In [1]:
```
#Import packages that will be used for this analysis
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns

from sklearn.datasets import make_classification
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.svm import SVC

import re
from collections import Counter
import spacy
from gensim.corpora.dictionary import Dictionary
from gensim.models.tfidfmodel import TfidfModel
import nltk
from nltk.tokenize import word_tokenize
from collections import Counter
from gensim.models.tfidfmodel import TfidfModel
from nltk.corpus import stopwords
import random
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Embedding, Dense, GlobalAveragePooling1D, LSTM, Dropout, SimpleRNN
from keras.callbacks import History
from sklearn.model_selection import train_test_split
```

In [2]:
```
#Import and concatenate the datasets
amazon = pd.read_csv('amazon_cells_labelled.txt', delimiter='\t', header=None, names=['review', 'rating'])
imbd = pd.read_csv('imdb_labelled.txt', delimiter='\t', header=None, names=['review', 'rating'])
yelp =  pd.read_csv('yelp_labelled.txt', delimiter='\t', header=None, names=['review', 'rating'])
df = pd.concat([amazon, imbd, yelp])
reviews_df = df
reviews_df.reset_index(inplace=True)
reviews_df.sample(10)
```

Out[2]:

	index	review	rating
1587	587	I loved it, it was really scary.	1
739	739	I great reception all the time.	1
1196	196	(My mother and brother had to do this)When I s...	1
2471	723	Special thanks to Dylan T. for the recommendat...	1
102	102	Definitely a bargain.	1
1049	49	The acting helps the writing along very well (...	1
1285	285	I have seen many movies starring Jaclyn Smith,...	1
2320	572	Waited and waited and waited.	0
2692	944	The cashew cream sauce was bland and the veget...	0
882	882	The only good thing was that it fits comfortab...	1

In [3]:
```
#Visually inspect the dataframe
reviews_df.shape
```

Out[3]: (2748, 3)

In [4]:
```
reviews_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2748 entries, 0 to 2747
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   index   2748 non-null   int64 
 1   review  2748 non-null   object
 2   rating  2748 non-null   int64 
dtypes: int64(2), object(1)
memory usage: 64.5+ KB
```

In [5]:
```
reviews_df.describe()
```

Out[5]:

	index	rating
count	2748.000000	2748.000000
mean	465.203057	0.504367
std	276.612338	0.500072
min	0.000000	0.000000
25%	228.750000	0.000000
50%	457.500000	1.000000
75%	686.250000	1.000000
max	999.000000	1.000000

In [6]:
```
reviews_df['rating'].value_counts()
```

Out[6]:
```
1    1386
0    1362
Name: rating, dtype: int64
```

In [7]:
```
reviews_df.isna().sum()
```

Out[7]:
```
index     0
review    0
rating    0
dtype: int64
```

In [8]:
```
reviews_df.review.sample(10)
```

Out[8]:
```
1093                              Again, no plot at all.
525                                         REALLY UGLY.
485                                  A Disappointment.
1992    like the other reviewer said "you couldn't pay...
1873    Host staff were, for lack of a better word, BI...
773               The reception has been generally good.
1047      An hour and a half I wish I could bring back.
1406    To be honest with you, this is unbelievable no...
2668    Needless to say, I won't be going back anytime...
1612    Of course the footage from the 70s was grainy,...
Name: review, dtype: object
```

In [9]:
```
#Perform exploratory data analysis on the concatenated dataset
reviews = reviews_df['review']
char_list = []
for review in reviews:
    for word in word_tokenize(review.lower()):
        for char in word:
            if char not in char_list:
                char_list.append(char)

alpha = '[a-zA-Z]'
num = '[0-9]'
alpha_chars = []
num_chars = []
nonal_num_chars = []

for char in char_list:
    try:
        try:
            alpha_chars.append(re.match(alpha, char)[0])
        except:
            num_chars.append(re.match(num, char)[0])
    except:
        nonal_num_chars.append(char)

print('All alpha Characters:')
print(alpha_chars)
print('There are ',len(alpha_chars),' unique english letters in this dataset')
print(' ')

print('All numeric Characters:')
print(num_chars)
print('There are ',len(num_chars),' unique numerical characters in this dataset')
print(' ')

print('All non-alphanumeric characters:')
print(nonal_num_chars)
print('There are ',len(nonal_num_chars),' unique special characters in this dataset')
```

```
All alpha Characters:
['s', 'o', 't', 'h', 'e', 'r', 'i', 'n', 'w', 'a', 'y', 'f', 'm', 'p', 'l', 'u', 'g', 'b', 'c', 'v', 'd',
'x', 'j', 'z', 'q', 'k']
There are  26  unique english letters in this dataset

All numeric Characters:
['4', '5', '7', '3', '6', '8', '0', '2', '1', '9']
There are  10  unique numerical characters in this dataset

All non-alphanumeric characters:
['.', ',', '!', '+', '`', '"', '/', '?', '-', ':', ')', '(', '&', '$', '*', ';', '%', '#', '[', ']', '\x96',
'é', 'å', '\x97', 'ê']
There are  25  unique special characters in this dataset
```

In [10]:
```
#Divide the reviews into seperate words
#Convert words into lowercase
#Elimnate stopwords
#lemmatize the list
rev_list = []
rev_len = []
stop_words = stopwords.words('english')

for review in df.review:
    review = re.sub("[^a-zA-Z\s]", "", review)
    review = review.lower()
    review = nltk.word_tokenize(review)
    review = [word for word in review if not word in stop_words]
    lemma = nltk.WordNetLemmatizer()
    review = [lemma.lemmatize(word) for word in review]
    length = len(review)
    rev_len.append(length)
    rev_list.append(review)

n = random.randint(0, len(rev_list))
rev_list = np.asarray(rev_list, dtype=object)
print(rev_list[n])
```

```
['design', 'might', 'ergonomic', 'theory', 'could', 'stand', 'ear']
```

In [11]:
```
#Convert words into numerical values
#Sequence the tokenizer
tokenizer = Tokenizer(lower=True)
tokenizer.fit_on_texts(rev_list)
word_index = tokenizer.word_index
word_counts = list(tokenizer.word_counts.items())
word_counts.sort(key=lambda y: y[1], reverse=True)
vocab_size = len(tokenizer.word_index)+1

max_seq_emb = int(round(vocab_size ** (1/4))) #, 0))
max_len = len(max(rev_list, key=len))

sequence = tokenizer.texts_to_sequences(rev_list)
```

In [12]:
```
#Padding
padded_sequence = pad_sequences(sequence, maxlen=max_len, padding='post', truncating='post')
print('Vocabulary size: ',vocab_size)
print('max sequence embed: ', max_seq_emb)
print('max review length: ', max_len)
```

```
Vocabulary size:  4764
max sequence embed:  8
max review length:  686
```

In [13]:

```
#Provide a single padded sequence
n = random.randint(0, len(rev_list))
print('Original Review:')
print('"', df.review[n], '"')
print('____________________________________________________________')
print('')

print('Review split, lemmatized and stop words removed')
print(rev_list[n])
print('____________________________________________________________')
print('')


print('Review tokenized, sequenced and padded:')
print(padded_sequence[n])
```

```
Original Review:
" I own 2 of these cases and would order another. "
____________________________________________________________

Review split, lemmatized and stop words removed
['case', 'would', 'order', 'another']
____________________________________________________________

Review tokenized, sequenced and padded:
[ 60  17 169 142   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0]
```

In [14]:

```
#Create the model, using binary cross entropy for the loss function and sigmoid for final layer activation
keras.backend.clear_session()

model = keras.Sequential()
model.add(Embedding(vocab_size, max_seq_emb))
model.add(GlobalAveragePooling1D())
model.add(Dense(50, activation="sigmoid"))
model.add(Dense(1, activation="sigmoid"))

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

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 8)           38112

 global_average_pooling1d (  (None, 8)                 0
 GlobalAveragePooling1D)

 dense (Dense)               (None, 50)                450

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 38613 (150.83 KB)
Trainable params: 38613 (150.83 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [15]:
```
#Create the training and test splits
X_train, X_test, y_train, y_test = train_test_split(padded_sequence,
                                                    np.array(df.rating),
                                                    test_size=0.2,
                                                    random_state=42)

pd.DataFrame(X_train).to_csv('X_training_data.csv')
pd.DataFrame(X_test).to_csv('X_testing_data.csv')
pd.DataFrame(y_train).to_csv('y_training_data.csv')
pd.DataFrame(y_test).to_csv('y_testing_data.csv')
```

In [16]:
```
#Showcase the size and shape of the training and test splits
print('Size and shape of the training data set:')
print('Training data X values (reviews text) size = ',X_train.size, ' and shape = ', X_train.shape)
print('Training data Y values (review ratings) size = ',y_train.size, ' and shape = ', y_train.shape)

print('')
print('')

print('Size and shape of the training data set:')
print('Training data X values (reviews text) size = ',X_test.size, ' and shape = ', X_test.shape)
print('Training data Y values (review ratings) size = ',y_test.size, ' and shape = ', y_test.shape)
```

```
Size and shape of the training data set:
Training data X values (reviews text) size =  1507828  and shape =  (2198, 686)
Training data Y values (review ratings) size =  2198  and shape =  (2198,)


Size and shape of the training data set:
Training data X values (reviews text) size =  377300  and shape =  (550, 686)
Training data Y values (review ratings) size =  550  and shape =  (550,)
```

In [17]:
```
#Provide visualizations of the model’s training process
stop_monitor = keras.callbacks.EarlyStopping(patience=5)

history = model.fit(X_train, y_train,
                    epochs=1000,
                    validation_split=.2,
                    shuffle=True,
                    verbose=2,
                    callbacks=stop_monitor)
```

```
Epoch 1/1000
55/55 - 1s - loss: 0.6932 - accuracy: 0.5142 - val_loss: 0.6934 - val_accuracy: 0.5068 - 827ms/epoch - 15ms/
step
Epoch 2/1000
55/55 - 0s - loss: 0.6936 - accuracy: 0.5142 - val_loss: 0.6932 - val_accuracy: 0.5068 - 152ms/epoch - 3ms/s
tep
Epoch 3/1000
55/55 - 0s - loss: 0.6934 - accuracy: 0.4858 - val_loss: 0.6938 - val_accuracy: 0.5068 - 151ms/epoch - 3ms/s
tep
Epoch 4/1000
55/55 - 0s - loss: 0.6931 - accuracy: 0.5142 - val_loss: 0.6933 - val_accuracy: 0.5068 - 157ms/epoch - 3ms/s
tep
Epoch 5/1000
55/55 - 0s - loss: 0.6932 - accuracy: 0.5142 - val_loss: 0.6930 - val_accuracy: 0.5068 - 154ms/epoch - 3ms/s
tep
Epoch 6/1000
55/55 - 0s - loss: 0.6937 - accuracy: 0.5028 - val_loss: 0.6938 - val_accuracy: 0.5068 - 155ms/epoch - 3ms/s
tep
Epoch 7/1000
55/55 - 0s - loss: 0.6933 - accuracy: 0.5142 - val_loss: 0.6930 - val_accuracy: 0.5068 - 151ms/epoch - 3ms/s
tep
Epoch 8/1000
55/55 - 0s - loss: 0.6947 - accuracy: 0.4881 - val_loss: 0.6938 - val_accuracy: 0.5068 - 154ms/epoch - 3ms/s
tep
Epoch 9/1000
55/55 - 0s - loss: 0.6931 - accuracy: 0.5142 - val_loss: 0.6931 - val_accuracy: 0.5068 - 151ms/epoch - 3ms/s
tep
Epoch 10/1000
55/55 - 0s - loss: 0.6929 - accuracy: 0.5142 - val_loss: 0.6930 - val_accuracy: 0.5068 - 149ms/epoch - 3ms/s
tep
Epoch 11/1000
55/55 - 0s - loss: 0.6930 - accuracy: 0.5142 - val_loss: 0.6930 - val_accuracy: 0.5068 - 150ms/epoch - 3ms/s
tep
Epoch 12/1000
55/55 - 0s - loss: 0.6931 - accuracy: 0.5142 - val_loss: 0.6930 - val_accuracy: 0.5068 - 152ms/epoch - 3ms/s
tep
Epoch 13/1000
55/55 - 0s - loss: 0.6935 - accuracy: 0.4949 - val_loss: 0.6930 - val_accuracy: 0.5068 - 151ms/epoch - 3ms/s
tep
Epoch 14/1000
55/55 - 0s - loss: 0.6933 - accuracy: 0.5074 - val_loss: 0.6931 - val_accuracy: 0.4932 - 149ms/epoch - 3ms/s
tep
Epoch 15/1000
```

```
55/55 - 0s - loss: 0.6930 - accuracy: 0.5085 - val_loss: 0.6930 - val_accuracy: 0.5068 - 155ms/epoch - 3ms/s
tep
Epoch 16/1000
55/55 - 0s - loss: 0.6930 - accuracy: 0.5102 - val_loss: 0.6941 - val_accuracy: 0.5068 - 149ms/epoch - 3ms/s
tep
Epoch 17/1000
55/55 - 0s - loss: 0.6931 - accuracy: 0.5159 - val_loss: 0.6929 - val_accuracy: 0.6068 - 149ms/epoch - 3ms/s
tep
Epoch 18/1000
55/55 - 0s - loss: 0.6932 - accuracy: 0.5091 - val_loss: 0.6929 - val_accuracy: 0.4932 - 148ms/epoch - 3ms/s
tep
Epoch 19/1000
55/55 - 0s - loss: 0.6922 - accuracy: 0.5319 - val_loss: 0.6935 - val_accuracy: 0.5068 - 151ms/epoch - 3ms/s
tep
Epoch 20/1000
55/55 - 0s - loss: 0.6931 - accuracy: 0.5148 - val_loss: 0.6926 - val_accuracy: 0.5068 - 147ms/epoch - 3ms/s
tep
Epoch 21/1000
55/55 - 0s - loss: 0.6923 - accuracy: 0.5148 - val_loss: 0.6924 - val_accuracy: 0.5068 - 148ms/epoch - 3ms/s
tep
Epoch 22/1000
55/55 - 0s - loss: 0.6919 - accuracy: 0.5154 - val_loss: 0.6923 - val_accuracy: 0.5068 - 149ms/epoch - 3ms/s
tep
Epoch 23/1000
55/55 - 0s - loss: 0.6921 - accuracy: 0.5154 - val_loss: 0.6921 - val_accuracy: 0.5068 - 144ms/epoch - 3ms/s
tep
Epoch 24/1000
55/55 - 0s - loss: 0.6915 - accuracy: 0.5159 - val_loss: 0.6920 - val_accuracy: 0.5068 - 142ms/epoch - 3ms/s
tep
Epoch 25/1000
55/55 - 0s - loss: 0.6915 - accuracy: 0.5262 - val_loss: 0.6922 - val_accuracy: 0.5068 - 142ms/epoch - 3ms/s
tep
Epoch 26/1000
55/55 - 0s - loss: 0.6907 - accuracy: 0.5154 - val_loss: 0.6914 - val_accuracy: 0.5295 - 146ms/epoch - 3ms/s
tep
Epoch 27/1000
55/55 - 0s - loss: 0.6904 - accuracy: 0.5193 - val_loss: 0.6913 - val_accuracy: 0.5068 - 142ms/epoch - 3ms/s
tep
Epoch 28/1000
55/55 - 0s - loss: 0.6894 - accuracy: 0.5404 - val_loss: 0.6911 - val_accuracy: 0.4955 - 145ms/epoch - 3ms/s
tep
Epoch 29/1000
55/55 - 0s - loss: 0.6878 - accuracy: 0.5324 - val_loss: 0.6941 - val_accuracy: 0.5068 - 146ms/epoch - 3ms/s
```

```
tep
Epoch 30/1000
55/55 - 0s - loss: 0.6895 - accuracy: 0.5284 - val_loss: 0.6925 - val_accuracy: 0.5068 - 162ms/epoch - 3ms/s
tep
Epoch 31/1000
55/55 - 0s - loss: 0.6889 - accuracy: 0.5301 - val_loss: 0.6910 - val_accuracy: 0.5068 - 154ms/epoch - 3ms/s
tep
Epoch 32/1000
55/55 - 0s - loss: 0.6873 - accuracy: 0.5944 - val_loss: 0.6890 - val_accuracy: 0.5159 - 159ms/epoch - 3ms/s
tep
Epoch 33/1000
55/55 - 0s - loss: 0.6857 - accuracy: 0.5825 - val_loss: 0.6894 - val_accuracy: 0.5068 - 161ms/epoch - 3ms/s
tep
Epoch 34/1000
55/55 - 0s - loss: 0.6841 - accuracy: 0.6615 - val_loss: 0.6897 - val_accuracy: 0.5068 - 150ms/epoch - 3ms/s
tep
Epoch 35/1000
55/55 - 0s - loss: 0.6832 - accuracy: 0.5341 - val_loss: 0.6871 - val_accuracy: 0.5159 - 146ms/epoch - 3ms/s
tep
Epoch 36/1000
55/55 - 0s - loss: 0.6817 - accuracy: 0.6189 - val_loss: 0.6864 - val_accuracy: 0.5114 - 163ms/epoch - 3ms/s
tep
Epoch 37/1000
55/55 - 0s - loss: 0.6805 - accuracy: 0.6621 - val_loss: 0.6854 - val_accuracy: 0.5182 - 158ms/epoch - 3ms/s
tep
Epoch 38/1000
55/55 - 0s - loss: 0.6786 - accuracy: 0.5336 - val_loss: 0.6871 - val_accuracy: 0.4932 - 143ms/epoch - 3ms/s
tep
Epoch 39/1000
55/55 - 0s - loss: 0.6767 - accuracy: 0.5489 - val_loss: 0.6842 - val_accuracy: 0.5091 - 149ms/epoch - 3ms/s
tep
Epoch 40/1000
55/55 - 0s - loss: 0.6758 - accuracy: 0.6337 - val_loss: 0.6838 - val_accuracy: 0.5091 - 145ms/epoch - 3ms/s
tep
Epoch 41/1000
55/55 - 0s - loss: 0.6740 - accuracy: 0.6018 - val_loss: 0.6827 - val_accuracy: 0.5091 - 144ms/epoch - 3ms/s
tep
Epoch 42/1000
55/55 - 0s - loss: 0.6714 - accuracy: 0.6445 - val_loss: 0.6799 - val_accuracy: 0.7386 - 143ms/epoch - 3ms/s
tep
Epoch 43/1000
55/55 - 0s - loss: 0.6672 - accuracy: 0.6786 - val_loss: 0.6791 - val_accuracy: 0.5159 - 146ms/epoch - 3ms/s
tep
```

```
Epoch 44/1000
55/55 - 0s - loss: 0.6644 - accuracy: 0.7582 - val_loss: 0.6767 - val_accuracy: 0.5705 - 143ms/epoch - 3ms/s
tep
Epoch 45/1000
55/55 - 0s - loss: 0.6602 - accuracy: 0.8089 - val_loss: 0.6775 - val_accuracy: 0.5091 - 144ms/epoch - 3ms/s
tep
Epoch 46/1000
55/55 - 0s - loss: 0.6584 - accuracy: 0.6997 - val_loss: 0.6727 - val_accuracy: 0.7455 - 150ms/epoch - 3ms/s
tep
Epoch 47/1000
55/55 - 0s - loss: 0.6550 - accuracy: 0.6894 - val_loss: 0.6742 - val_accuracy: 0.5136 - 150ms/epoch - 3ms/s
tep
Epoch 48/1000
55/55 - 0s - loss: 0.6504 - accuracy: 0.6962 - val_loss: 0.6684 - val_accuracy: 0.7773 - 180ms/epoch - 3ms/s
tep
Epoch 49/1000
55/55 - 0s - loss: 0.6461 - accuracy: 0.7144 - val_loss: 0.6660 - val_accuracy: 0.7727 - 177ms/epoch - 3ms/s
tep
Epoch 50/1000
55/55 - 0s - loss: 0.6401 - accuracy: 0.7969 - val_loss: 0.6631 - val_accuracy: 0.7455 - 143ms/epoch - 3ms/s
tep
Epoch 51/1000
55/55 - 0s - loss: 0.6364 - accuracy: 0.7952 - val_loss: 0.6603 - val_accuracy: 0.7727 - 144ms/epoch - 3ms/s
tep
Epoch 52/1000
55/55 - 0s - loss: 0.6303 - accuracy: 0.7526 - val_loss: 0.6588 - val_accuracy: 0.6795 - 148ms/epoch - 3ms/s
tep
Epoch 53/1000
55/55 - 0s - loss: 0.6243 - accuracy: 0.8123 - val_loss: 0.6536 - val_accuracy: 0.7636 - 143ms/epoch - 3ms/s
tep
Epoch 54/1000
55/55 - 0s - loss: 0.6187 - accuracy: 0.7850 - val_loss: 0.6504 - val_accuracy: 0.7159 - 146ms/epoch - 3ms/s
tep
Epoch 55/1000
55/55 - 0s - loss: 0.6102 - accuracy: 0.8396 - val_loss: 0.6463 - val_accuracy: 0.7341 - 146ms/epoch - 3ms/s
tep
Epoch 56/1000
55/55 - 0s - loss: 0.6027 - accuracy: 0.8845 - val_loss: 0.6428 - val_accuracy: 0.7182 - 144ms/epoch - 3ms/s
tep
Epoch 57/1000
55/55 - 0s - loss: 0.5959 - accuracy: 0.8220 - val_loss: 0.6422 - val_accuracy: 0.6136 - 142ms/epoch - 3ms/s
tep
Epoch 58/1000
```

```
55/55 - 0s - loss: 0.5865 - accuracy: 0.8413 - val_loss: 0.6351 - val_accuracy: 0.7455 - 145ms/epoch - 3ms/s
tep
Epoch 59/1000
55/55 - 0s - loss: 0.5789 - accuracy: 0.8606 - val_loss: 0.6292 - val_accuracy: 0.7727 - 158ms/epoch - 3ms/s
tep
Epoch 60/1000
55/55 - 0s - loss: 0.5699 - accuracy: 0.8413 - val_loss: 0.6315 - val_accuracy: 0.6136 - 152ms/epoch - 3ms/s
tep
Epoch 61/1000
55/55 - 0s - loss: 0.5578 - accuracy: 0.8697 - val_loss: 0.6306 - val_accuracy: 0.5977 - 145ms/epoch - 3ms/s
tep
Epoch 62/1000
55/55 - 0s - loss: 0.5520 - accuracy: 0.8345 - val_loss: 0.6134 - val_accuracy: 0.7795 - 147ms/epoch - 3ms/s
tep
Epoch 63/1000
55/55 - 0s - loss: 0.5378 - accuracy: 0.9050 - val_loss: 0.6079 - val_accuracy: 0.7523 - 142ms/epoch - 3ms/s
tep
Epoch 64/1000
55/55 - 0s - loss: 0.5280 - accuracy: 0.8862 - val_loss: 0.6029 - val_accuracy: 0.7773 - 143ms/epoch - 3ms/s
tep
Epoch 65/1000
55/55 - 0s - loss: 0.5191 - accuracy: 0.8919 - val_loss: 0.6008 - val_accuracy: 0.7227 - 142ms/epoch - 3ms/s
tep
Epoch 66/1000
```

```
55/55 - 0s - loss: 0.5059 - accuracy: 0.8953 - val_loss: 0.5923 - val_accuracy: 0.7682 - 156ms/epoch - 3ms/s
tep
Epoch 67/1000
55/55 - 0s - loss: 0.4963 - accuracy: 0.9067 - val_loss: 0.5854 - val_accuracy: 0.7659 - 159ms/epoch - 3ms/s
tep
Epoch 68/1000
55/55 - 0s - loss: 0.4838 - accuracy: 0.9078 - val_loss: 0.5795 - val_accuracy: 0.7795 - 148ms/epoch - 3ms/s
tep
Epoch 69/1000
55/55 - 0s - loss: 0.4727 - accuracy: 0.9158 - val_loss: 0.5738 - val_accuracy: 0.7705 - 148ms/epoch - 3ms/s
tep
Epoch 70/1000
55/55 - 0s - loss: 0.4608 - accuracy: 0.9226 - val_loss: 0.5727 - val_accuracy: 0.7523 - 158ms/epoch - 3ms/s
tep
Epoch 71/1000
55/55 - 0s - loss: 0.4507 - accuracy: 0.9130 - val_loss: 0.5641 - val_accuracy: 0.7795 - 149ms/epoch - 3ms/s
tep
Epoch 72/1000
55/55 - 0s - loss: 0.4392 - accuracy: 0.9164 - val_loss: 0.5582 - val_accuracy: 0.7841 - 155ms/epoch - 3ms/s
tep
Epoch 73/1000
55/55 - 0s - loss: 0.4294 - accuracy: 0.9170 - val_loss: 0.5526 - val_accuracy: 0.7705 - 154ms/epoch - 3ms/s
tep
Epoch 74/1000
55/55 - 0s - loss: 0.4168 - accuracy: 0.9243 - val_loss: 0.5474 - val_accuracy: 0.7795 - 162ms/epoch - 3ms/s
tep
Epoch 75/1000
55/55 - 0s - loss: 0.4082 - accuracy: 0.9170 - val_loss: 0.5444 - val_accuracy: 0.7818 - 163ms/epoch - 3ms/s
tep
Epoch 76/1000
55/55 - 0s - loss: 0.3987 - accuracy: 0.9101 - val_loss: 0.5431 - val_accuracy: 0.7682 - 153ms/epoch - 3ms/s
tep
Epoch 77/1000
55/55 - 0s - loss: 0.3895 - accuracy: 0.9209 - val_loss: 0.5345 - val_accuracy: 0.7750 - 161ms/epoch - 3ms/s
tep
Epoch 78/1000
55/55 - 0s - loss: 0.3771 - accuracy: 0.9215 - val_loss: 0.5312 - val_accuracy: 0.7614 - 147ms/epoch - 3ms/s
tep
Epoch 79/1000
55/55 - 0s - loss: 0.3703 - accuracy: 0.9158 - val_loss: 0.5285 - val_accuracy: 0.7591 - 143ms/epoch - 3ms/s
tep
Epoch 80/1000
55/55 - 0s - loss: 0.3649 - accuracy: 0.9147 - val_loss: 0.5375 - val_accuracy: 0.7523 - 145ms/epoch - 3ms/s
```

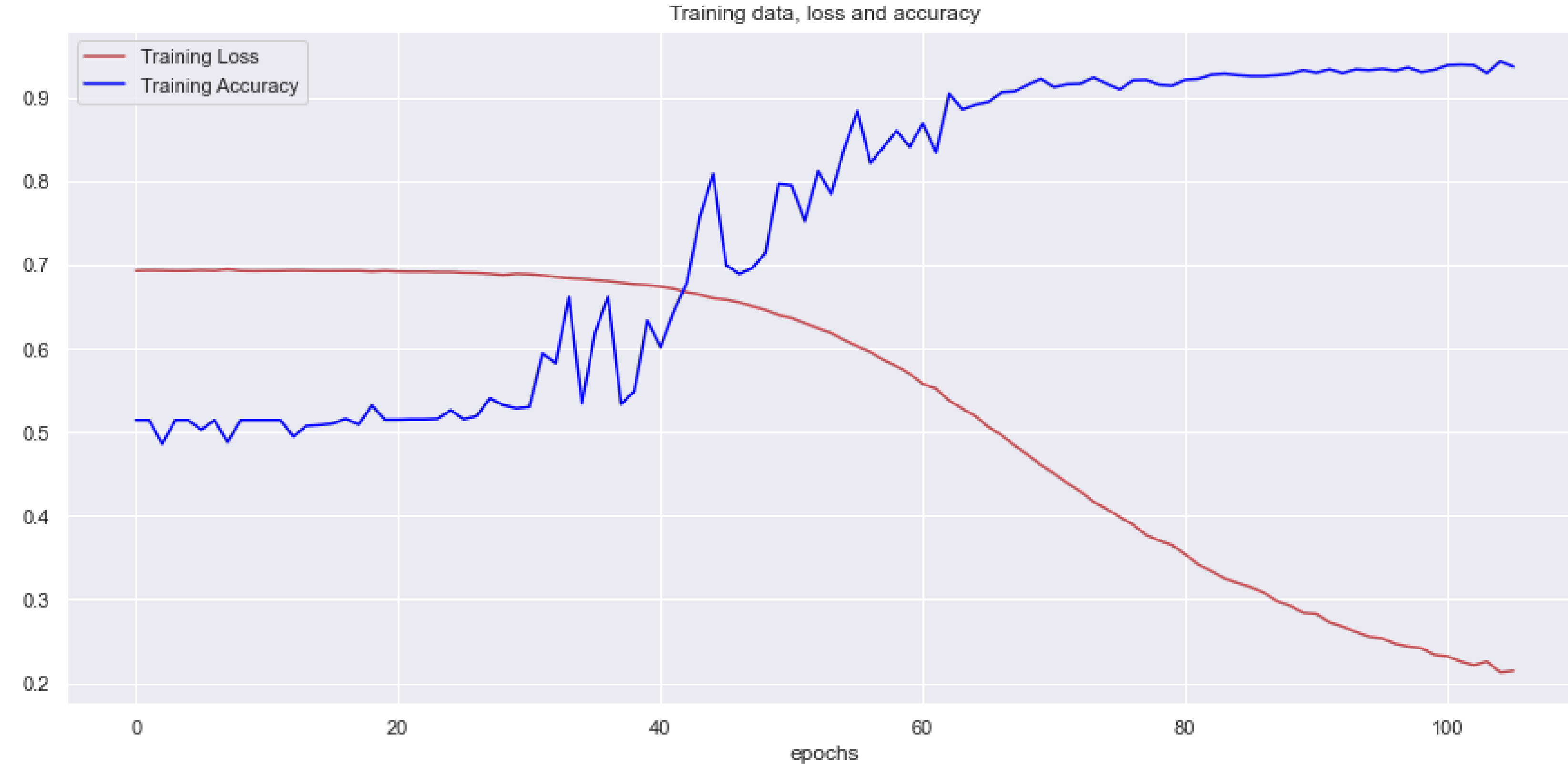
```
tep
Epoch 81/1000
55/55 - 0s - loss: 0.3540 - accuracy: 0.9215 - val_loss: 0.5199 - val_accuracy: 0.7773 - 153ms/epoch - 3ms/s
tep
Epoch 82/1000
55/55 - 0s - loss: 0.3417 - accuracy: 0.9226 - val_loss: 0.5158 - val_accuracy: 0.7795 - 155ms/epoch - 3ms/s
tep
Epoch 83/1000
55/55 - 0s - loss: 0.3337 - accuracy: 0.9278 - val_loss: 0.5129 - val_accuracy: 0.7818 - 144ms/epoch - 3ms/s
tep
Epoch 84/1000
55/55 - 0s - loss: 0.3249 - accuracy: 0.9289 - val_loss: 0.5100 - val_accuracy: 0.7795 - 160ms/epoch - 3ms/s
tep
Epoch 85/1000
55/55 - 0s - loss: 0.3195 - accuracy: 0.9272 - val_loss: 0.5081 - val_accuracy: 0.7773 - 148ms/epoch - 3ms/s
tep
Epoch 86/1000
55/55 - 0s - loss: 0.3144 - accuracy: 0.9261 - val_loss: 0.5303 - val_accuracy: 0.7250 - 143ms/epoch - 3ms/s
tep
Epoch 87/1000
55/55 - 0s - loss: 0.3079 - accuracy: 0.9261 - val_loss: 0.5086 - val_accuracy: 0.7841 - 148ms/epoch - 3ms/s
tep
Epoch 88/1000
55/55 - 0s - loss: 0.2977 - accuracy: 0.9272 - val_loss: 0.5104 - val_accuracy: 0.7727 - 145ms/epoch - 3ms/s
tep
Epoch 89/1000
55/55 - 0s - loss: 0.2926 - accuracy: 0.9289 - val_loss: 0.5002 - val_accuracy: 0.7841 - 146ms/epoch - 3ms/s
tep
Epoch 90/1000
55/55 - 0s - loss: 0.2841 - accuracy: 0.9329 - val_loss: 0.5002 - val_accuracy: 0.7841 - 157ms/epoch - 3ms/s
tep
Epoch 91/1000
55/55 - 0s - loss: 0.2828 - accuracy: 0.9300 - val_loss: 0.4979 - val_accuracy: 0.7795 - 152ms/epoch - 3ms/s
tep
Epoch 92/1000
55/55 - 0s - loss: 0.2726 - accuracy: 0.9340 - val_loss: 0.4968 - val_accuracy: 0.7773 - 144ms/epoch - 3ms/s
tep
Epoch 93/1000
55/55 - 0s - loss: 0.2676 - accuracy: 0.9295 - val_loss: 0.4955 - val_accuracy: 0.7818 - 143ms/epoch - 3ms/s
tep
Epoch 94/1000
55/55 - 0s - loss: 0.2612 - accuracy: 0.9340 - val_loss: 0.4940 - val_accuracy: 0.7886 - 143ms/epoch - 3ms/s
tep
```

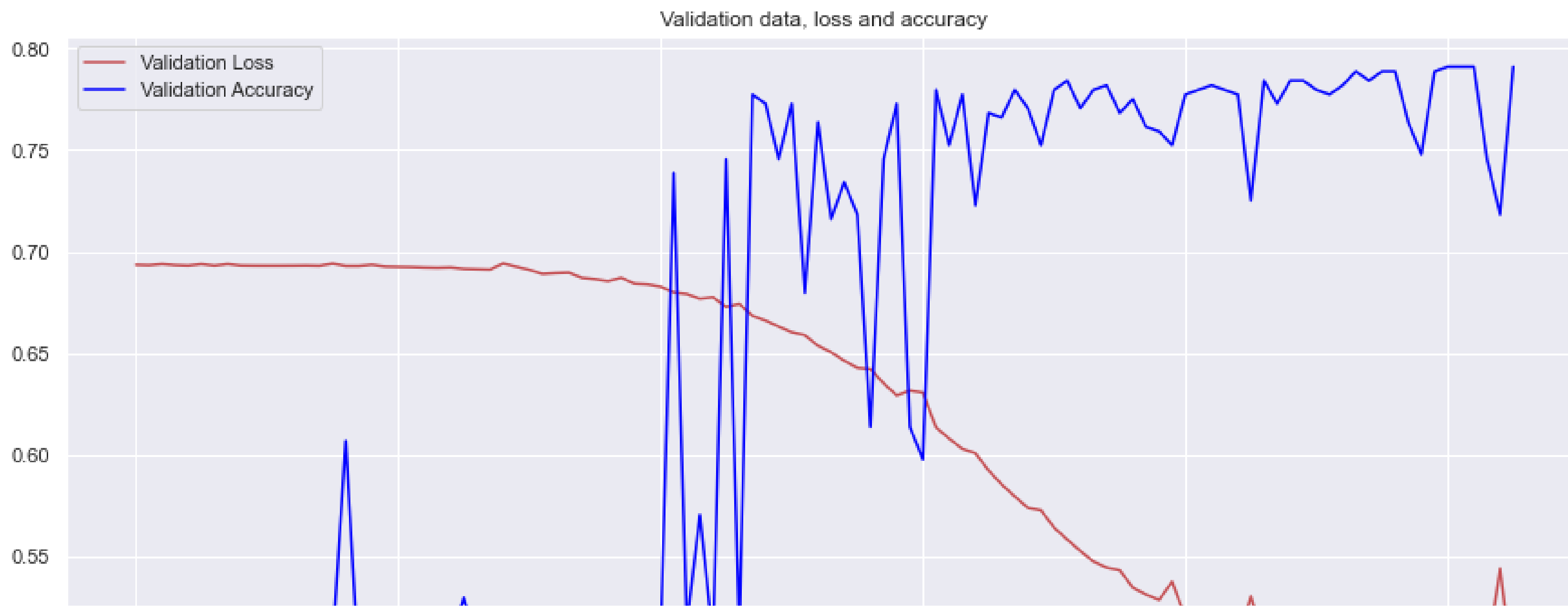
```
Epoch 95/1000
55/55 - 0s - loss: 0.2556 - accuracy: 0.9329 - val_loss: 0.4939 - val_accuracy: 0.7841 - 144ms/epoch - 3ms/s
tep
Epoch 96/1000
55/55 - 0s - loss: 0.2534 - accuracy: 0.9346 - val_loss: 0.5004 - val_accuracy: 0.7886 - 143ms/epoch - 3ms/s
tep
Epoch 97/1000
55/55 - 0s - loss: 0.2470 - accuracy: 0.9323 - val_loss: 0.4915 - val_accuracy: 0.7886 - 142ms/epoch - 3ms/s
tep
Epoch 98/1000
55/55 - 0s - loss: 0.2437 - accuracy: 0.9363 - val_loss: 0.4955 - val_accuracy: 0.7636 - 145ms/epoch - 3ms/s
tep
Epoch 99/1000
55/55 - 0s - loss: 0.2416 - accuracy: 0.9306 - val_loss: 0.5059 - val_accuracy: 0.7477 - 144ms/epoch - 3ms/s
tep
Epoch 100/1000
55/55 - 0s - loss: 0.2339 - accuracy: 0.9334 - val_loss: 0.4932 - val_accuracy: 0.7886 - 146ms/epoch - 3ms/s
tep
Epoch 101/1000
55/55 - 0s - loss: 0.2319 - accuracy: 0.9391 - val_loss: 0.4906 - val_accuracy: 0.7909 - 142ms/epoch - 3ms/s
tep
Epoch 102/1000
55/55 - 0s - loss: 0.2257 - accuracy: 0.9397 - val_loss: 0.4959 - val_accuracy: 0.7909 - 144ms/epoch - 3ms/s
tep
Epoch 103/1000
55/55 - 0s - loss: 0.2214 - accuracy: 0.9391 - val_loss: 0.4927 - val_accuracy: 0.7909 - 143ms/epoch - 3ms/s
tep
Epoch 104/1000
55/55 - 0s - loss: 0.2258 - accuracy: 0.9295 - val_loss: 0.5007 - val_accuracy: 0.7455 - 148ms/epoch - 3ms/s
tep
Epoch 105/1000
55/55 - 0s - loss: 0.2128 - accuracy: 0.9437 - val_loss: 0.5441 - val_accuracy: 0.7182 - 159ms/epoch - 3ms/s
tep
Epoch 106/1000
55/55 - 0s - loss: 0.2146 - accuracy: 0.9374 - val_loss: 0.4913 - val_accuracy: 0.7909 - 145ms/epoch - 3ms/s
tep
```

In [18]:
```
#Print the score of the Final Model Loss and Final Model Accuracy
score = model.evaluate(X_test, y_test, verbose=1)

print('Final Model Loss: ', round(score[0],5))
print('Final Model Accuracy: ', round(score[1]*100, 2),'%')
```

```
18/18 [==============================] - 0s 1ms/step - loss: 0.4531 - accuracy: 0.7909
Final Model Loss:  0.45313
Final Model Accuracy:  79.09 %
```

In [24]:
```
#Plot the Training and Validation data loss and accuracy
plt.figure(figsize=(15,15))

plt.subplot(2,1,1)
plt.plot(history.history['loss'], label='Training Loss', c='r')
plt.plot(history.history['accuracy'], label='Training Accuracy', c='blue')
plt.xlabel('epochs')
plt.legend()
plt.title('Training data, loss and accuracy')

plt.subplot(2,1,2)
plt.plot(history.history['val_loss'], label='Validation Loss', c='r')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', c='blue')
plt.xlabel('epochs')
plt.legend()
plt.title('Validation data, loss and accuracy')

plt.show;
```

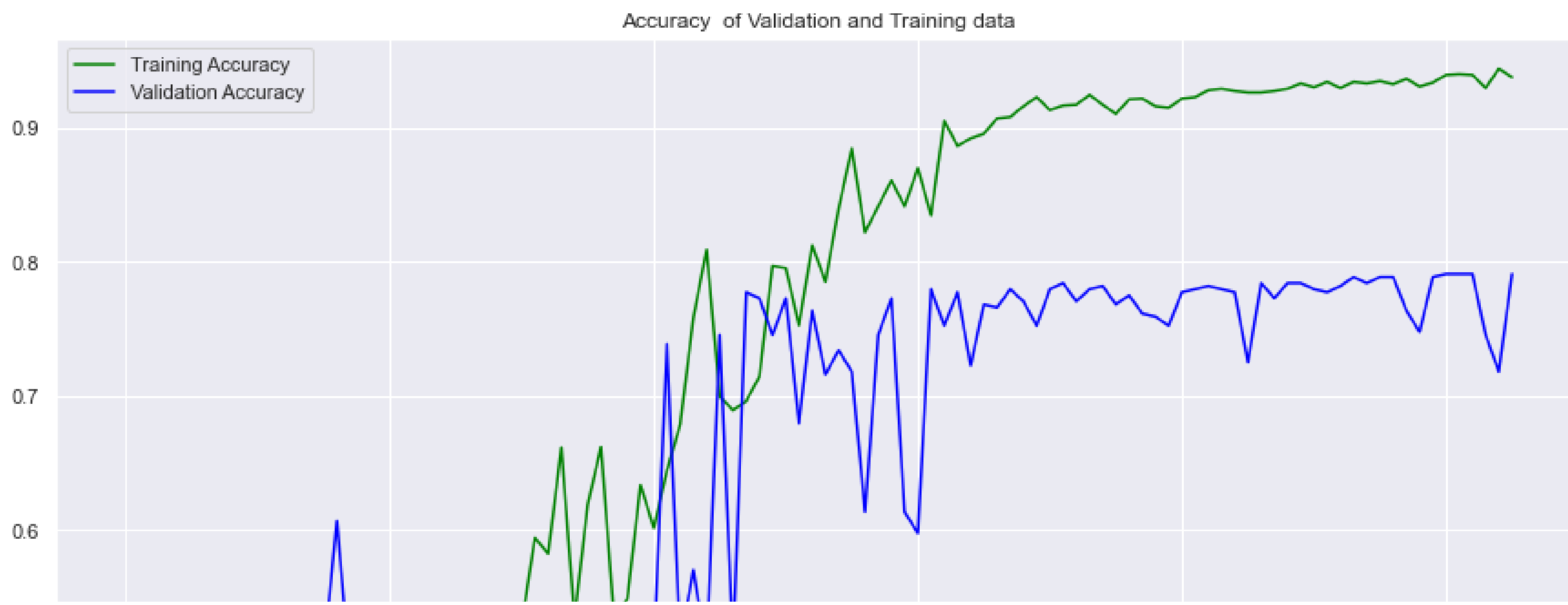

Training data, loss and accuracy
Training Loss
Training Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100
epochs


Validation data, loss and accuracy
Validation Loss
Validation Accuracy
0.80
0.75
0.70
0.65
0.60
0.55


0.50
0
20
40
60
80
100
epochs

In [22]:
```
#Plot the Loss and Accuracy of Valication and Training data
plt.figure(figsize=(15,15))

plt.subplot(2,1,1)
plt.plot(history.history['loss'], label='Training Loss', c='green')
plt.plot(history.history['val_loss'], label='Validation Loss', c='blue')

plt.xlabel('epochs')
plt.legend()
plt.title('Loss of Validation and Training data')

plt.subplot(2,1,2)
plt.plot(history.history['accuracy'], label='Training Accuracy', c='green')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', c='blue')
plt.xlabel('epochs')
plt.legend()
plt.title('Accuracy  of Validation and Training data')

plt.show();
```

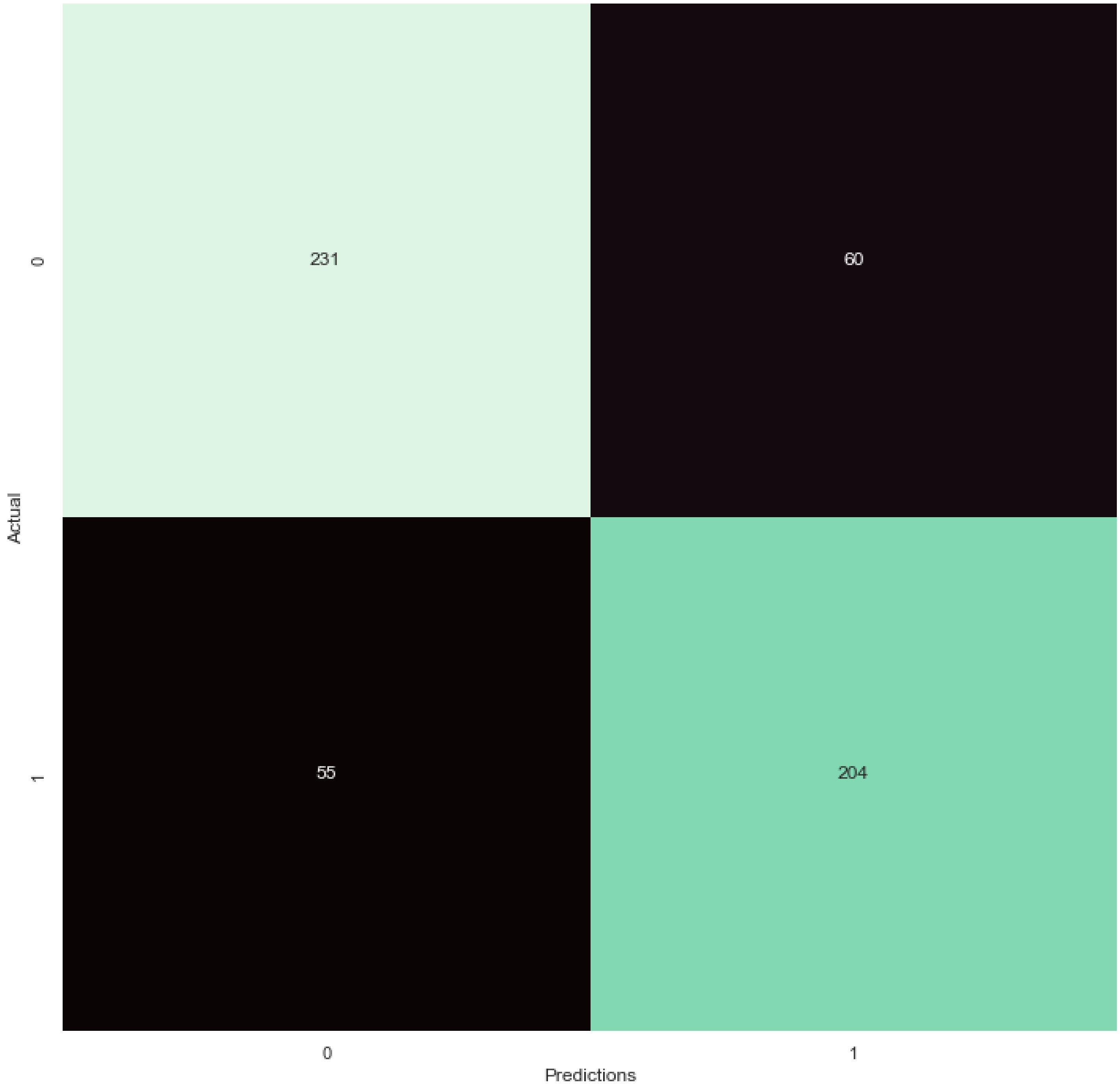

Loss of Validation and Training data
Training Loss
Validation Loss
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100
epochs


Accuracy of Validation and Training data
Training Accuracy
Validation Accuracy
0.9
0.8
0.7
0.6


0.5
0
20
40
60
80
100
epochs

In [21]:
```
#Generate a confusion matrix to see how correctly the model can predict positive and negative reviews
predictions = model.predict(X_test)
predictions = np.round(predictions,0).astype(int)

con_mat = confusion_matrix(y_test, predictions)
sns.set(rc={'figure.figsize':(12,12)})
sns.heatmap(
    con_mat, annot=True,
    fmt='d', cbar=False,
    cmap='mako').set(
    ylabel='Actual',
    xlabel='Predictions');
```

```
18/18 [==============================] - 0s 1ms/step
```


231
60
55
204
0
1
Actual
0
1
Predictions

In [26]:
```
#Save the trained network within the neural network
model.save('D213_Task2_Sentiment_Analys.keras')
```

In [ ]:
```
```