6th program and output:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.preprocessing import LabelBinarizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud
from bs4 import BeautifulSoup
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import re
import warnings

# Ignore warnings
warnings.filterwarnings('ignore')

# Download necessary NLTK data
nltk.download('stopwords')
nltk.download('punkt_tab')

# Load the dataset
imdb_data = pd.read_csv('//content/IMDB Dataset.csv')
print("Dataset Shape:", imdb_data.shape)
print(imdb_data.head(10))

# Sentiment count
print(imdb_data['sentiment'].value_counts())

# Split the dataset
# Use the actual length of the dataset for splitting
train_reviews = imdb_data.review[:int(len(imdb_data) * 0.8)]  # 80% for training
train_sentiments = imdb_data.sentiment[:int(len(imdb_data) * 0.8)]
test_reviews = imdb_data.review[int(len(imdb_data) * 0.8):]  # Remaining 20% for testing
test_sentiments = imdb_data.sentiment[int(len(imdb_data) * 0.8):]
print("Train Shape:", train_reviews.shape, train_sentiments.shape)
print("Test Shape:", test_reviews.shape, test_sentiments.shape)

# Preprocessing Functions
def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

def remove_between_square_brackets(text):
    return re.sub(r'\[[^]]*\]', '', text)

def denoise_text(text):
    text = strip_html(text)
    text = remove_between_square_brackets(text)
    return text

def remove_special_characters(text):
    return re.sub(r'[^a-zA-Z\s]', '', text)

def remove_stopwords(text):
```

```
    stopword_list = set(stopwords.words('english'))
    words = text.split()
    return ' '.join([word for word in words if word.lower() not in stopword_list])

# Apply preprocessing
imdb_data['review'] = imdb_data['review'].apply(denoise_text)
imdb_data['review'] = imdb_data['review'].apply(remove_special_characters)
imdb_data['review'] = imdb_data['review'].apply(remove_stopwords)

# Vectorization
cv = CountVectorizer(min_df=1.0, max_df=1.0, binary=False, ngram_range=(1, 3))  # Change min_df to 1 or a float between (
tv = TfidfVectorizer(min_df=0.0, max_df=1.0, use_idf=True, ngram_range=(1, 3))

cv_train_reviews = cv.fit_transform(train_reviews)
cv_test_reviews = cv.transform(test_reviews)
tv_train_reviews = tv.fit_transform(train_reviews)
tv_test_reviews = tv.transform(test_reviews)
print("BOW Shape (Train):", cv_train_reviews.shape)
print("BOW Shape (Test):", cv_test_reviews.shape)
print("TFIDF Shape (Train):", tv_train_reviews.shape)
print("TFIDF Shape (Test):", tv_test_reviews.shape)

# Label Encoding
lb = LabelBinarizer()
train_sentiments = lb.fit_transform(train_sentiments)
test_sentiments = lb.transform(test_sentiments)

# Logistic Regression
lr = LogisticRegression(max_iter=500, random_state=42)

# BOW
lr_bow = lr.fit(cv_train_reviews, train_sentiments)
lr_bow_predict = lr.predict(cv_test_reviews)
lr_bow_score = accuracy_score(test_sentiments, lr_bow_predict)
print("Logistic Regression BOW Accuracy:", lr_bow_score)

# TFIDF
lr_tfidf = lr.fit(tv_train_reviews, train_sentiments)
lr_tfidf_predict = lr.predict(tv_test_reviews)
lr_tfidf_score = accuracy_score(test_sentiments, lr_tfidf_predict)
print("Logistic Regression TFIDF Accuracy:", lr_tfidf_score)

# Classification Reports
print("Classification Report for Logistic Regression (BOW):")
print(classification_report(test_sentiments, lr_bow_predict, target_names=['Negative', 'Positive']))

print("Classification Report for Logistic Regression (TFIDF):")
print(classification_report(test_sentiments, lr_tfidf_predict, target_names=['Negative', 'Positive']))

# Confusion Matrix
print("Confusion Matrix (BOW):")
print(confusion_matrix(test_sentiments, lr_bow_predict))

print("Confusion Matrix (TFIDF):")
print(confusion_matrix(test_sentiments, lr_tfidf_predict))

# Word Cloud for Positive Reviews
plt.figure(figsize=(10, 10))
positive_text = ' '.join(imdb_data.loc[imdb_data['sentiment'] == 'positive', 'review'])
wordcloud = WordCloud(width=1000, height=500, max_words=500, min_font_size=5).generate(positive_text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Word Cloud for Positive Reviews")
plt.show()
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
Dataset Shape: (6, 2)
                                              review sentiment
0  I loved this movie. It was fantastic, full of ...  positive
1  The film was a waste of time. Poor acting and ...  negative
2  An amazing experience, great cinematography, a...  positive
3  Terrible! One of the worst movies I've ever se...  negative
4  Simply brilliant. It kept me on the edge of my...  positive
5  Not my kind of movie. The pacing was too slow,...  negative
sentiment
positive    3
negative    3
Name: count, dtype: int64
Train Shape: (4,) (4,)
Test Shape: (2,) (2,)
BOW Shape (Train): (4, 1)
BOW Shape (Test): (2, 1)
TFIDF Shape (Train): (4, 126)
TFIDF Shape (Test): (2, 126)
Logistic Regression BOW Accuracy: 0.5
Logistic Regression TFIDF Accuracy: 0.5
Classification Report for Logistic Regression (BOW):
              precision    recall  f1-score   support

    Negative       0.50      1.00      0.67         1
    Positive       0.00      0.00      0.00         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2

Classification Report for Logistic Regression (TFIDF):
              precision    recall  f1-score   support

    Negative       0.50      1.00      0.67         1
    Positive       0.00      0.00      0.00         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2

Confusion Matrix (BOW):
[[1 0]
 [1 0]]
Confusion Matrix (TFIDF):
[[1 0]
 [1 0]]
```

Word Cloud for Positive Reviews

7th program and output :

```
import keras
from keras import layers
from keras import regularizers
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt

# This is the size of our encoded representations
encoding_dim = 32  # 32 floats -> compression factor of 24.5, assuming the input is 784 floats

# This is our input image (28x28 pixels flattened into 784)
input_img = keras.Input(shape=(784,))

# "encoded" is the encoded representation of the input
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)

# "decoded" is the lossy reconstruction of the input
decoded = layers.Dense(784, activation='sigmoid')(encoded)

# This model maps an input to its reconstruction
autoencoder = keras.Model(input_img, decoded)

# This model maps an input to its encoded representation
encoder = keras.Model(input_img, encoded)

# This is our encoded (32-dimensional) input
encoded_input = keras.Input(shape=(encoding_dim,))

# Retrieve the last layer of the autoencoder model (decoder layer)
decoder_layer = autoencoder.layers[-1]

# Create the decoder model
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Load the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize the data
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Flatten the images
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

# Train the autoencoder model
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))

# Encode and decode some digits
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

# Display the original and reconstructed images
n = 10  # Number of digits to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
```

```python
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()
```

```
Epoch 1/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - loss: 0.3824 - val_loss: 0.1883
Epoch 2/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - loss: 0.1775 - val_loss: 0.1526
Epoch 3/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1487 - val_loss: 0.1337
Epoch 4/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.1316 - val_loss: 0.1209
Epoch 5/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.1200 - val_loss: 0.1129
Epoch 6/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.1123 - val_loss: 0.1072
Epoch 7/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1069 - val_loss: 0.1030
Epoch 8/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1032 - val_loss: 0.0997
Epoch 9/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.1004 - val_loss: 0.0975
Epoch 10/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.0980 - val_loss: 0.0959
Epoch 11/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0967 - val_loss: 0.0947
Epoch 12/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - loss: 0.0958 - val_loss: 0.0941
Epoch 13/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.0950 - val_loss: 0.0935
Epoch 14/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0948 - val_loss: 0.0931
Epoch 15/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 20ms/step - loss: 0.0943 - val_loss: 0.0930
Epoch 16/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0940 - val_loss: 0.0927
Epoch 17/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.0939 - val_loss: 0.0925
Epoch 18/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0938 - val_loss: 0.0923
Epoch 19/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.0938 - val_loss: 0.0923
Epoch 20/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0934 - val_loss: 0.0922
Epoch 21/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - loss: 0.0932 - val_loss: 0.0921
Epoch 22/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.0934 - val_loss: 0.0920
Epoch 23/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.0932 - val_loss: 0.0921
Epoch 24/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.0931 - val_loss: 0.0919
Epoch 25/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.0931 - val_loss: 0.0919
Epoch 26/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0931 - val_loss: 0.0919
Epoch 27/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.0931 - val_loss: 0.0918
Epoch 28/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.0931 - val_loss: 0.0919
Epoch 29/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0930 - val_loss: 0.0918
Epoch 30/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0929 - val_loss: 0.0918
Epoch 31/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0929 - val_loss: 0.0918
Epoch 32/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0928 - val_loss: 0.0918
Epoch 33/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0927 - val_loss: 0.0917
Epoch 34/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.0928 - val_loss: 0.0917
Epoch 35/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.0928 - val_loss: 0.0917
Epoch 36/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - loss: 0.0927 - val_loss: 0.0917
Epoch 37/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0929 - val_loss: 0.0917
Epoch 38/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - loss: 0.0929 - val_loss: 0.0916
Epoch 39/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - loss: 0.0928 - val_loss: 0.0916
Epoch 40/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.0927 - val_loss: 0.0917
Epoch 41/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0928 - val_loss: 0.0917
Epoch 42/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0928 - val_loss: 0.0916
Epoch 43/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0926 - val_loss: 0.0915
Epoch 44/50
```

```
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - loss: 0.0925 - val_loss: 0.0916
Epoch 45/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0926 - val_loss: 0.0915
Epoch 46/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.0923 - val_loss: 0.0915
Epoch 47/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - loss: 0.0924 - val_loss: 0.0915
Epoch 48/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 49/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - loss: 0.0925 - val_loss: 0.0915
Epoch 50/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.0924 - val_loss: 0.0915
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```