In [ ]:
```python
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import spacy
import pandas as pd
import numpy as np
import re
from bs4 import BeautifulSoup
from string import punctuation
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import f1_score, accuracy_score

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier, GradientBoostingClas
sifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation, GRU, Flatten
from keras.layers import Bidirectional, GlobalMaxPool1D, Convolution1D
from keras.models import Model, Sequential
from keras import initializers, regularizers, constraints, optimizers, layers
```

In [ ]:
```python
df = pd.read_csv('/content/Flipkart_ratings.csv')
```

In [ ]:
```python
df.head(10)
```

In [ ]:
```python
df.columns
```

In [ ]:
```python
df.info()
```

In [ ]:
```python
df.isnull().sum()
```

In [ ]:
```python
l = ['marketplace', 'product_category', 'verified_purchase', 'vine', 'helpful_votes', 'total
_votes']

for i in l:
  print('The unique element in : ', i)
  print('-'*30)
  print(df[i].unique())
  print()
```

In [ ]:
```python
l=['product_category','verified_purchase','vine','helpful_votes', 'total_votes']

for i in l:
  print('The unique element in : ', i)
  print('-'*30)
  print(df[i].value_counts())
  print()
```

In [ ]:
```python
features = ['product_category', 'verified_purchase', 'vine', 'helpful_votes', 'total_votes']
for i in features:
  print('The count plot of : ', i)
  print('-'*30)
  plt.figure(figsize=(10,5))
  sns.countplot(df[i])
  plt.show()
  print()
```

In [ ]:
```python
df.star_rating.value_counts()
```

In [ ]:
```python
plt.figure()
plt.figure(figsize=(4,4),dpi=100)
sns.countplot(x='star_rating', data=df)
```

In [ ]:
```python
print('-' * 20 + "Percentage ratings" + '-' * 20)
star_ratings = df['star_rating'].value_counts() / len(df) * 100
star_ratings
```

In [ ]:
```python
plt.figure()
sns.barplot(star_ratings.index, star_ratings.values, order=star_ratings.index)
plt.ylabel("Percentage")
plt.xlabel('Rating')
```

In [ ]:
```python
df = df[df['star_rating'] != 3]
```

In [ ]:
```python
df['rating'] = df['star_rating'].apply(lambda x: 1 if x >= 4 else 0)
df['rating'].value_counts()
```

In [ ]:
```python
plt.figure()
sns.countplot(x = 'rating', data = df, palette ='rainbow')
plt.ylabel("Ratings count")
plt.xlabel("Rating")
```

In [ ]:
```python
df['reviews'] = df['review_body'] + " " + df['review_headline']
df.head(10)
```

In [ ]:
```python
df_new = df[['rating', 'reviews']]
df_new.head(10)
```

In [ ]:
```python
lemmitizer = WordNetLemmatizer()
nltk.download('stopwords')
nltk.download('wordnet')

df_new['reviews'] = df_new['reviews'].apply(lambda sentence: sentence.lower())
df_new.head(10)
```

In [ ]:
```python
emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
```

In [ ]:
```python
def decontracting_words(sentence):
    # specific
    sentence = re.sub(r"won\'t", "will not", sentence)
    sentence = re.sub(r"can\'t", "can not", sentence)

    # general
    sentence = re.sub(r"n\'t", " not", sentence)
    sentence = re.sub(r"\'re", " are", sentence)
    sentence = re.sub(r"\'s", " is", sentence)
    sentence = re.sub(r"\'d", " would", sentence)
    sentence = re.sub(r"\'ll", " will", sentence)
    sentence = re.sub(r"\'t", " not", sentence)
    sentence = re.sub(r"\'ve", " have", sentence)
    sentence = re.sub(r"\'m", " am", sentence)
    return sentence
```

In [ ]:
```python
stopwrd = stopwords.words('english')
def remove_html_urls_emoticons(sentence):
  # Remove HTML, XML, tags from data
  sentence = BeautifulSoup(sentence, 'lxml').get_text()
  # Remove URLs from the data
  sentence = re.sub("http\S+", "", sentence)
  # Decontract words
  sentence = decontracting_words(sentence)
  # Removingemoticons
  sentence = emoji_pattern.sub(' ', sentence)
  return sentence
```

In [ ]:
```python
def remove_numeric_punctuations_stopwords(sentence):
  # Remove words with numbers in them from the data
  sentence = re.sub(r"\S*\d\S*", "", sentence)
  # Remove punctuations, numbers
  sentence = re.sub(r"[^A-Za-z]", ' ', sentence)
  # Remove stopwords
  sentence = " ".join([word for word in sentence.split() if word not in stopwords])
  return sentence
```

In [ ]:
```python
df_new['reviews'] = df_new['reviews'].apply(remove_html_urls_emoticons)
df_new['reviews'] = df_new['reviews'].apply(remove_numeric_punctuations_stopwords)
```

In [ ]:
```python
df_new['reviews'] = df_new['reviews'].apply(lambda x: " ".join([lemmitizer.lemmatize(token)
for token in x.split()]))
```

In [ ]:
```python
from wordcloud import WordCloud

for i in range (0,2):
  print('When star Rating is : ', i)
  print('-' * 50)
  print()
  text = df_new[df_new['rating'] == i]
  all_words = ' '.join([text for text in text.reviews])
  wordcloud = WordCloud(width= 1500, height= 800,
                        max_font_size = 170,
                        collocations = False).generate(all_words)
  plt.figure(figsize=(10,7))
  plt.imshow(wordcloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()
```

In [ ]:
```python
from nltk import tokenize

token_space = tokenize.WhitespaceTokenizer()
def counter(text, column_text, quantity):
    all_words = ' '.join([text for text in text[column_text]])
    token_phrase = token_space.tokenize(all_words)
    frequency = nltk.FreqDist(token_phrase)
    data_frequency = pd.DataFrame({"Word": list(frequency.keys()),
                                   "Frequency": list(frequency.values())})
    data_frequency = data_frequency.nlargest(columns = "Frequency", n = quantity)
    plt.figure(figsize=(15,8))
    ax = sns.barplot(data = data_frequency, x = "Word", y = "Frequency", color = 'maroon')
    ax.set(ylabel = "Count")
    plt.xticks(rotation='vertical')
    plt.show()


for i in range(0,2):
  print('When star Rating is : ', i)
  print('_'*30)
  print()
  counter(df_new[df_new['rating'] == i], 'reviews', 20)
  print()
```

In [ ]:

Modelling Phase

In [ ]:
```python
X = df_new['reviews']
Y = df_new['rating']
```

In [ ]:
```python
#Splitting data into train test set
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42
)
```

In [ ]:
```python
print(x_train.shape)
print('-' * 50)
print(x_test.shape)
print('-' * 50)
print(y_train.shape)
print('-' * 50)
print(y_test.shape)
print('-' * 50)
```

In [ ]:
```python
# Tokenizing the words using Tokenizer
# Using the most popular 8000 words from the dataset
# Encoding the words in sentences with their key from Tokenizers and padding the encodings w
ith small lengths than 140
# Max character length in sentence is set to 140
top_words = 8000
tokenizer = Tokenizer(num_words = top_words, oov_token="#OOV")
tokenizer.fit_on_texts(x_train)
list_tokenized_train = tokenizer.texts_to_sequences(x_train)

max_review_length = 140
X_train = pad_sequences(list_tokenized_train, maxlen = max_review_length)
Y_train = y_train
```

In [ ]:
```python
# Encoding the text to sequences using tokenizer to prepare data for neural network
test_word_list = tokenizer.texts_to_sequences(x_test)
X_test = pad_sequences(test_word_list, maxlen = max_review_length)
Y_test = y_test
```

Modelling using CountVectorizer

In [ ]:
```python
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(max_features=8000, ngram_range=(2,2))
x_stem = cv.fit_transform(X).toarray()
x_stem
```

In [ ]:
```python
x_train, x_test, y_train, y_test = train_test_split(x_stem, Y, test_size = 0.2, random_state
= 42)
```

Decision Tree Classifier

In [ ]:
```python
dt = DecisionTreeClassifier(max_depth=20)
dt.fit(x_train, y_train)
```

In [ ]:
```python
print("Train set Accuracy: ", dt.score(x_train, y_train))
print("Test set Accuracy: ", dt.score(x_test, y_test))
print("Train set f1-score: ", f1_score(dt.predict(x_train), y_train))
print("Test set f1-score: ", f1_score(dt.predict(x_test), y_test))
```

Logistic Regression

In [ ]:
```python
lr = LogisticRegression(max_iter=5000)
lr.fit(x_train, y_train)
```

In [ ]:
```python
print("Train set Accuracy: ",lr.score(x_train, y_train))
print("Test set Accuracy: ", lr.score(x_test, y_test))
print("Train set f1-score: ",f1_score(dt.predict(x_train), y_train))
print("Test set f1-score: ", f1_score(dt.predict(x_test), y_test))
```

Random Forest Classifier

In [ ]:
```python
rfc = RandomForestClassifier(n_estimators = 60)
rfc.fit(x_train, y_train)
```

In [ ]:
```python
print("Train set Accuracy: ",rfc.score(x_train, y_train))
print("Test set Accuracy: ", rfc.score(x_test, y_test))
print("Train set f1-score: ",f1_score(rfc.predict(x_train), y_train))
print("Test set f1-score: ", f1_score(rfc.predict(x_test), y_test))
```

Gradient Boosting Classifier

In [ ]:
```python
gbc = GradientBoostingClassifier(n_estimators=130)
gbc.fit(x_train, y_train)
```

In [ ]:
```python
print("Train set Accuracy: ",gbc.score(x_train, y_train))
print("Test set Accuracy: ", gbc.score(x_test, y_test))
print("Train set f1-score: ",f1_score(gbc.predict(x_train), y_train))
print("Test set f1-score: ", f1_score(gbc.predict(x_test), y_test))
```

Bagging Classifier

In [ ]:
```python
bc = BaggingClassifier(n_estimators = 80)
bc.fit(x_train, y_train)
```

In [ ]:
```python
y_pred_train = bc.predict(x_train)
y_pred_test = bc.predict(x_test)
```

In [ ]:
```python
print("Train set Accuracy: ",accuracy_score(y_pred_train, y_train))
print("Test set Accuracy: ",accuracy_score(y_pred_test, y_test))
print("Train set f1-score: ",f1_score(y_pred_train, y_train))
print("Test set f1-score: ", f1_score(y_pred_test, y_test))
```

Gaussian Naive Bayes

In [ ]:
```python
gnb = GaussianNB()
gnb.fit(x_train, y_train)
```

In [ ]:
```python
y_pred_train = gnb.predict(x_train)
y_pred_test = gnb.predict(x_test)
```

In [ ]:
```python
print("Train set Accuracy: ",accuracy_score(y_pred=y_pred_train, y_true=y_train))
print("Test set Accuracy: ", accuracy_score(y_pred=y_pred_test, y_true=y_test))
print("Train set f1-score: ",f1_score(gnb.predict(x_train), y_train))
print("Test set f1-score: ", f1_score(gnb.predict(x_test), y_test))
```

K Neighbours Classifier

In [ ]:
```python
knc = KNeighborsClassifier(n_neighbors = 13)
knc.fit(x_train, y_train)
```

In [ ]:
```python
y_pred_train = knc.predict(x_train)
y_pred_test = knc.predict(x_test)
```

In [ ]:
```python
print("Train set Accuracy: ",accuracy_score(y_pred_train, y_train))
print("Test set Accuracy: ", accuracy_score(y_pred_test, y_test))
print("Train set f1-score: ",f1_score(y_pred_train, y_train))
print("Test set f1-score: ", f1_score(y_pred_test, y_test))
```

Deep Learning Model

In [ ]:
```python
model = Sequential()
model.add(Embedding(top_words+1, 32, input_length=max_review_length))
model.add(LSTM(200))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

In [ ]:
```python
model.fit(X_train, Y_train, epochs=20, batch_size=8, validation_split=0.2)
```

In [ ]:
```python
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)
```

In [ ]:
```python
y_pred_train = (y_pred_train > 0.5)
y_pred_test = (y_pred_test > 0.5)
```

In [ ]:
```python
print("Train set Accuracy: ",accuracy_score(y_pred=y_pred_train, y_true=y_train))
print("Test set Accuracy: ", accuracy_score(y_pred=y_pred_test, y_true=y_test))
print("Train set f1-score: ",f1_score(y_pred_train, y_train))
print("Test set f1-score: ", f1_score(y_pred_test, y_test))
```

In [ ]:
```python
from textblob import TextBlob
```

In [ ]:
```python
df_new.columns
```

In [ ]:
```python
df_new['reviews']= df_new['reviews'].astype(str) # Make sure about the correct data type
pol = lambda x: TextBlob(x).sentiment.polarity
df_new['polarity'] = df_new['reviews'].apply(pol)
num_bins = 50
plt.figure(figsize=(10,6))
n, bins, patches = plt.hist(df_new.polarity, num_bins, facecolor='blue', alpha=0.5)
plt.xlabel('Polarity')
plt.ylabel('Number of Reviews')
plt.title('Histogram of Polarity Score')
```

In [ ]:
```python
plt.figure(figsize=(10,6))
sns.boxenplot(x='rating', y='polarity', data=df_new)
```

In [ ]:
```python
sub = lambda x: TextBlob(x).sentiment.subjectivity
df_new['subjectivity'] = df_new['reviews'].apply(sub)
df_new.sample(10)
```

In [ ]:
```python
# Density Plot and Histogram of subjectivity
plt.figure(figsize=(10,5))
sns.distplot(
    df_new['subjectivity'],
    hist=True, kde=True,bins=int(30),
    color = 'darkblue',hist_kws={'edgecolor':'black'},
    kde_kws={'linewidth': 4}
)

plt.xlim([-0.001,1.001])
plt.xlabel('Subjectivity', fontsize=13)
plt.ylabel('Frequency', fontsize=13)
plt.title('Distribution of Subjectivity Score', fontsize=15)
```

In [ ]:
```python
plt.figure(figsize=(10,6))
sns.scatterplot(x='polarity', y='subjectivity', hue="rating", data=df_new)
plt.xlabel('Polarity', fontsize=13)
plt.ylabel('Subjectivity', fontsize=13)
plt.title('Polarity vs Subjectivity', fontsize=15)
```

In [ ]: