Objective of this Project: Sentiment Analysis is one of the most used applications of data science in Real-World Analysis. As the whole world is dependent on Social Media, user opinion and explanation can help us to understand their sentiments and intentions. The tweets have been annotated as (0 = negative, 4 = positive) and they can be used to detect sentiment our objective is to predict the labels on the given test dataset

Dataset: We have extracted the data from the Kaggle that contains 1,600,000 tweets and have been annoted as (0 = negative, 4 = positive).

This is the sentiment140 dataset. It contains 1,600,000 tweets extracted using the twitter api . The tweets have been annotated (0 = negative,4 = positive) and they can be used to detect sentiment . It contains the following 6 fields:

- target: the polarity of the tweet (0 = negative 4 = positive)
- ids: The id of the tweet (2087)
- date: the date of the tweet (Sat May 16 23:58:44 UTC 2009)
- flag: The query (lyx). If there is no query, then this value is NO_QUERY.
- user: the user that tweeted (robotickilldozr)
- text: the text of the tweet (Lyx is cool)

The official link regarding the dataset with resources about how it was generated is here The official paper detailing the approach is here

According to the creators of the dataset:

"Our approach was unique because our training data was automatically created, as opposed to having humans manual annotate tweets. In our approach, we assume that any tweet with positive emoticons, like :), were positive, and tweets with negative emoticons, like :(, were negative. We used the Twitter Search API to collect these tweets by using keyword search"

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
ount("/content/drive", force_remount=True).
```

Loading the libraries and the Data

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Read the data set and load it

In [3]:
```
data=pd.read_csv('/content/drive/MyDrive/AI_Final_Project/training.1600000.processed.r
```

Exploratory Data Analysis:Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

It is a good practice to understand the data first and try to gather as many insights from it. EDA is all about making sense of data in hand,before getting them dirty with it.

In [4]:
```
data.head() #prints the five top records of data
```

Out[4]:

	0	1	2	3	4	5
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....

In [5]:
```
data.columns = ["Label","id","date","flag","user","tweet"]
data.head() #prints the columns or features in the data
```

Out[5]:

	Label	id	date	flag	user	tweet
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....

In [6]:
```
df = data[["Label", "tweet"]]#we are only selecting the useful columns
df.head()
```

Out[6]:

	Label	tweet
0	0	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	is upset that he can't update his Facebook by ...
2	0	@Kenichan I dived many times for the ball. Man...
3	0	my whole body feels itchy and like its on fire
4	0	@nationwideclass no, it's not behaving at all....

In [7]:
```
# Lets check the total number of rows and column in the dataframe
df.shape
```

Out[7]:
```
(1600000, 2)
```

In [8]:
```
df['Label'].unique()#printing the unique labels
```

Out[8]:
```
array([0, 4])
```

In [9]:
```
# Currently (0=negative,4=Positive) changing the notation to (0=Negative,1=Positive)
#so that we can understand the data
df['Label']=df['Label'].replace(4,1)
df.head(10)#printing the first 10 record
```

```
<ipython-input-9-7e48774ffb5b>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df['Label']=df['Label'].replace(4,1)
```

Out[9]:

	Label	tweet
0	0	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	is upset that he can't update his Facebook by ...
2	0	@Kenichan I dived many times for the ball. Man...
3	0	my whole body feels itchy and like its on fire
4	0	@nationwideclass no, it's not behaving at all....
5	0	@Kwesidei not the whole crew
6	0	Need a hug
7	0	@LOLTrish hey long time no see! Yes.. Rains a...
8	0	@Tatiana_K nope they didn't have it
9	0	@twittera que me muera ?

In [10]:
```
#finding the dimension of train and test dataset

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

```
#Train set has 1600,000 tweets
```

Out[10]:
```
0    800000
1    800000
Name: Label, dtype: int64
```

In [11]:
```
# Lets check the datatypes of columns in the dataframe
df.dtypes
```

Out[11]:
```
Label     int64
tweet    object
dtype: object
```

In [12]:
```
#Checking for Null values

np.sum(df.isnull().any(axis=1))
```

Out[12]:
```
0
```

In [13]:
```
#prints the data information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600000 entries, 0 to 1599999
Data columns (total 2 columns):
 #   Column  Non-Null Count    Dtype 
---  ------  --------------    ----- 
 0   Label   1600000 non-null  int64 
 1   tweet   1600000 non-null  object
dtypes: int64(1), object(1)
memory usage: 24.4+ MB
```

In [14]:
```
df.describe()

#describes the dataset
```

Out[14]:

	Label
count	1600000.0
mean	0.5
std	0.5
min	0.0
25%	0.0
50%	0.5
75%	1.0
max	1.0

In [15]:
```
#finding the length of postive and negative tweets
pos_tweet = df[df['Label'] == 1]
neg_tweet = df[df['Label'] == 0]
print(len(pos_tweet), len(neg_tweet))
```

```
800000 800000
```

In [16]:
```
# finding  the distribution of tweets in the train set

temp = df.groupby('Label').count()['tweet'].reset_index().sort_values(by='tweet',ascen
temp.style.background_gradient(cmap='Purples')
```

Out[16]:

	Label	tweet
0	0	800000
1	1	800000

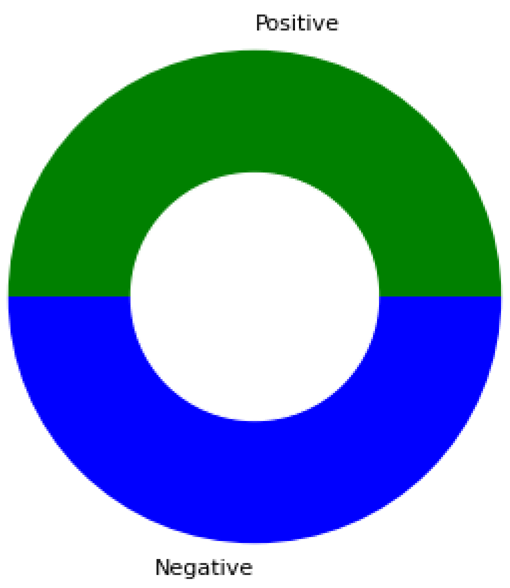
In [17]:
```
# Distribution of different classes in sentiment
def count_values_in_column(data,feature):
    total=data.loc[:,feature].value_counts(dropna=False)
    percentage=round(data.loc[:,feature].value_counts(dropna=False,normalize=True)*10
    return pd.concat([total,percentage],axis=1,keys=["Total","Percentage"])
count_values_in_column(df,"Label")
```

Out[17]:

	Total	Percentage
0	800000	50.0
1	800000	50.0

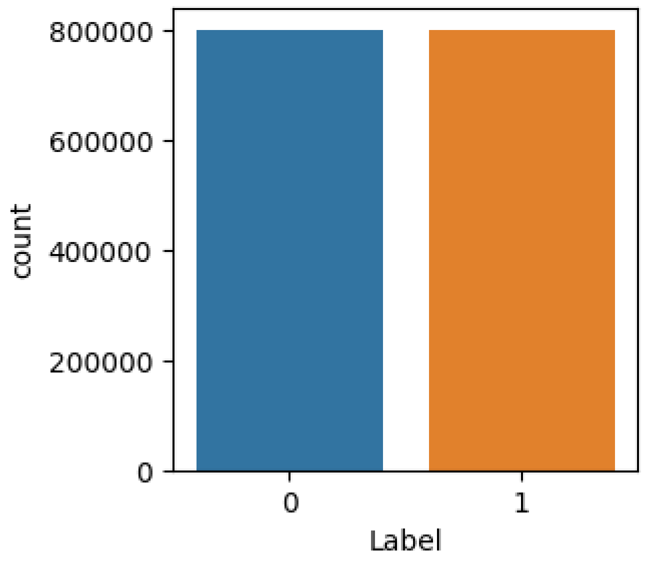
In [18]:
```
#create data for Pie Chart
plt.figure(figsize=(10, 5), dpi=80)
pichart = count_values_in_column(df,"Label")
names= ["Positive","Negative"]
size=pichart["Percentage"]

# Create a circle for the center of the plot
my_circle=plt.Circle( (0,0), 0.5, color='white')
plt.pie(size, labels=names, colors=['green','blue'])
p=plt.gcf()
p.gca().add_artist(my_circle)
plt.show()
```



```
In [19]: plt.figure(figsize=(3,3))#equal number of positive and negative tweets
         sns.countplot(x='Label',data=df)
```

```
Out[19]: <Axes: xlabel='Label', ylabel='count'>
```



Data Preprocessing

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format.

Steps Involved in Data Preprocessing:.

• The stop words were removed as these words repeatedly appear in the text however, they do not add much value to it. Removing them will shift the focus to more of unique data that holds significant information.

• All the special characters/punctuation were removed in the preprocessing step, which is the important step in the data preprocessing, and a cleaner tweet was generated.

• Stemming/Lematization: Stemming and Lemmatization both generate the foundation sort of the inflected words and therefore the only difference is that stem may not be an actual word whereas, lemma is an actual language word. Stemming follows an algorithm with steps to perform on the words which makes it faster. adjustable >> adjust #stemming was >> (to) be # Lematization

```
In [20]: import nltk
         nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
Out[20]: True
```

```
In [21]: import re
         from nltk.corpus import stopwords
         from nltk.stem import SnowballStemmer
```

```
stop_words = stopwords.words('english')
stemmer = SnowballStemmer('english')

def preprocess(text, stem=False):
    text = re.sub(r"@[A-Za-z0-9_]+", "", str(text))  # remove @mentions
    text = re.sub(r"http\S+", "", text)  # remove URLs
    text = re.sub(r"www\S+", "", text)  # remove URLs
    text = re.sub(r"[^A-Za-z0-9_]", " ", text)  # remove non-alphanumeric characters
    text = text.lower().strip()  # convert to lowercase and remove leading/trailing wh
    tokens = [token for token in text.split() if token not in stop_words]  # remove st
    if stem:
        tokens = [stemmer.stem(token) for token in tokens]  # apply stemming
    return " ".join(tokens)
```

In [22]:
```
df.tweet = df.tweet.apply(lambda x: preprocess(x))
```

```
<ipython-input-22-86bc3436b33a>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df.tweet = df.tweet.apply(lambda x: preprocess(x))
```

In [23]:
```
df.head(5)#displaying the cleaner tweet
```

Out[23]:

	Label	tweet
0	0	bummer shoulda got david carr third day
1	0	upset update facebook texting might cry result...
2	0	dived many times ball managed save 50 rest go ...
3	0	whole body feels itchy like fire
4	0	behaving mad see

In [24]:
```
lab_to_sentiment = {0:"Negative", 1:"Positive"}
def label_decoder(label):
  return lab_to_sentiment[label]
df.Label = df.Label.apply(lambda x: label_decoder(x))
df.head()
```

```
<ipython-input-24-da467bb1c6a0>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df.Label = df.Label.apply(lambda x: label_decoder(x))
```

Out[24]:

	Label	tweet
0	Negative	bummer shoulda got david carr third day
1	Negative	upset update facebook texting might cry result...
2	Negative	dived many times ball managed save 50 rest go ...
3	Negative	whole body feels itchy like fire
4	Negative	behaving mad see

Data Visualization:

A) Understanding the common words used in the tweets: WordCloud. A wordcloud is a visualization wherein the most frequent words appear in large size and the less frequent words appear in smaller sizes.

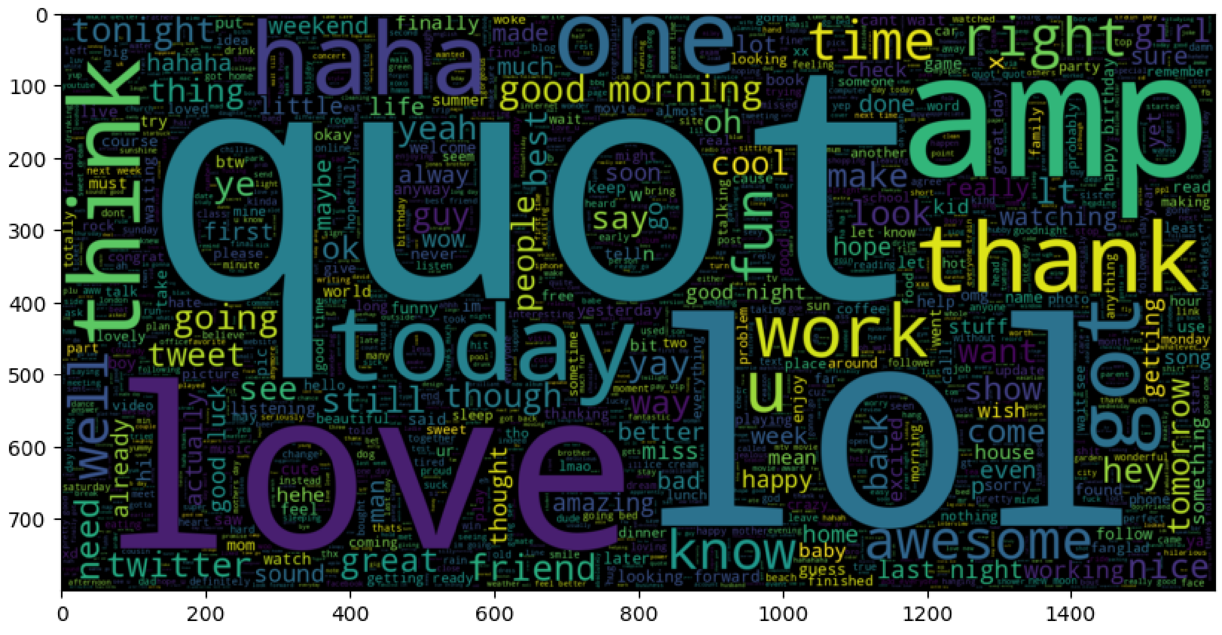
Let's visualize all the words our data using the wordcloud plot.

Positive Words

In [25]:
```
from wordcloud import WordCloud

plt.figure(figsize = (10,10))
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(df[d
plt.imshow(wc , interpolation = 'bilinear')
```

Out[25]:
```
<matplotlib.image.AxesImage at 0x7f5d22dba950>
```



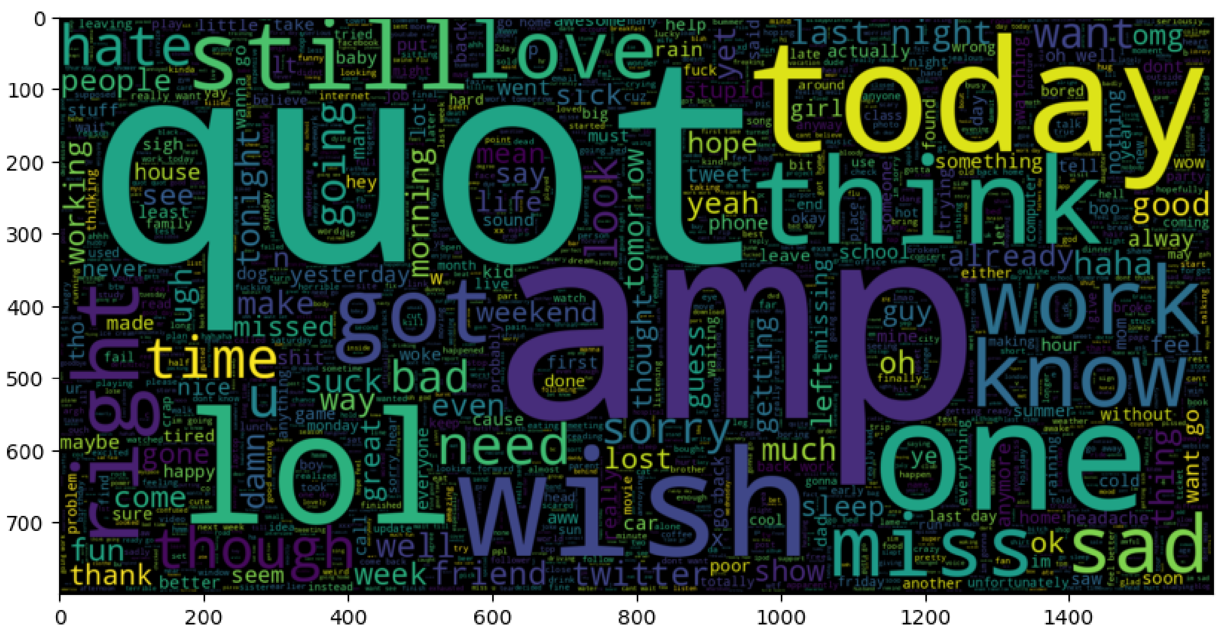
Negtaive words

In [26]:
```
from wordcloud import WordCloud

plt.figure(figsize = (10,10))
```

```
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(df[df
plt.imshow(wc , interpolation = 'bilinear')
```

Out[26]:
```
<matplotlib.image.AxesImage at 0x7f5d2077e680>
```



Model Preparation

Splitting the dataset

In [27]:
```
#let us split the train and test data
TRAIN_SIZE = 0.8#80% of the data will be used for training and the remaining 20% will
MAX_NB_WORDS = 100000# tokenizer will only consider the top 100,000 most frequent word
MAX_SEQUENCE_LENGTH = 30 #maximum length of a sequence (or input text) will be 30 word
```

In [28]:
```
from sklearn.model_selection import train_test_split
```

In [29]:
```
# splitting of the data
train_data, test_data = train_test_split(df, test_size=0.2,
                                         random_state=1) # Splits Dataset into Trainin
print("Train Data size:", len(train_data))
print("Test Data size", len(test_data))
```

```
Train Data size: 1280000
Test Data size 320000
```

In [30]:
```
train_data.head(5)
```

Out[30]:

	Label	tweet
1556092	Positive	use like
868905	Positive	almost home aaand need pee rather badly also r...
218471	Negative	dropping marmite cheese covered bread floor fa...
620327	Negative	issues xfire broadcast cancelled
981867	Positive	ask hermes heritage complex small eatery nothi...

Tokenization:

Tokenization is a process of breaking down a character sequence into individual pieces called tokens, while discarding certain characters like punctuation. Tokenization is commonly used in natural language processing and machine learning tasks.

Tokenization can be performed using a tokenizer, which creates a token for each word in the data corpus and maps them to an index using a dictionary. The resulting index is known as the "word index," which contains the index for each word in the data corpus.

The "vocab size" represents the total number of unique words in the data corpus. It is an important parameter to consider when building machine learning models that require vectorizing text data. A larger vocab size indicates a more diverse range of words in the corpus and can potentially improve the model's performance, but also increase computational complexity.

In [31]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
```

In [32]:
```
tokenizer = Tokenizer()#utility class used for tokenizing texts
tokenizer.fit_on_texts(train_data.tweet)#tokenizer will convert each word in the text
word_index = tokenizer.word_index#a dictionary containing the mapping of words to thei
vocab_size = len(tokenizer.word_index) + 1#calculates the total number of unique words
```

In [33]:
```
print (vocab_size)
```

```
244373
```

Padding: Padding is done after tokenization to convert the word sequence into a list of arrays of the same length. Neural networks require fixed-length inputs, and hence padding is done to ensure that all input sequences have the same length.

In the context of NLP, padding refers to adding extra zeros or some other value to the beginning or end of shorter sequences to make them of the same length as the longest sequence. This ensures that all sequences have the same length, and the neural network can process them in a batch.

Therefore, padding ensures that each input sequence has the same length, and the neural network can process them efficiently in batches.

In [34]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences
X_train = pad_sequences(tokenizer.texts_to_sequences(train_data.tweet),
                        maxlen =30)
X_test = pad_sequences(tokenizer.texts_to_sequences(test_data.tweet),
                       maxlen = 30)
print("Training X Shape:",X_train.shape)
print("Testing X Shape:",X_test.shape)
```

```
Training X Shape: (1280000, 30)
Testing X Shape: (320000, 30)
```

Label Encoding: The objective is to categorize data points as either 0 or 1. Since Machine Learning algorithms cannot interpret textual representations of these values, we need to convert them into a numerical format, a process known as encoding.

In [35]:
```
labels = train_data.Label.unique().tolist()# extracts all unique labels from the Label
```

In [36]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoder.fit(train_data.Label.to_list())#Fits the encoder on the training data's "Label
y_train = encoder.transform(train_data.Label.to_list())#Uses the fitted encoder to tra
y_test = encoder.transform(test_data.Label.to_list())#Uses the fitted encoder to trans
y_train = y_train.reshape(-1,1)#Reshapes the y_train array from a 1D array to a 2D arr
y_test = y_test.reshape(-1,1)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
y_train shape: (1280000, 1)
y_test shape: (320000, 1)
```

Word embedding

Word embedding is a technique used in natural language processing to create a numerical representation of words in a way that captures their meaning and context. It allows words with similar meanings to have similar representations in the embedding space. This technique has been a significant breakthrough in deep learning for natural language processing tasks, as it has improved the accuracy of models by enabling them to better understand and interpret the meaning of words in a text. we will be using pretrained glove word embedding

The pretrained Word Embedding like GloVe & Word2Vec gives more insights for a word which can be used for classification

In [37]:
```
embeddings_index = {}# creates an empty dictionary to store the word embeddings

with open('/content/drive/MyDrive/AI_Final_Project/glove.6B.300d.txt', encoding='utf-8
    for line in f:
        values = line.split()#splits the line by whitespace characters
        word = values[0] #extracts the first element of values
        coefs = np.asarray(values[1:], dtype='float32')# converts the rest of the elem
        embeddings_index[word] = coefs
    f.close()

print('The total number of word vectors found in the GloVe file is' ,len(embeddings_in
```

```
The total number of word vectors found in the GloVe file is 400000
```

Creating the Embedding Matrix Creating the embedding matrix is necessary to initialize the embedding layer of the neural network with pre-trained GloVe word embeddings. This is done to leverage the pre-trained word embeddings that are trained on a large corpus of text data to improve the performance of the neural network. The embedding matrix is initialized with zeros and is filled with GloVe embeddings for words that are present in the training data.

In [38]:
```
EMBEDDING_DIM=300
embedding_matrix = np.zeros((vocab_size, EMBEDDING_DIM))
for word, i in word_index.items():
  embedding_vector = embeddings_index.get(word)#gets the pre-trained GloVe embedding v
  if embedding_vector is not None:#checks if the embedding vector exists for the curre
    embedding_matrix[i] = embedding_vector
print("Embedding Matrix Shape:", embedding_matrix.shape)
```

```
Embedding Matrix Shape: (244373, 300)
```

In [39]:
```
MAX_SEQUENCE_LENGTH=30
```

In [40]:
```
embedding_layer = tf.keras.layers.Embedding(vocab_size,
                                             EMBEDDING_DIM,
                                             weights=[embedding_matrix],
                                             input_length=MAX_SEQUENCE_LENGTH,
                                             trainable=False)
```

Model Training

CNN Model Convolutional Neural Networks (CNNs) have been successfully applied to sentiment analysis tasks, where the goal is to classify a piece of text as positive, negative, or neutral.

CNNs can be used to extract useful features from raw text data, which can then be used for sentiment classification. The input data for the CNN would typically be a sequence of words, such as a sentence or a paragraph, that needs to be classified into one of the sentiment categories.

In [41]:
```
import tensorflow as tf
from tensorflow.keras.layers import Input, Embedding, SpatialDropout1D, Conv1D, MaxPoo
from tensorflow.keras.callbacks import ReduceLROnPlateau,EarlyStopping
from tensorflow.keras.layers import BatchNormalization

# Define the model architecture with additional layers and batch normalization
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedding_sequences = embedding_layer(sequence_input)
x = SpatialDropout1D(0.2)(embedding_sequences)
x = Conv1D(64, 5, activation='relu')(x)
x = BatchNormalization()(x)
x = Conv1D(64, 5, activation='relu')(x)
x = BatchNormalization()(x)
x = MaxPooling1D(pool_size=2)(x)
x = Conv1D(64, 5, activation='relu')(x)
x = BatchNormalization()(x)
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
```

```
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
x = BatchNormalization()(x)
outputs = Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(sequence_input, outputs)

# Compile the model with binary crossentropy loss and Adam optimizer
model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])

# Train the model with early stopping
early_stopping = EarlyStopping(monitor="val_loss", patience=5, restore_best_weights=Tr
reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=3, min_delta=0.

history = model.fit(X_train, y_train, batch_size=64, epochs=6, validation_split=0.2,
                    callbacks=[early_stopping, reduce_lr])
```

```
Epoch 1/6
16000/16000 [==============================] - 132s 8ms/step - loss: 0.5723 - accurac
y: 0.6976 - val_loss: 0.5074 - val_accuracy: 0.7475 - lr: 1.0000e-04
Epoch 2/6
16000/16000 [==============================] - 121s 8ms/step - loss: 0.5124 - accurac
y: 0.7452 - val_loss: 0.4916 - val_accuracy: 0.7600 - lr: 1.0000e-04
Epoch 3/6
16000/16000 [==============================] - 122s 8ms/step - loss: 0.4970 - accurac
y: 0.7560 - val_loss: 0.4828 - val_accuracy: 0.7659 - lr: 1.0000e-04
Epoch 4/6
16000/16000 [==============================] - 124s 8ms/step - loss: 0.4873 - accurac
y: 0.7628 - val_loss: 0.4783 - val_accuracy: 0.7686 - lr: 1.0000e-04
Epoch 5/6
16000/16000 [==============================] - 123s 8ms/step - loss: 0.4806 - accurac
y: 0.7673 - val_loss: 0.4783 - val_accuracy: 0.7684 - lr: 1.0000e-04
Epoch 6/6
16000/16000 [==============================] - 123s 8ms/step - loss: 0.4758 - accurac
y: 0.7701 - val_loss: 0.4732 - val_accuracy: 0.7713 - lr: 1.0000e-04
```

Model Evaluation on the Test Dataset for CNN

In [42]:
```
# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test loss: {loss:.3f}')
print(f'Test accuracy: {accuracy*100:.2f}%')
```

```
10000/10000 [==============================] - 36s 4ms/step - loss: 0.4696 - accurac
y: 0.7736
Test loss: 0.470
Test accuracy: 77.36%
```

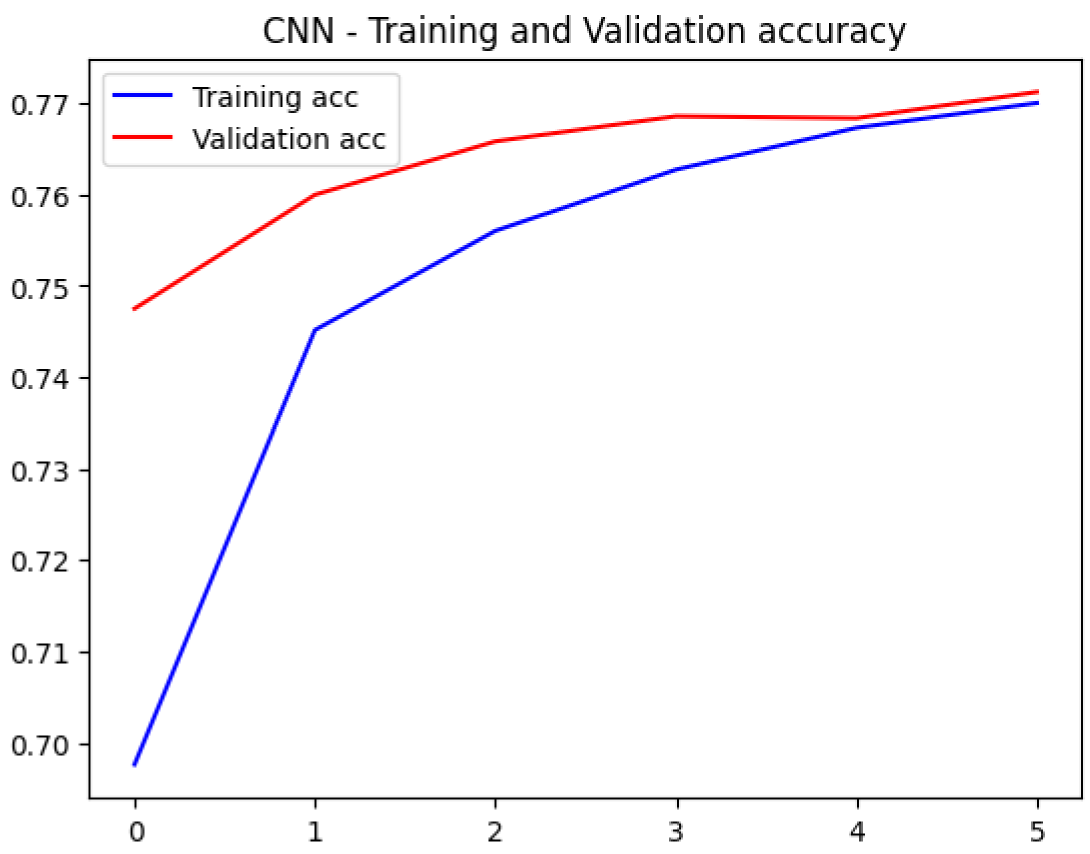
Plotting the Training and Validation Accuracy and loss curves for CNN

In [43]:
```
acc,  val_acc  = history.history['accuracy'], history.history['val_accuracy']

epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('CNN - Training and Validation accuracy')
plt.legend()
plt.figure()
```

Out[43]:
```
<Figure size 640x480 with 0 Axes>
```

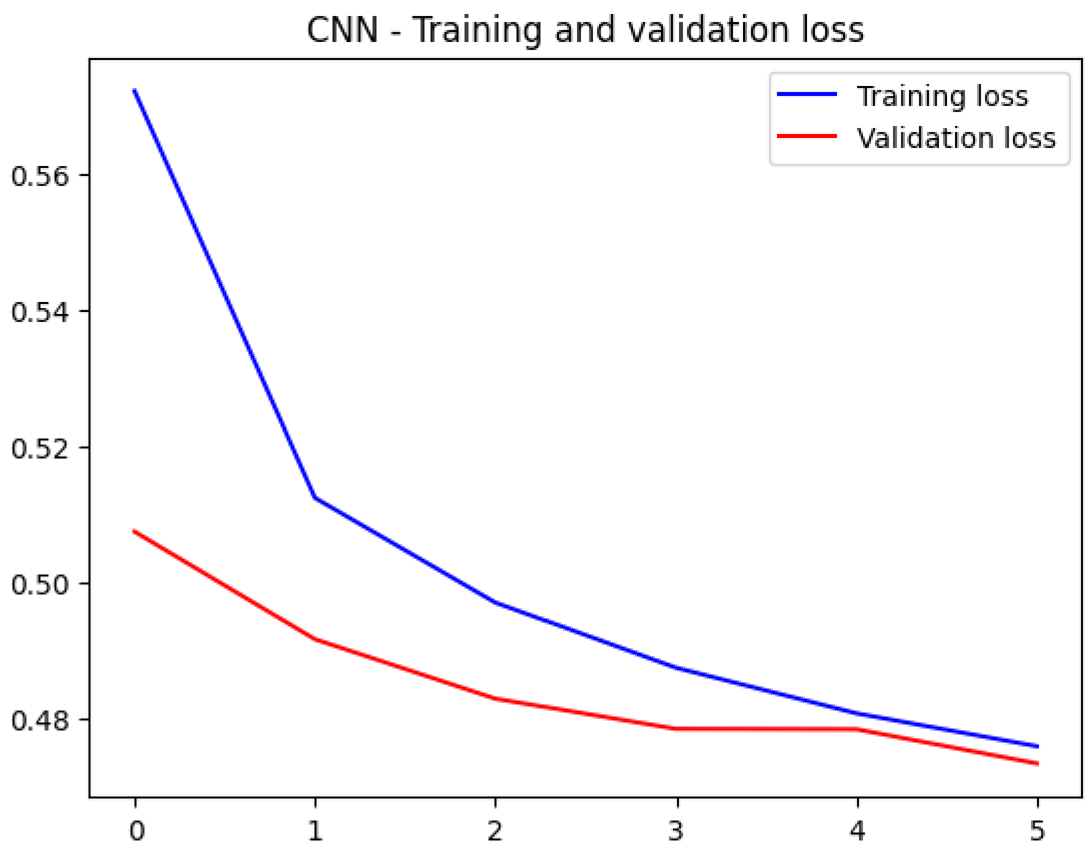


```
<Figure size 640x480 with 0 Axes>
```

In [44]:
```
loss, val_loss = history.history['loss'], history.history['val_loss']

plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('CNN - Training and validation loss')
plt.legend()

plt.show()
```



In [45]:
```
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)
```

```
10000/10000 [==============================] - 24s 2ms/step
```

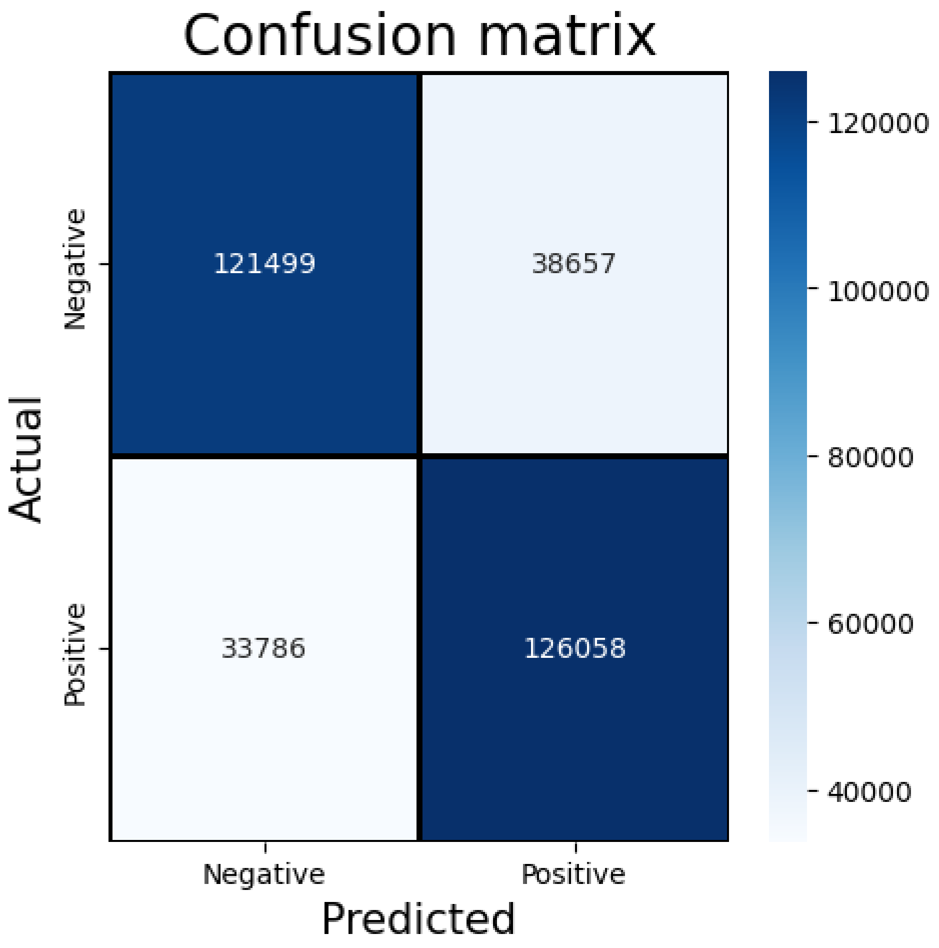
Confusion Matrix

In [46]:
```
import seaborn as sns
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
def plot_confusion_matrix (test_labels, predictions):
  """
  plotting the confusion matrix
  Parameters - test_labels, predictions

  """
  cm = confusion_matrix(y_test, y_pred)
  cm = pd.DataFrame(cm, index = ['0', '1'], columns = ['0', '1'])
  sentiment_classes = ['Negative', 'Positive']
  plt.figure(figsize = (5,5))
  sns.heatmap(
      cm,
      cmap = 'Blues',
      linecolor = 'black',
      linewidth = 1,
      annot = True,
      fmt = 'd',
      xticklabels = sentiment_classes,
      yticklabels = sentiment_classes)

  plt.xlabel('Predicted', fontsize=15)
  plt.ylabel('Actual', fontsize=15)
  plt.title('Confusion matrix', fontsize=20)
```

```
# calling function to visualize the confusion matrix
plot_confusion_matrix(y_test, y_pred)
```



Classification Report

In [47]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.76      0.77    160156
           1       0.77      0.79      0.78    159844

    accuracy                           0.77    320000
   macro avg       0.77      0.77      0.77    320000
weighted avg       0.77      0.77      0.77    320000
```

LSTM: LSTM networks are a type of recurrent neural network (RNN) that are commonly used for natural language processing tasks, such as sentiment analysis. LSTMs are designed to address the issue of vanishing gradients in RNNs, which can make it difficult to capture long-term dependencies in sequential data.

In the context of sentiment analysis, LSTMs can be used to model the context and sequence of a piece of text, such as a review or social media post, in order to predict its sentiment.

```
In [48]: from keras.models import Model
         from keras.layers import Dense, Embedding, LSTM, Input, Dropout, LeakyReLU
         from keras.utils.np_utils import to_categorical
         from tensorflow.keras.callbacks import ReduceLROnPlateau,EarlyStopping

         # Define the model architecture
         sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
         embedding_sequences = embedding_layer(sequence_input)
         x = LSTM(128, dropout=0.2, recurrent_dropout=0.2)(embedding_sequences)

         x = Dense(128)(x)
         x = LeakyReLU(alpha=0.1)(x)
         x = Dropout(0.5)(x)

         x = Dense(64)(x)
         x = LeakyReLU(alpha=0.1)(x)
         x = Dropout(0.5)(x)

         outputs = Dense(1, activation='sigmoid')(x)
         model1 = Model(sequence_input, outputs)

         # Compile the model with binary crossentropy loss and Adam optimizer
         model1.compile(loss='binary_crossentropy',
                       optimizer=tf.keras.optimizers.Adam(1e-4),
                       metrics=['accuracy'])
         # Train the model with early stopping
         early_stopping = EarlyStopping(monitor="val_loss", patience=5, restore_best_weights=Tr
         reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=3, min_delta=0.

         history = model1.fit(X_train, y_train, batch_size=64, epochs=6, validation_split=0.2,
                             callbacks=[early_stopping, reduce_lr])
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the cr
iteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/6
16000/16000 [==============================] - 964s 60ms/step - loss: 0.5351 - accura
cy: 0.7301 - val_loss: 0.4931 - val_accuracy: 0.7582 - lr: 1.0000e-04
Epoch 2/6
16000/16000 [==============================] - 935s 58ms/step - loss: 0.4946 - accura
cy: 0.7596 - val_loss: 0.4784 - val_accuracy: 0.7679 - lr: 1.0000e-04
Epoch 3/6
16000/16000 [==============================] - 919s 57ms/step - loss: 0.4819 - accura
cy: 0.7685 - val_loss: 0.4703 - val_accuracy: 0.7736 - lr: 1.0000e-04
Epoch 4/6
16000/16000 [==============================] - 926s 58ms/step - loss: 0.4743 - accura
cy: 0.7728 - val_loss: 0.4649 - val_accuracy: 0.7771 - lr: 1.0000e-04
Epoch 5/6
16000/16000 [==============================] - 942s 59ms/step - loss: 0.4686 - accura
cy: 0.7766 - val_loss: 0.4620 - val_accuracy: 0.7786 - lr: 1.0000e-04
Epoch 6/6
16000/16000 [==============================] - 930s 58ms/step - loss: 0.4634 - accura
cy: 0.7799 - val_loss: 0.4591 - val_accuracy: 0.7801 - lr: 1.0000e-04
```

Model Evaluation on the Test Dataset for LSTM

```
In [49]: # Evaluate the model on the test set
         loss, accuracy = model1.evaluate(X_test, y_test)
         print(f'Test loss: {loss:.3f}')
         print(f'Test accuracy: {accuracy*100:.2f}%')
```

```
10000/10000 [==============================] - 121s 12ms/step - loss: 0.4554 - accura
cy: 0.7823
Test loss: 0.455
Test accuracy: 78.23%
```

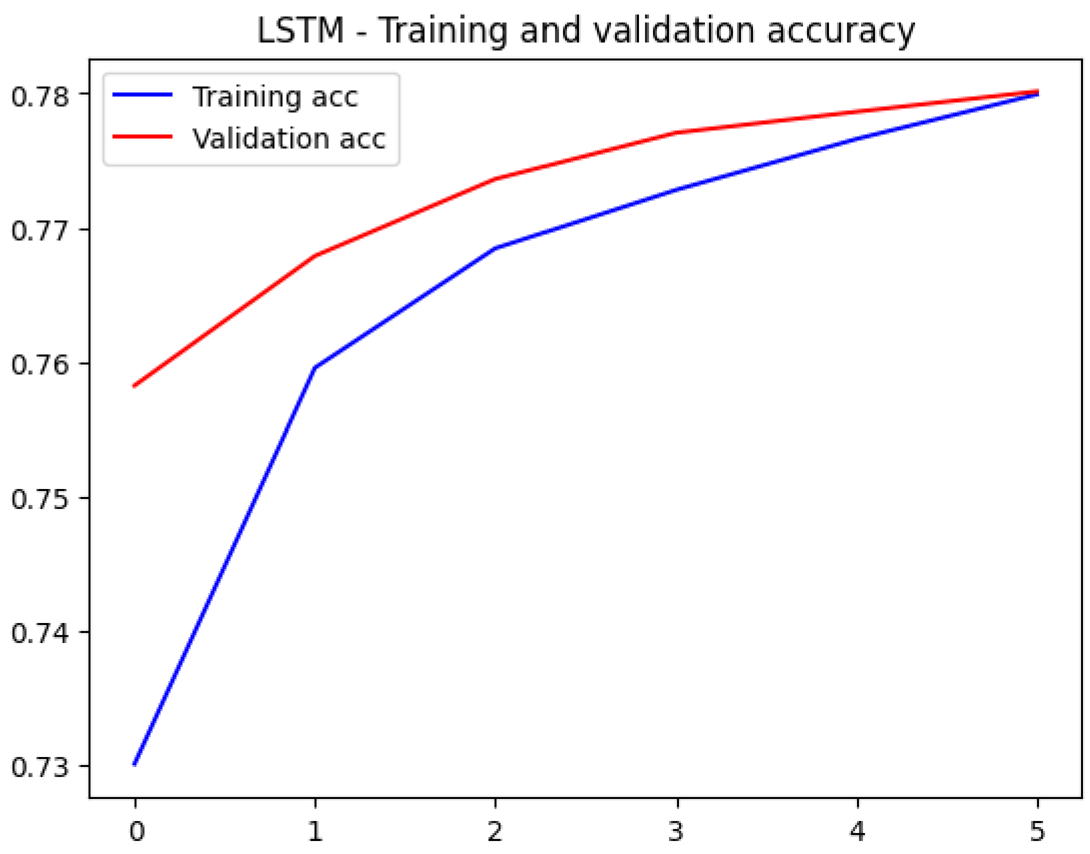
Plotting the Training and Validation Accuracy and loss curves for LSTM

In [50]:
```
acc,  val_acc  = history.history['accuracy'], history.history['val_accuracy']
loss, val_loss = history.history['loss'], history.history['val_loss']
epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('LSTM - Training and validation accuracy')
plt.legend()

plt.figure()
```

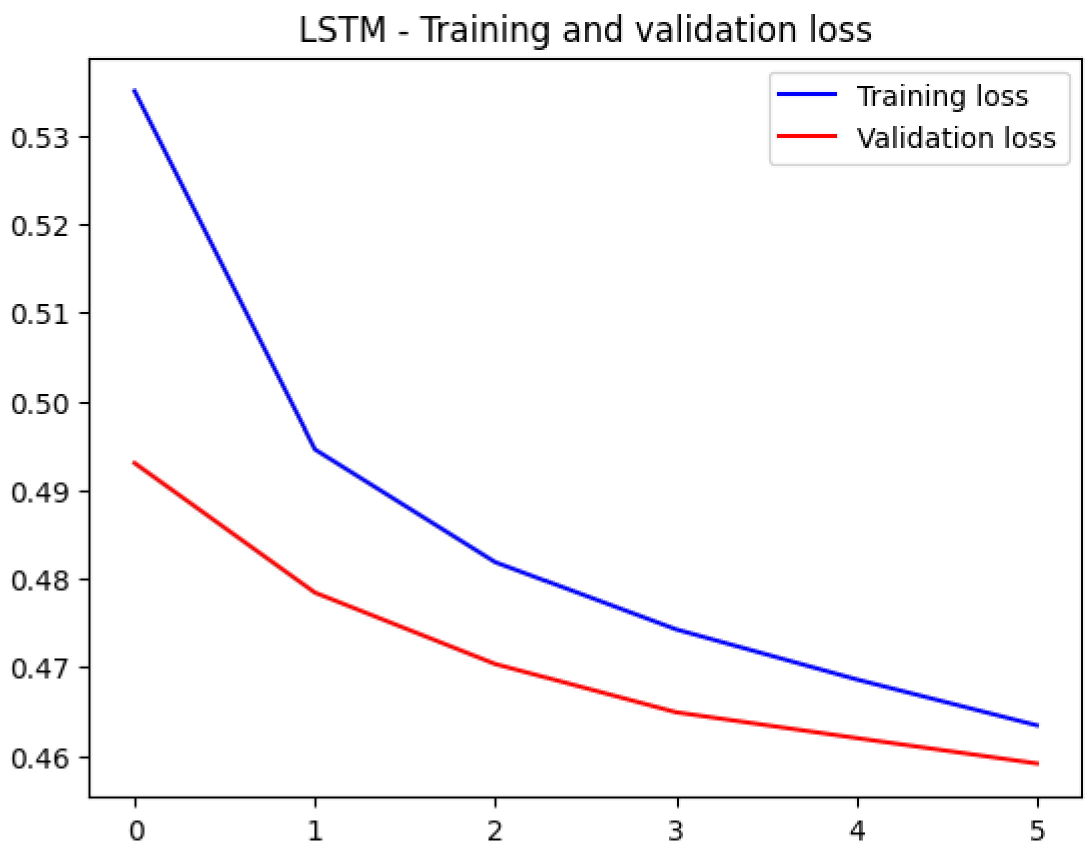
Out[50]:
```
<Figure size 640x480 with 0 Axes>
```



```
<Figure size 640x480 with 0 Axes>
```

In [51]:
```
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('LSTM - Training and validation loss')
plt.legend()

plt.show()
```

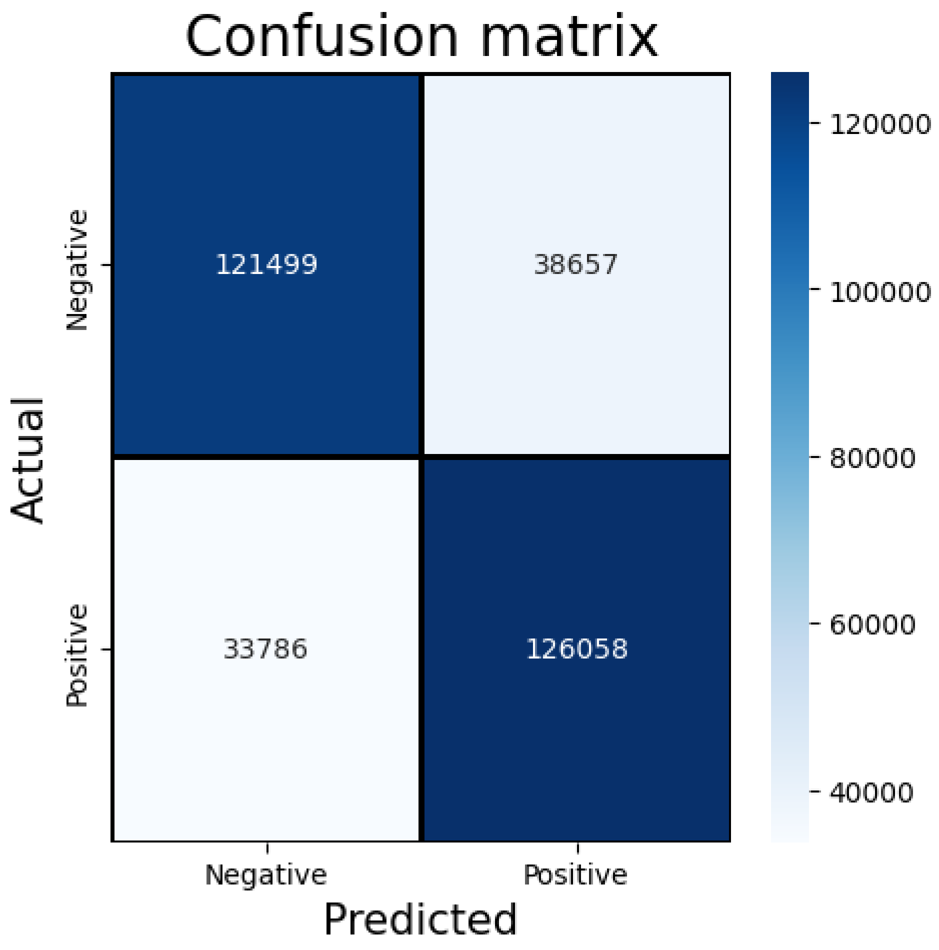


```
In [52]: y_pred1 = model1.predict(X_test)
         y_pred1 = (y_pred1 > 0.5).astype(int)
```

```
10000/10000 [==============================] - 114s 11ms/step
```

Confusion Matrix

```
In [53]: plot_confusion_matrix(y_test, y_pred1)
```



Classification Report

In [54]:
```
print(classification_report(y_test,y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.79      0.77      0.78    160156
           1       0.77      0.80      0.79    159844

    accuracy                           0.78    320000
   macro avg       0.78      0.78      0.78    320000
weighted avg       0.78      0.78      0.78    320000
```