جهت سهولت در نوشتن کد ها، هر فاز به صورت جداگانه نوشته شده و در نهایت خروجی فاز برای استفاده در مراحل بعدی ذخیره شده.

کتابخانه های استفاده شده:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from textblob import TextBlob
import matplotlib.pyplot as plt
from nltk.stem import PorterStemmer
import re

from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import numpy as np

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

import urllib.request
import zipfile
import os

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GRU, Dense, Bidirectional
from tensorflow.keras.callbacks import History
```

۱-پیش پردازش

در مرحله اول یک تابع برای بازکردن فایل اکسل نوشته شده:

```
# Step 1: Load the dataset
def load_dataset(file_path):
    df = pd.read_csv(file_path)
    return df
```

مرحله ۱-۱-

در این مرحله مثبت ها را به یک تبدیل کردیم و منفی ها را به صفر

```
# Step 1.1: Convert positive labels to 1 and negative labels to 0
def convert_labels(df):
    df['sentiment'] = df['sentiment'].apply(lambda x: 1 if x == 'positive' else 0)
    return df
```

مرحله ۱-۲-

در این مرحله رکود های تکراری حذف شده. از inplace استفاده شده تا فایل جدیدی ایجاد نشود و در همان فایل تغییرات اعمال شود.

```
# Step 1.2: Remove duplicate rows
def remove_duplicates(df):
    df.drop_duplicates(inplace=True)
```

مرحله ۱-۳-

در این مرحله ۳۰ درصد داده ها برای آموزش با رندم استیت ۴۲ جدا شده اند:

```
# Step 1.3: Randomly select 30% of the dataset
def random_selection(df):
    df = df.sample(frac=0.3, random_state=42)
    return df
```

مرحله ۱-۴-

در این مرحله یک ستون جدید ایجاد شده و با استفاده از کتابخانه textBlob قطبیت جمله در آن ثبت شده:

```
# Step 1.4: Calculate polarity using TextBlob
def get_polarity(df):
    print('step-1-4')
    df['polarity'] = df['review'].apply(lambda x: TextBlob(str(x)).sentiment.polarity)
    return df
```

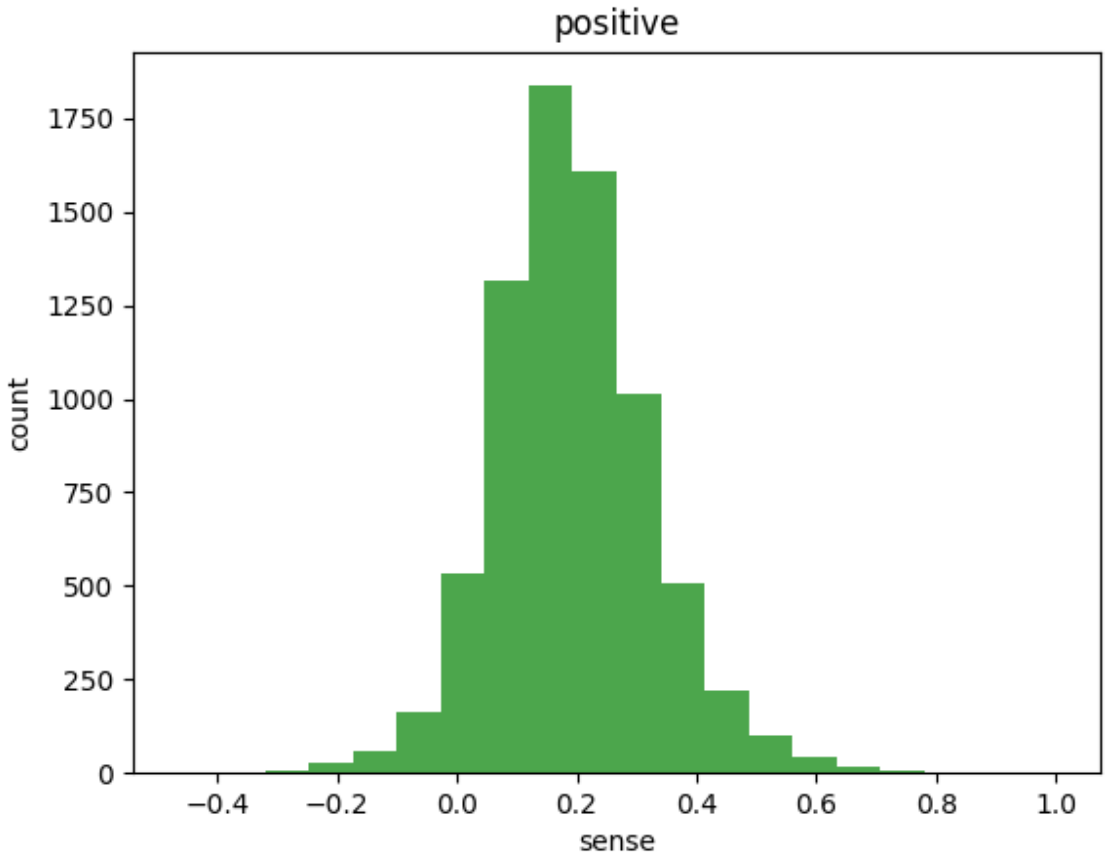
مرحله ۱-۵-

در این مرحله از ستون sentiment ستون های ۱ را مثبت و ۰ را منفی در نظر گرفتیم و با پلات نمودار رسم کرده و ذخیره کردیم:

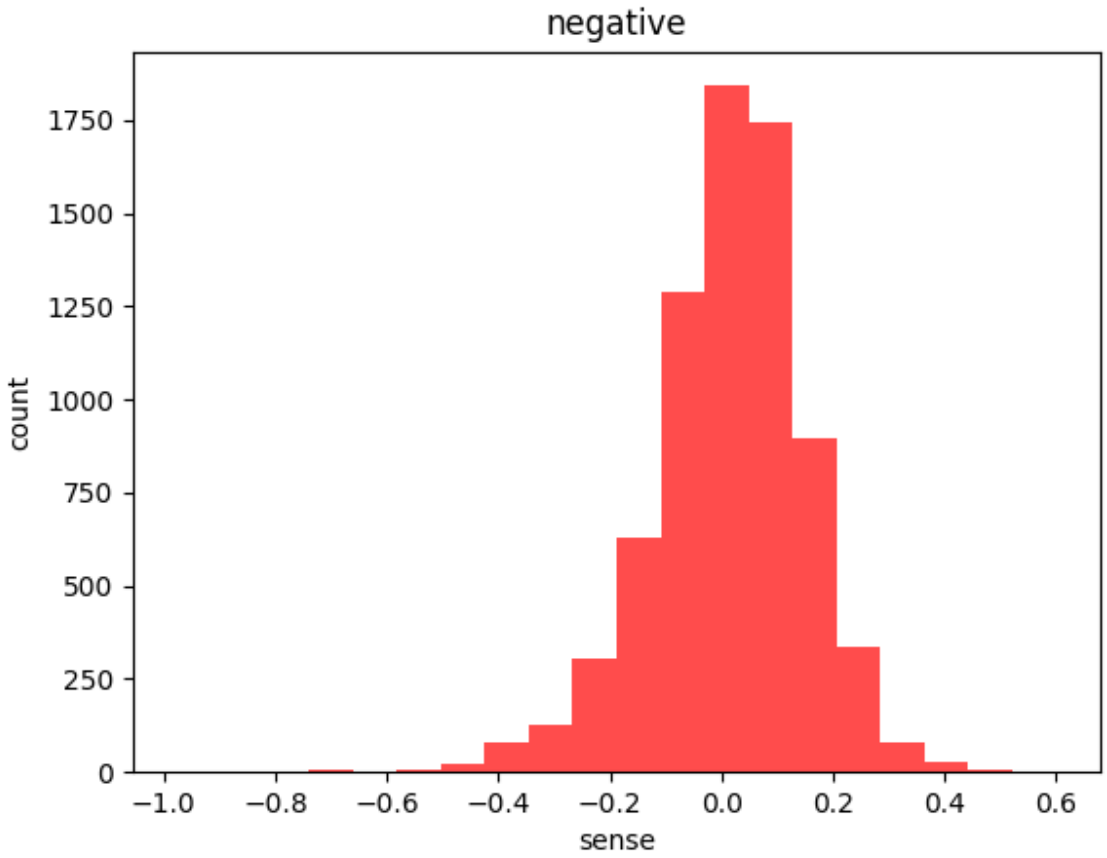
```
# Step 1.5: Separate positive and negative polarities, plot histograms, and analyze means
def plot_sentiment_histograms(df):
    print('step-1-5')
    positive_reviews = df[df['sentiment'] == 1]
    negative_reviews = df[df['sentiment'] == 0]

    # Plotting histogram for positive sentiment
    plt.hist(positive_reviews['polarity'], bins=20, color='green', alpha=0.7)
    plt.title('positive')
    plt.xlabel('sense')
    plt.ylabel('count')
    plt.savefig('positive.png')
    plt.show()

    # Plotting histogram for negative sentiment
    plt.hist(negative_reviews['polarity'], bins=20, color='red', alpha=0.7)
    plt.title('negative')
    plt.xlabel('sense')
```

```
    plt.ylabel('count')
    plt.savefig('negative.png')
    plt.show()

    # Calculate mean polarity for positive and negative reviews
    mean_polarity_positive = positive_reviews['polarity'].mean()
    mean_polarity_negative = negative_reviews['polarity'].mean()

    # Print mean polarities
    print("Mean Polarity for Positive Reviews:", mean_polarity_positive)
    print("Mean Polarity for Negative Reviews:", mean_polarity_negative)

```

خروجی مثبت:



خروجی منفی:



خروجی میانگین ها در کنسول:

```
1. Mean Polarity for Positive Reviews: 0.19375026016013788
2. Mean Polarity for Negative Reviews: 0.012643524175639971
```

مرحله ۱-۶-

در این مرحله ابتدا تمام حروف کوچک شده و با رجکس اصلاح شده اند سپس استاپ ورد ها حذف شده و استم کلمات استخراج شده:

```
 1. # Step 1.6: Apply preprocessing steps
 2. def preprocess_text(text):
 3.     text = text.lower()
 4.     text = re.sub(r'\W', ' ', text)
 5.     text = re.sub(r'\s+', ' ', text)
 6.
 7.     stop_words = set(stopwords.words('english'))
 8.     text = ' '.join([word for word in text.split() if word not in stop_words])
 9.
10.     stemmer = PorterStemmer()
```

```
    text = ' '.join([stemmer.stem(word) for word in text.split()])
```

مرحله ۱-۷-

برای تقسیم داده به شکر زیر اقدام شده:

```
# Step 1.7: Split the dataset into training and testing sets
def split_dataset(df):
    train_data, test_data, train_labels, test_labels = train_test_split(
        df['review'], df['sentiment'], test_size=0.5, random_state=42
    )
    return train_data, test_data, train_labels, test_labels
```

۲-خوشه بندی

مرحله ۲-۱-

در این CountVectorizer فراخوانی شده و داده های آموزشی و تست فیت ترنسفورم شده اند.

```
# Step 2-1: Create Bag of Words (BOW) representation for each review
def create_bow_representation(reviews_train, reviews_test):
    vectorizer = CountVectorizer()

    bow_matrix_train = vectorizer.fit_transform(reviews_train)
    bow_matrix_test = vectorizer.transform(reviews_test)

    return bow_matrix_train, bow_matrix_test
```

مرحله ۲-۲-

در این مرحله خوشه بند k-means را فراخوانی کرده و دسته بندی را انجام دادیم:

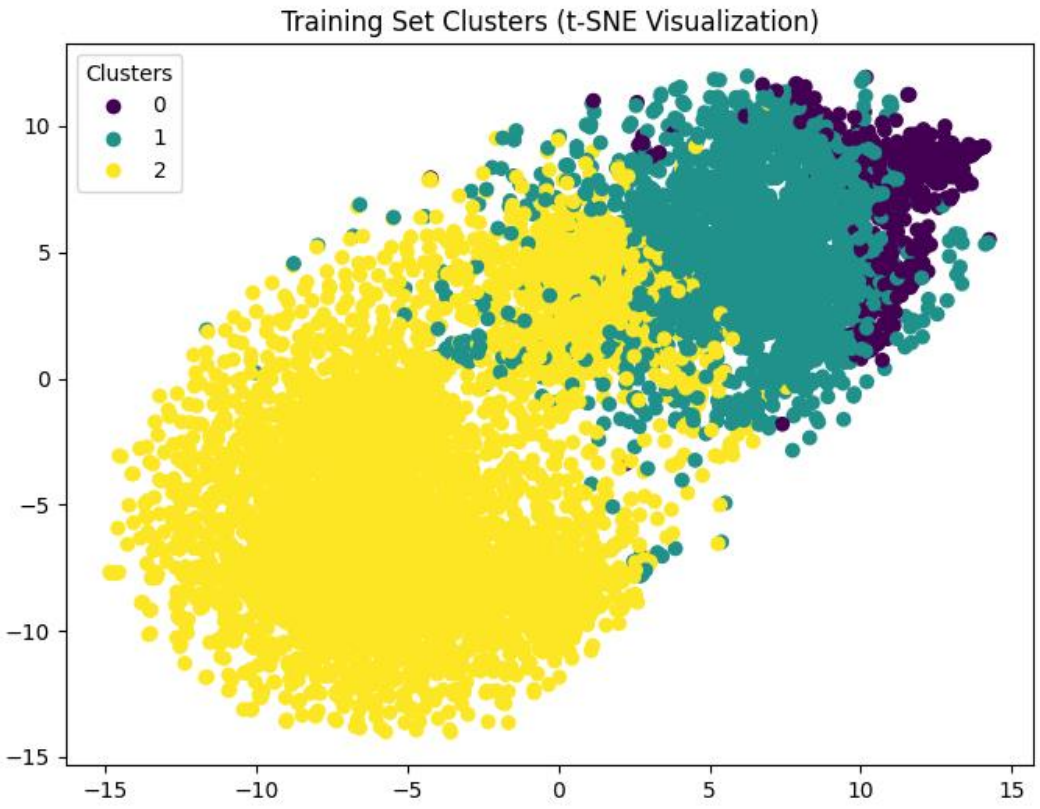
```
# Step 2-2: Cluster the training and testing sets using K-means
def kmeans_clustering(bow_matrix_train, bow_matrix_test, num_clusters=3):
    kmeans = KMeans(n_clusters=num_clusters, random_state=42)

    # Fit on training set
    train_clusters = kmeans.fit_predict(bow_matrix_train)

    # Predict on testing set
    test_clusters = kmeans.predict(bow_matrix_test)

    return train_clusters, test_clusters

```

مرحله ۲-۳-

در این مرحله با استفاده tsne خوشه ها رسم شده اند و در پلات نمایش داده شده اند:

```
# Step 2-3: Visualize clusters using t-sne
def visualize_clusters(bow_matrix, clusters, title):
    print('step-2-3')
    tsne = TSNE(n_components=2, random_state=42)
    tsne_result = tsne.fit_transform(bow_matrix.toarray())

    plt.figure(figsize=(8, 6))
    scatter = plt.scatter(tsne_result[:, 0], tsne_result[:, 1], c=clusters, cmap='viridis')
    plt.title(title)
    plt.legend(*scatter.legend_elements(), title="Clusters")
    plt.savefig('training_clusters_plot.png')
    plt.show()
```

خروجی:



مرحله ۲-۴-

در این مرحله ابتدا از هر دسته داده های یکتا را جدا کردیم و مینیم تعداد را مشخص کردیم و به تعداد مینیمم از داده ها به صورت رندوم انتخاب کردیم، همچنین اگر داده ای تکراری وجود داشته باشد آن را جایگزین نمی کنیم.

```
# Step 2-4: Select a uniform sample from each cluster
def select_uniform_sample(bow_matrix, clusters, num_samples):
    unique_clusters, cluster_counts = np.unique(clusters, return_counts=True)

    selected_samples = []

    for cluster in unique_clusters:
        cluster_indices = np.where(clusters == cluster)[0]
        mi = min(num_samples, cluster_counts[cluster])
        selected_indices = np.random.choice(cluster_indices, size=mi, replace=False)
        selected_samples.extend(selected_indices)

    return selected_samples
```

مرحله ۳ بردار سازی متن

مرحله ۳-۱-

در این مرحله توکنایز را فراخوانی کرده ایم و بر روی داده های تست و آموزشی آن را اعمال کرده ایم:

```
# Step 3-1: Convert text sequences to numerical sequences using Tokenizer
def tokenize_text_sequences(X_train, X_test):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(X_train)

    # Convert text sequences to numerical sequences
    train_sequences = tokenizer.texts_to_sequences(X_train)
    test_sequences = tokenizer.texts_to_sequences(X_test)

    return tokenizer, train_sequences, test_sequences
```

مرحله ۳-۲-

در نهایت ماکسیم گیری انجام شده و به اندازه آن پدینگ اعمال شده:

```
# Step 3-2: Pad sequences to have a consistent length
def pad_text_sequences(train_sequences, test_sequences):
    max_length = max(max(len(seq) for seq in train_sequences), max(len(seq) for seq in
test_sequences))

    # Pad sequences
    padded_train_sequences = pad_sequences(train_sequences, maxlen=max_length, padding='post')
    padded_test_sequences = pad_sequences(test_sequences, maxlen=max_length, padding='post')

    return padded_train_sequences, padded_test_sequences, max_length
```

مرحله ۴- تعبیه کلمات

در این مرحله طبق کد نمونه فایل مورد نظر دانلود شده و سپس امبدنیگ بر روی کلمات اعمال شده تا کلمات به بردار تبدیل شوند.

```
# Step 4: Download word embeddings and create an embedding matrix
def download_and_create_embedding_matrix(tokenizer):

    embedding_file_path = f'embeddings/wiki-news-300d-1M.vec'

    if not os.path.isfile(embedding_file_path):
        print('Downloading word vectors ... ')
        url = f'https://dl.fbaipublicfiles.com/fasttext/vectors-english/wiki-news-300d-1M.vec.zip'
        urllib.request.urlretrieve(url, f'wiki-news-300d-1M.vec.zip')

        print('Unzipping ... ')
        with zipfile.ZipFile(f'wiki-news-300d-1M.vec.zip', 'r') as zip_ref:
            zip_ref.extractall('embeddings')
        print('done.')

        os.remove(f'wiki-news-300d-1M.vec.zip')

    embeddings_index = {}
    with open(embedding_file_path, encoding='utf-8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs

    embedding_matrix = np.zeros((len(tokenizer.word_index) + 1, 300))
    for word, i in tokenizer.word_index.items():
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

    return embedding_matrix
```

مرحله ۵ ساخت مدل دسته بند

مرحله ۵-۱-

در این مرحله مطابق اطلاعات داده شده مدل را آموزش دادیم و هیستوری را جهت نمایش نمودار بازگشت داده ایم:

```
# Step 5-1: Build and train a GRU model
def build_and_train_gru_model(embedding_matrix, max_length, input_dim):
    model = Sequential()
    model.add(Embedding(input_dim=input_dim, output_dim=embedding_matrix.shape[1],
```

```
                        weights=[embedding_matrix], input_length=max_length, trainable=False))
    model.add(GRU(units=32, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(1, activation='sigmoid'))

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

    # Train the model
    history = model.fit(padded_train_sequences, y_train.iloc[selected_samples_train],
                        epochs=10, batch_size=64, validation_split=0.2)

    return model, history
```

خروجی کنسول این مرحله:

```
Epoch 1/10
4/4 [==============================] - 10s 2s/step - loss: 0.6925 - accuracy: 0.5292 - val_loss: 0.6911 - val_accuracy: 0.5500
Epoch 2/10
4/4 [==============================] - 6s 2s/step - loss: 0.6901 - accuracy: 0.5542 - val_loss: 0.6897 - val_accuracy: 0.5500
Epoch 3/10
4/4 [==============================] - 6s 2s/step - loss: 0.6890 - accuracy: 0.5542 - val_loss: 0.6887 - val_accuracy: 0.5500
Epoch 4/10
4/4 [==============================] - 7s 2s/step - loss: 0.6882 - accuracy: 0.5542 - val_loss: 0.6882 - val_accuracy: 0.5500
Epoch 5/10
4/4 [==============================] - 10s 3s/step - loss: 0.6874 - accuracy: 0.5542 - val_loss: 0.6882 - val_accuracy: 0.5500
Epoch 6/10
4/4 [==============================] - 9s 2s/step - loss: 0.6875 - accuracy: 0.5542 - val_loss: 0.6883 - val_accuracy: 0.5500
Epoch 7/10
4/4 [==============================] - 9s 2s/step - loss: 0.6877 - accuracy: 0.5542 - val_loss: 0.6884 - val_accuracy: 0.5500
Epoch 8/10
4/4 [==============================] - 9s 2s/step - loss: 0.6872 - accuracy: 0.5542 - val_loss: 0.6883 - val_accuracy: 0.5500
Epoch 9/10
4/4 [==============================] - 7s 2s/step - loss: 0.6879 - accuracy: 0.5542 - val_loss: 0.6882 - val_accuracy: 0.5500
Epoch 10/10
4/4 [==============================] - 7s 2s/step - loss: 0.6873 - accuracy: 0.5542 - val_loss: 0.6882 - val_accuracy: 0.5500
```

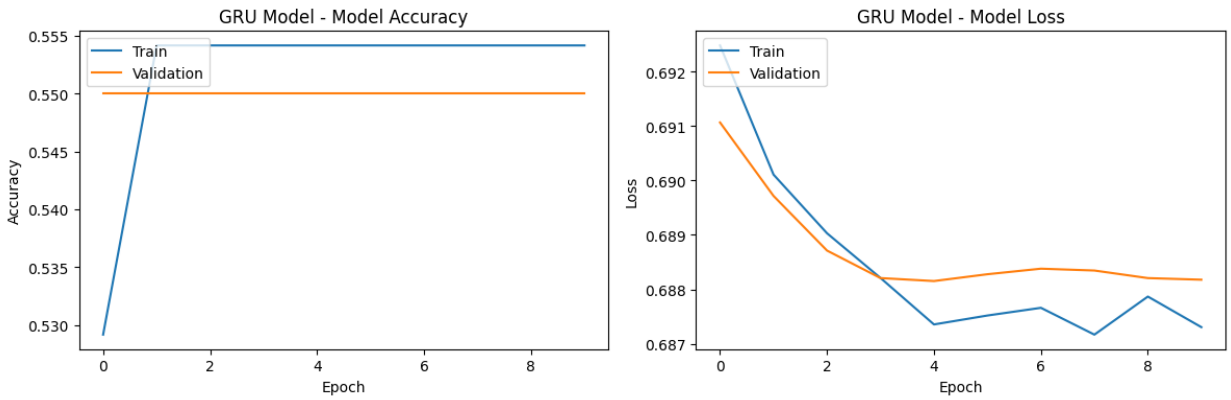
مرحله ۲-۵-

جهت نمایش نمودار از هیستوری مرحله قبل استفاده شده:

```
# Step 5-2: Plot loss and accuracy for the GRU model
def plot_metrics(history, title):
    plt.figure(figsize=(12, 4))

    # Plot training & validation accuracy values
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
```

```
    plt.title(title + ' - Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Train', 'Validation'], loc='upper left')

    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title(title + ' - Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.tight_layout()
    plt.show()

```

خروجی:



مرحله ۵-۳-

در این مرحله نیز مطابق اطلاعات داده شده آموزش مدل انجام شده:

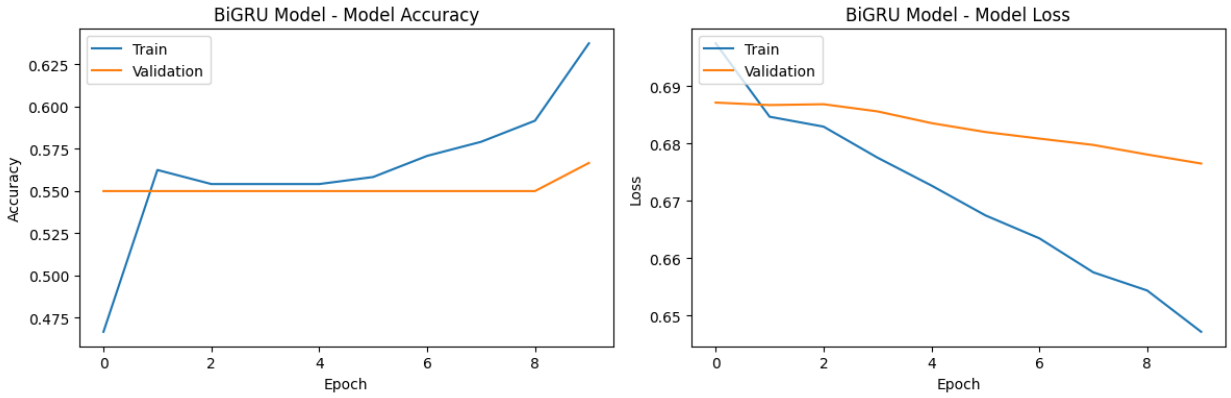
```
# Step 5-3: Build and train a BiGRU model
def build_and_train_bigru_model(embedding_matrix, max_length, input_dim):
    model = Sequential()
    model.add(Embedding(input_dim=input_dim, output_dim=embedding_matrix.shape[1],
                        weights=[embedding_matrix], input_length=max_length, trainable=False))
    model.add(Bidirectional(GRU(units=32, dropout=0.2, recurrent_dropout=0.2)))
    model.add(Dense(1, activation='sigmoid'))

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

    # Train the model
    history = model.fit(padded_train_sequences, y_train.iloc[selected_samples_train],
                        epochs=10, batch_size=64, validation_split=0.2)

    return model, history
```

خروجی کنسول این مرحله:

```
 1. Epoch 1/10
 2. 4/4 [==============================] - 30s 6s/step - loss: 0.6975 - accuracy: 0.4667 - val_loss:
0.6871 - val_accuracy: 0.5500
 3. Epoch 2/10
 4. 4/4 [==============================] - 22s 5s/step - loss: 0.6846 - accuracy: 0.5625 - val_loss:
0.6867 - val_accuracy: 0.5500
 5. Epoch 3/10
 6. 4/4 [==============================] - 20s 5s/step - loss: 0.6829 - accuracy: 0.5542 - val_loss:
0.6868 - val_accuracy: 0.5500
 7. Epoch 4/10
 8. 4/4 [==============================] - 23s 6s/step - loss: 0.6775 - accuracy: 0.5542 - val_loss:
0.6856 - val_accuracy: 0.5500
 9. Epoch 5/10
10. 4/4 [==============================] - 23s 6s/step - loss: 0.6726 - accuracy: 0.5542 - val_loss:
0.6835 - val_accuracy: 0.5500
11. Epoch 6/10
12. 4/4 [==============================] - 22s 5s/step - loss: 0.6674 - accuracy: 0.5583 - val_loss:
0.6820 - val_accuracy: 0.5500
13. Epoch 7/10
14. 4/4 [==============================] - 23s 6s/step - loss: 0.6634 - accuracy: 0.5708 - val_loss:
0.6808 - val_accuracy: 0.5500
15. Epoch 8/10
16. 4/4 [==============================] - 22s 5s/step - loss: 0.6575 - accuracy: 0.5792 - val_loss:
0.6797 - val_accuracy: 0.5500
17. Epoch 9/10
18. 4/4 [==============================] - 22s 6s/step - loss: 0.6543 - accuracy: 0.5917 - val_loss:
0.6781 - val_accuracy: 0.5500
19. Epoch 10/10
20. 4/4 [==============================] - 24s 6s/step - loss: 0.6471 - accuracy: 0.6375 - val_loss:
0.6765 - val_accuracy: 0.5667
```

مرحله ۵-۴-

در این مرحله با استفاده از تابع plot_metrics **که در مرحله ۵-۳ نوشته شده بود خروجی نموداری گزارش شده:**



در نهایت کد های مربوط به اجرای توابع به شرح زیر می باشد:

```
file_path = 'IMDB_Dataset.csv'
df = load_dataset(file_path)

# step1-1
convert_labels(df)

# step1-2
remove_duplicates(df)

# step1-3
df = random_selection(df)

# step1-4
df = get_polarity(df)

# step1-5
plot_sentiment_histograms(df)

# step1-6
i = 0
for index, row in df.iterrows():
    print(i)
    i += 1
    df.at[index, 'review'] = preprocess_text(row['review'])

df.to_csv('preprocessed_and_split_IMDB_Dataset.csv', index=False)

# step1-7

file_path = 'preprocessed_and_split_IMDB_Dataset.csv'
df = load_dataset(file_path)
X_train, X_test, y_train, y_test = split_dataset(df)

# # step2-1
bow_matrix_train, bow_matrix_test = create_bow_representation(X_train, X_test)

# # Step 2-2
train_clusters, test_clusters = kmeans_clustering(bow_matrix_train, bow_matrix_test)

# # Step 2-3
visualize_clusters(bow_matrix_train, train_clusters, title='Training Set Clusters (t-SNE
Visualization)')

# # Step 2-4
num_samples_per_cluster = 100  # Adjust the number of samples as needed
selected_samples_train = select_uniform_sample(bow_matrix_train, train_clusters,
num_samples_per_cluster)
selected_samples_test = select_uniform_sample(bow_matrix_test, test_clusters,
num_samples_per_cluster)

# # Step 3-1
tokenizer, train_sequences, test_sequences =
tokenize_text_sequences(X_train.iloc[selected_samples_train], X_test.iloc[selected_samples_test])

# # Step 3-2
```

```
52. padded_train_sequences, padded_test_sequences, max_length = pad_text_sequences(train_sequences,
test_sequences)
53.
54. # step 4
55. embedding_matrix = download_and_create_embedding_matrix(tokenizer)
56.
57. # Step 5-1 (GRU Model)
58. gru_model, gru_history = build_and_train_gru_model(embedding_matrix, max_length,
input_dim=len(tokenizer.word_index) + 1)
59.
60. # Step 5-2 (Plot Metrics for GRU Model)
61. plot_metrics(gru_history, 'GRU Model')
62.
63. # Step 5-3 (BiGRU Model)
64. bigru_model, bigru_history = build_and_train_bigru_model(embedding_matrix, max_length,
input_dim=len(tokenizer.word_index) + 1)
65.
66. # Step 5-4 (Plot Metrics for BiGRU Model)
67. plot_metrics(bigru_history, 'BiGRU Model')
```

همچنین دایرکتوری به شرح زیر می باشد: