```
#Importing necessary libraries
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import pointbiserialr
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Access the CSV file with the  path STEP1 LOAD DATA
path="/content/drive/MyDrive/OASIS/Twitter_Data.csv"
df = pd.read_csv(path, na_values=["NA", "NaN", "", "?","Not Available"])
```

```
df.shape
```

```
(162980, 2)
```

+ Code + Text

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 162980 entries, 0 to 162979
Data columns (total 2 columns):
 #   Column      Non-Null Count   Dtype  
---  ------      --------------   -----  
 0   clean_text  162976 non-null  object 
 1   category    162973 non-null  float64
dtypes: float64(1), object(1)
memory usage: 2.5+ MB
```

```
df.columns
```

```
Index(['clean_text', 'category'], dtype='object')
```

```
# Handle missing values (e.g., impute or drop rows/columns based on analysis)
(df.isnull().sum())
```

```
clean_text    4
category      7
dtype: int64
```

```
df['category'].value_counts()
```

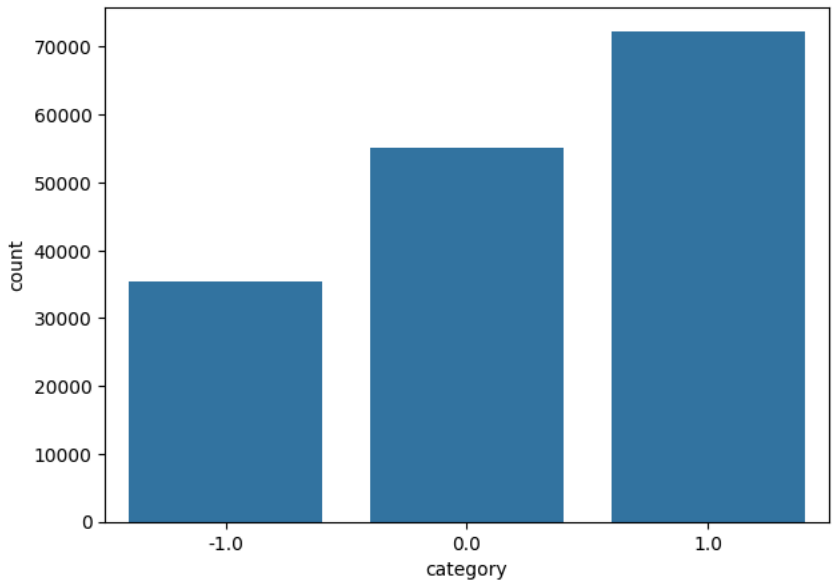
```
 1.0    72250
 0.0    55213
-1.0    35510
Name: category, dtype: int64
```

```
labels = pd.get_dummies(df.category)
labels.columns = ["negative", "neutral", "positive"]
labels.head()
```

	negative	neutral	positive
0	1	0	0
1	0	1	0
2	0	0	1
3	0	0	1
4	0	0	1

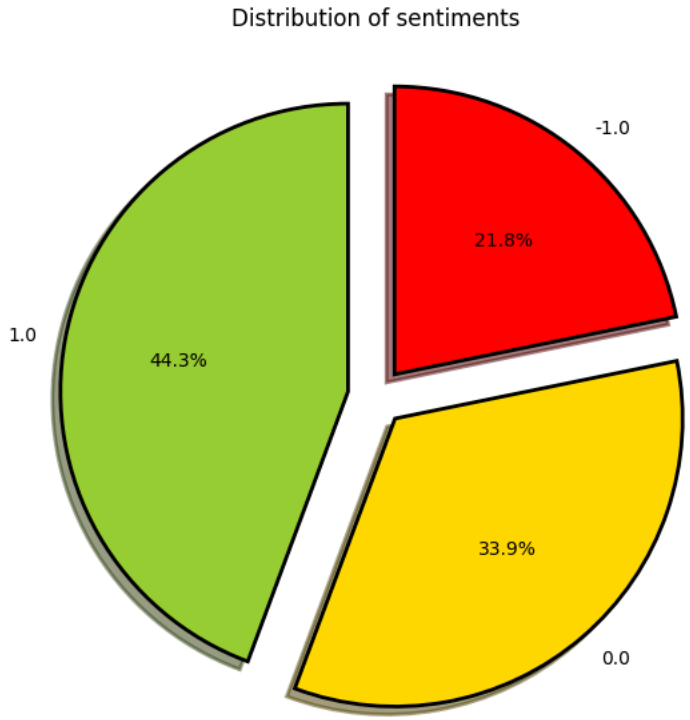
```
import matplotlib.pyplot as plt
import seaborn as sns
fig = plt.figure(figsize=(7,5))
sns.countplot(x="category",data=df)
```

```
<Axes: xlabel='category', ylabel='count'>
```



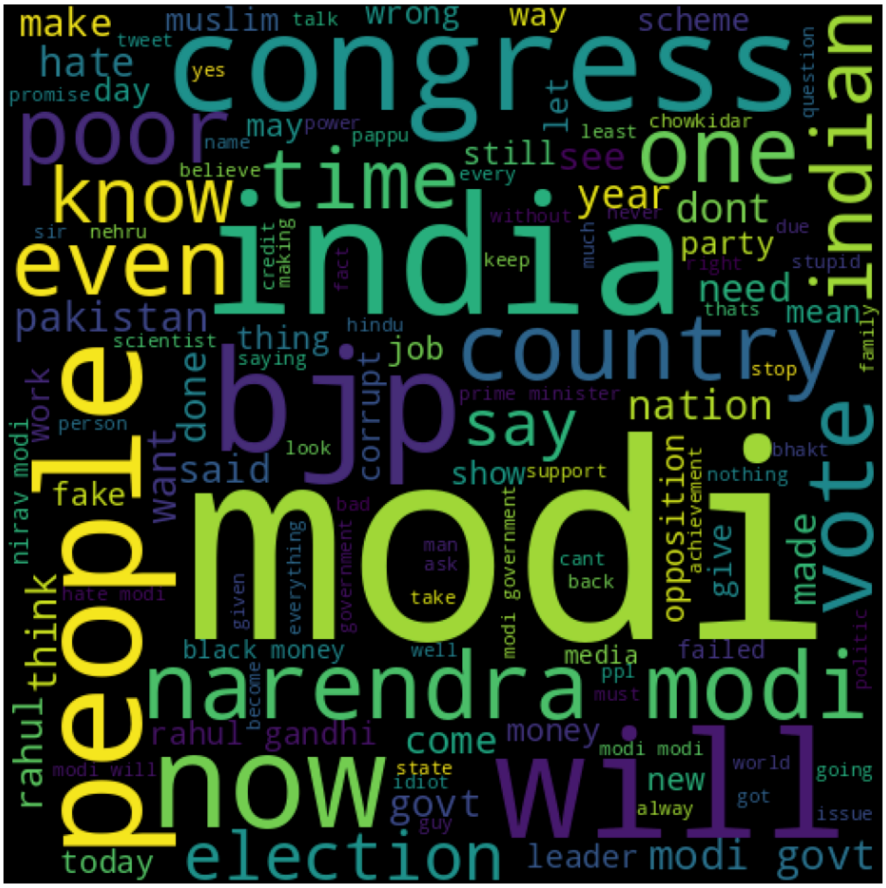

```
fig = plt.figure(figsize=(7,7))
colors = ("yellowgreen", "gold", "red")
wp = {'linewidth':2, 'edgecolor':"black"}
tags = df['category'].value_counts()
explode = (0.1,0.1,0.1)
tags.plot(kind='pie', autopct='%1.1f%%', shadow=True, colors = colors,
        startangle=90, wedgeprops = wp, explode = explode, label='')
plt.title('Distribution of sentiments')
```

```
Text(0.5, 1.0, 'Distribution of sentiments')
```



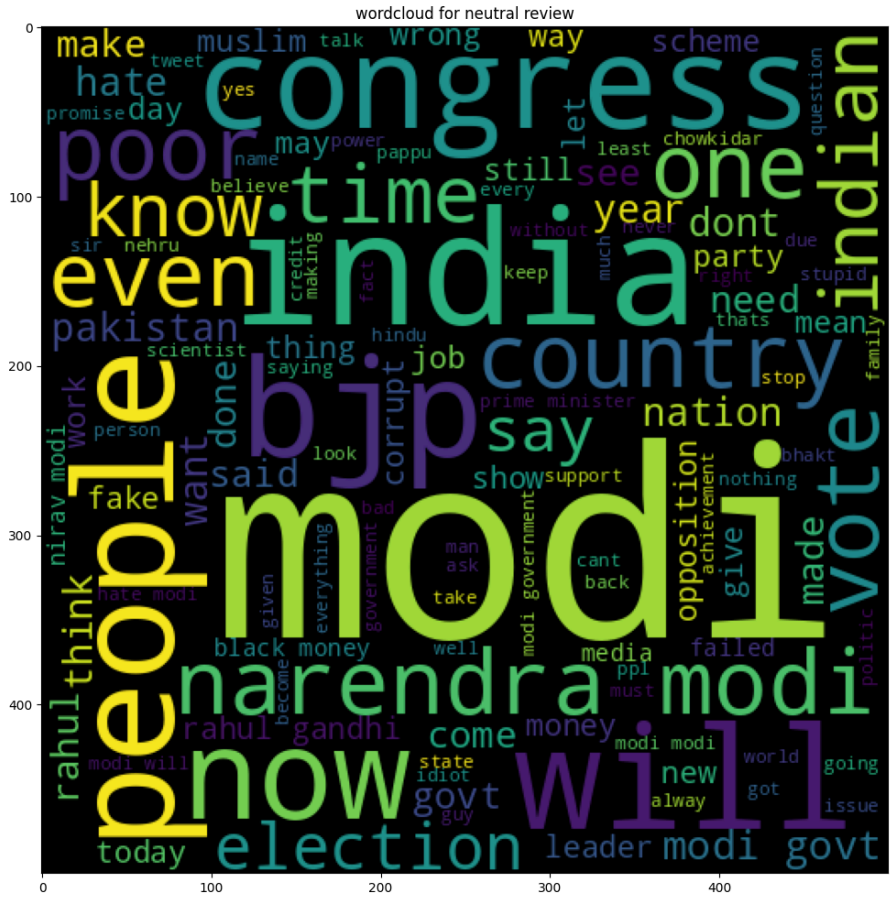
```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Initialize WordCloud
wc = WordCloud(width=500, height=500, min_font_size=10, background_color='black')

# Generate word clouds for positive, neutral, and negative categories
positive_wc = wc.generate(df[df['category'] == 1.0]['clean_text'].str.cat(sep=" "))
neutral_wc = wc.generate(df[df['category'] == 0.0]['clean_text'].str.cat(sep=" "))
negative_wc = wc.generate(df[df['category'] == -1.0]['clean_text'].str.cat(sep=" "))

# Plot word clouds
plt.figure(figsize=(12, 12))
plt.title('Wordcloud for Positive Review')
plt.imshow(positive_wc)
plt.axis('off')
plt.show()
```

Wordcloud for Positive Review




```
plt.figure(figsize = (12, 12))
plt.title('wordcloud for neutral review')
plt.imshow(neutral_wc)
```

```
<matplotlib.image.AxesImage at 0x7a1ea4e7ec20>
```




```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report,  ConfusionMatrixDisplay


from sklearn.model_selection import train_test_split

# Extracting features and target
X = df['clean_text'].values.astype('U')
y = df['category'].values.astype('U')

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y)
```

```
# Define the pipeline
pipe = Pipeline([
    ('tfidf_vectorizer', TfidfVectorizer(lowercase=True, stop_words='english', analyzer='word')),
    ('naive_bayes', MultinomialNB())
])

# Encoding the target labels
le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)

# Fitting the pipeline
pipe.fit(list(X_train), list(y_train))

# Making predictions
y_pred = pipe.predict(X_test)

# Evaluating the model
print(confusion_matrix(y_pred, y_test))
print(accuracy_score(y_pred, y_test))

# Accessing the Naive Bayes classifier from the pipeline
pipe['naive_bayes']
```

```
[[ 1279    41    51     0]
 [  428  5605   576     0]
 [ 8946 10918 21048     2]
 [    0     0     0     0]]
0.5712766392604409
```

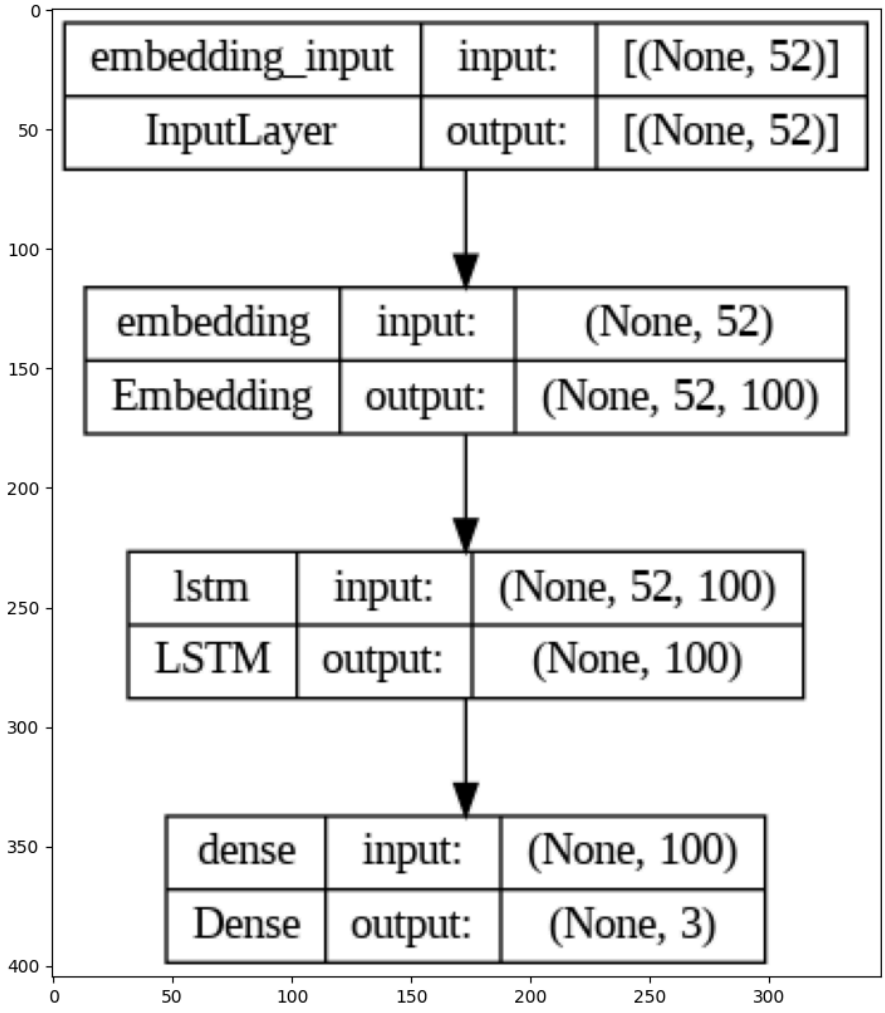
▾ MultinomialNB

MultinomialNB()

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split


# Tokenize the text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X)

# Convert text to sequences
X_sequences = tokenizer.texts_to_sequences(X)

# Pad sequences to have consistent length
X_padded = pad_sequences(X_sequences)

# Split the data into training and testing sets
X_train_seq, X_test_seq, y_train, y_test = train_test_split(X_padded, labels, test_size=0.3, stratify=labels)


model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=100, input_length=X_padded.shape[1]),
    tf.keras.layers.LSTM(100),
    tf.keras.layers.Dense(3, activation='softmax')  # Assuming 3 classes: negative, neutral, positive
])

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])


loss, accuracy = model.evaluate(X_test_seq, y_test)
print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')
```

```
1528/1528 [==============================] - 25s 15ms/step - loss: 0.3314 - accuracy: 0.8888
Test Loss: 0.3313680589199066, Test Accuracy: 0.8888002634048462
```

```
from tensorflow.keras.utils import plot_model
# Assuming 'model' is your Keras model
plot_model(model, to_file='model_architecture.png', show_shapes=True, show_layer_names=True)

# Display the plot
img = plt.imread('model_architecture.png')
plt.figure(figsize=(10, 10))
plt.imshow(img)
plt.show()
```


embedding_input
input:
[(None, 52)]
InputLayer
output:
[(None, 52)]
embedding
input:
(None, 52)
Embedding
output:
(None, 52, 100)
lstm
input:
(None, 52, 100)
LSTM
output:
(None, 100)
dense
input:
(None, 100)
Dense
output:
(None, 3)
0
50
100
150
200
250
300
350
400