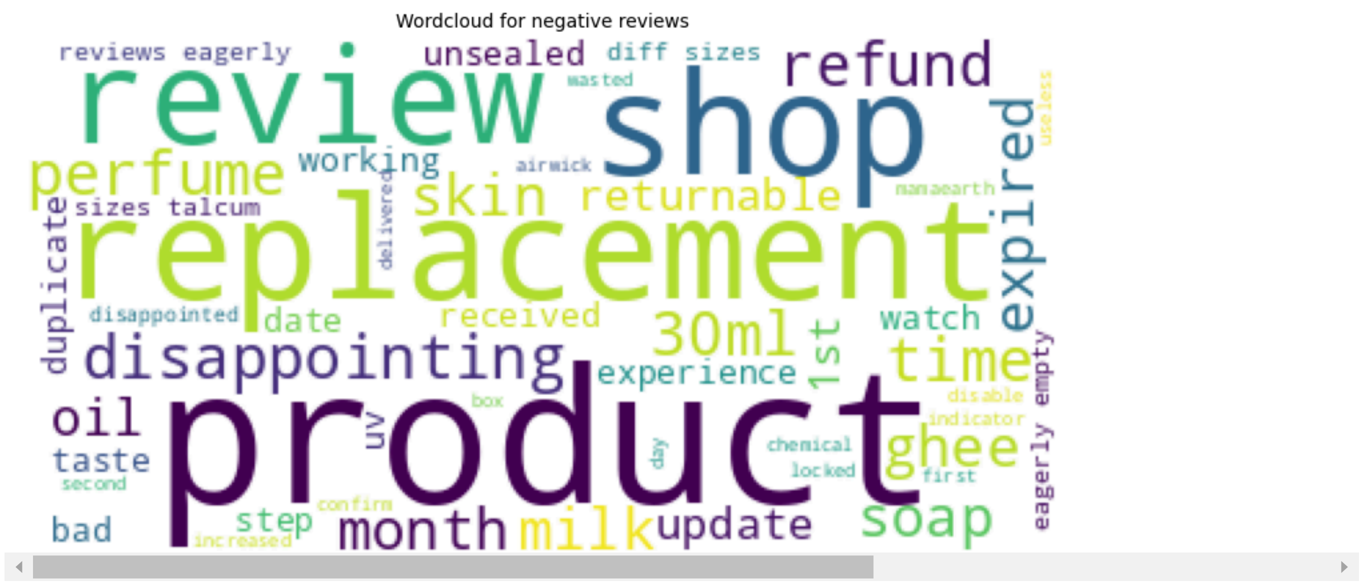
```
# Combine all reviews
reviews = " ".join([review for review in data['verified_reviews']])

# 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()
```



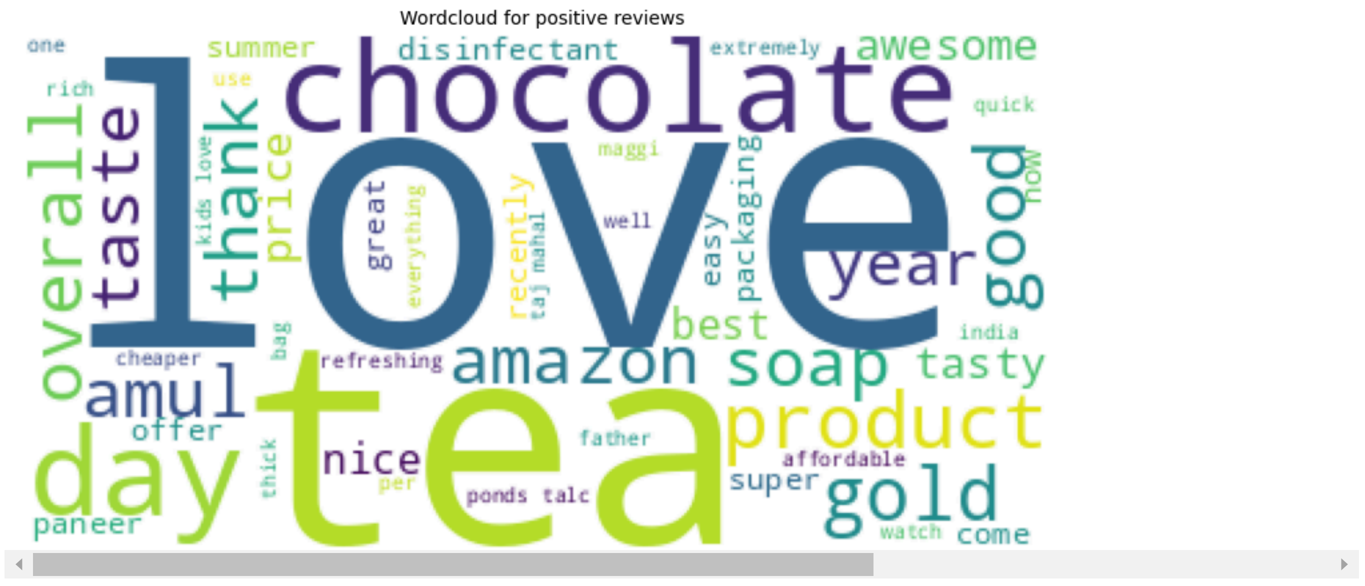
```
# Combine all reviews for each feedback category and splitting them into individual words
neg_reviews = " ".join([review for review in data[data['feedback'] == 0]['verified_reviews']])
neg_reviews = neg_reviews.lower().split()

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()

```



Preprocessing and Modelling To build the corpus from the 'verified_reviews' we perform the following -

1. Replace any non alphabet characters with a space
2. Covert to lower case and split into words
3. Iterate over the individual words and if it is not a stopword then add the stemmed form of the word to the corpus

```
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)
```

```
#using Count Vectorizer to create bag of words

cv = CountVectorizer(max_features = 2500)

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

```
#Saving the Count Vectorizer
pickle.dump(cv, open("/content/sample_data/countVectorizer.pkl", 'wb'))
```

```
#Checking the shape of X and y

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

```
X shape: (2763, 2500)
y shape: (2763,)
```

```
#Splitting data into train and test set with 30% data with testing.

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: (1934, 2500)
y train: (1934,)
X test: (829, 2500)
y test: (829,)
```

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

```
X train max value: 19
X test max value: 16
```

```
#We'll scale X_train and X_test so that all values are between 0 and 1.

scaler = MinMaxScaler()

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

```
#Saving the scaler model
pickle.dump(scaler, open("/content/sample_data/scaler.pkl", 'wb'))
```

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

```
▾ RandomForestClassifier
RandomForestClassifier()
```

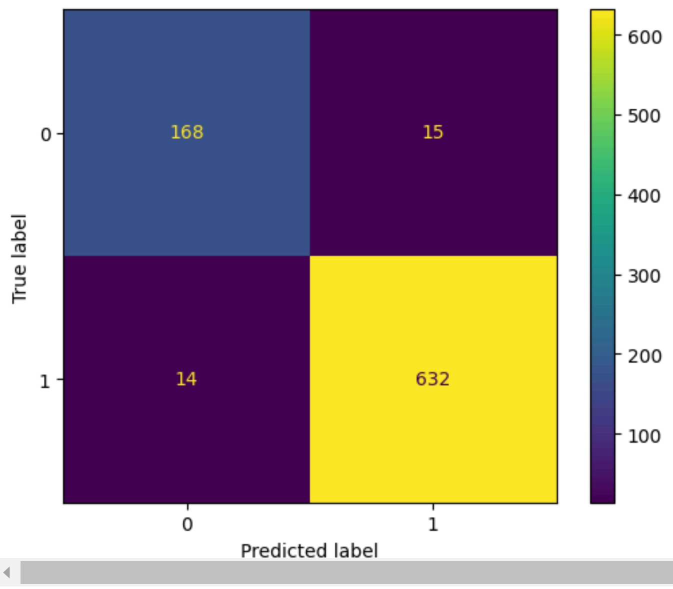
```
#Accuracy of the model on training and testing data

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

```
Training Accuracy : 0.9963805584281282
Testing Accuracy : 0.9650180940892642
```

```
#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()
```



```
#K fold cross-validation

accuracies = cross_val_score(estimator = model_rf, X = X_train_scl, y = y_train, cv = 10)

print("Accuracy :", accuracies.mean())
print("Standard Variance :", accuracies.std())
```

```
Accuracy : 0.956041343945302
Standard Variance : 0.015563172458390739
```

```
#Applying grid search to get the optimal parameters on random forest

params = {
    'bootstrap': [True],
    'max_depth': [80, 100],
    'min_samples_split': [8, 12],
    'n_estimators': [100, 300]
}


cv_object = StratifiedKFold(n_splits = 2)

grid_search = GridSearchCV(estimator = model_rf, param_grid = params, cv = cv_object, verbose = 0, return_train_score = True)
grid_search.fit(X_train_scl, y_train.ravel())
```

```
▸ GridSearchCV
▸ best_estimator_: RandomForestClassifier
    ▸ RandomForestClassifier
```

```
#Getting the best parameters from the grid search


print("Best Parameter Combination : {}".format(grid_search.best_params_))



print("Cross validation mean accuracy on train set : {}".format(grid_search.cv_results_['mean_train_score'].mean()*100))
print("Cross validation mean accuracy on test set : {}".format(grid_search.cv_results_['mean_test_score'].mean()*100))
print("Accuracy score for test set :", accuracy_score(y_test, y_preds))
```

```
Best Parameter Combination : {'bootstrap': True, 'max_depth': 100, 'min_samples_split': 8, 'n_estimators': 300}
Cross validation mean accuracy on train set : 97.66028955532575
Cross validation mean accuracy on test set : 89.80093071354705
Accuracy score for test set : 0.9650180940892642
```

```
#XgBoost
model_xgb = XGBClassifier()
model_xgb.fit(X_train_scl, y_train)
```

```
XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

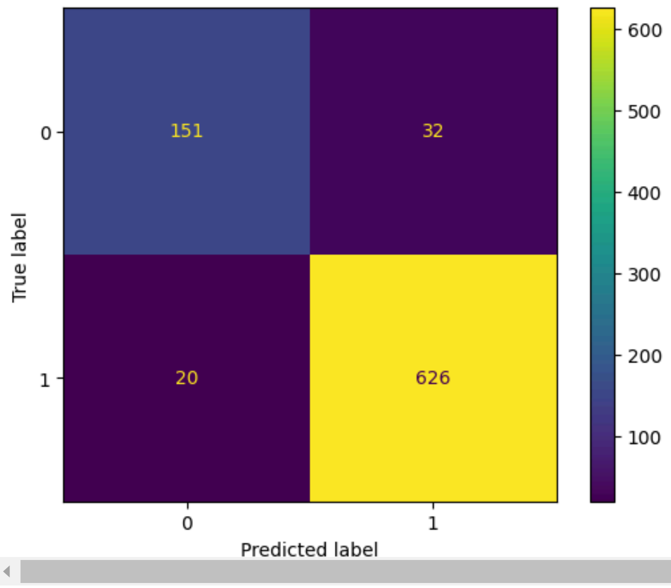
```
#Accuracy of the model on training and testing data

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

```
Training Accuracy : 0.9751809720785936
Testing Accuracy : 0.9396863691194209
```

```
y_preds = model_xgb.predict(X_test)

```

```
#Confusion Matrix
cm = confusion_matrix(y_test, y_preds)
print(cm)



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

```
[[151  32]
 [ 20 626]]
```



Decision Tree Classifier