AMAZON REVIEW SENTIMENT ANALYSIS

TEAM MEMBER

- nada hamdi
- sama mohamed
- jamila ahmed
- mai abdallah
- alyaa mamoon
- manar khaled
- nada khalid
- rehab bakhet

INTRODUCTION

Amazon review sentiment analysis is like having a robot that reads all the things people say about stuff they buy on Amazon, then figures out if they liked it, or didn't like it.

It's helpful because it saves time by not having to read every single review yourself, and it gives you a quick idea of what most people think about a product.

train : 3600000

test : 400001

STEPS

First step

Data analysis

Second step

Data preprocessing

Third step

Model Selection

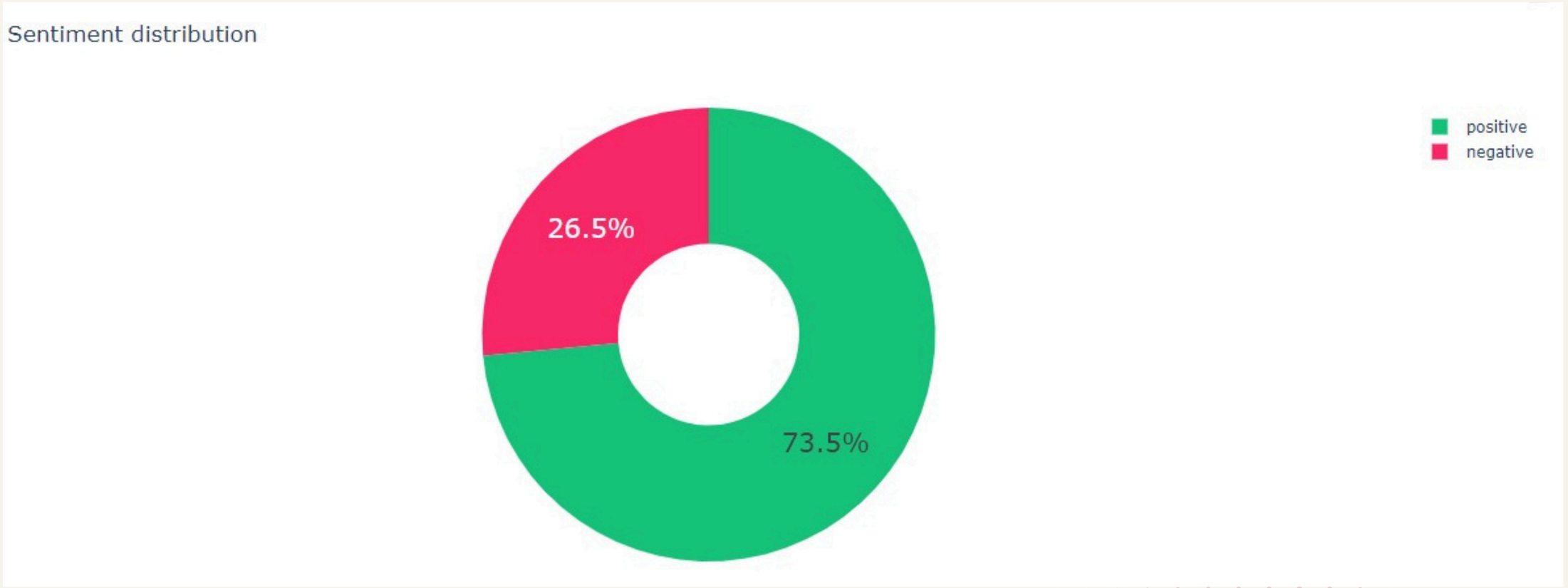
Fourth step

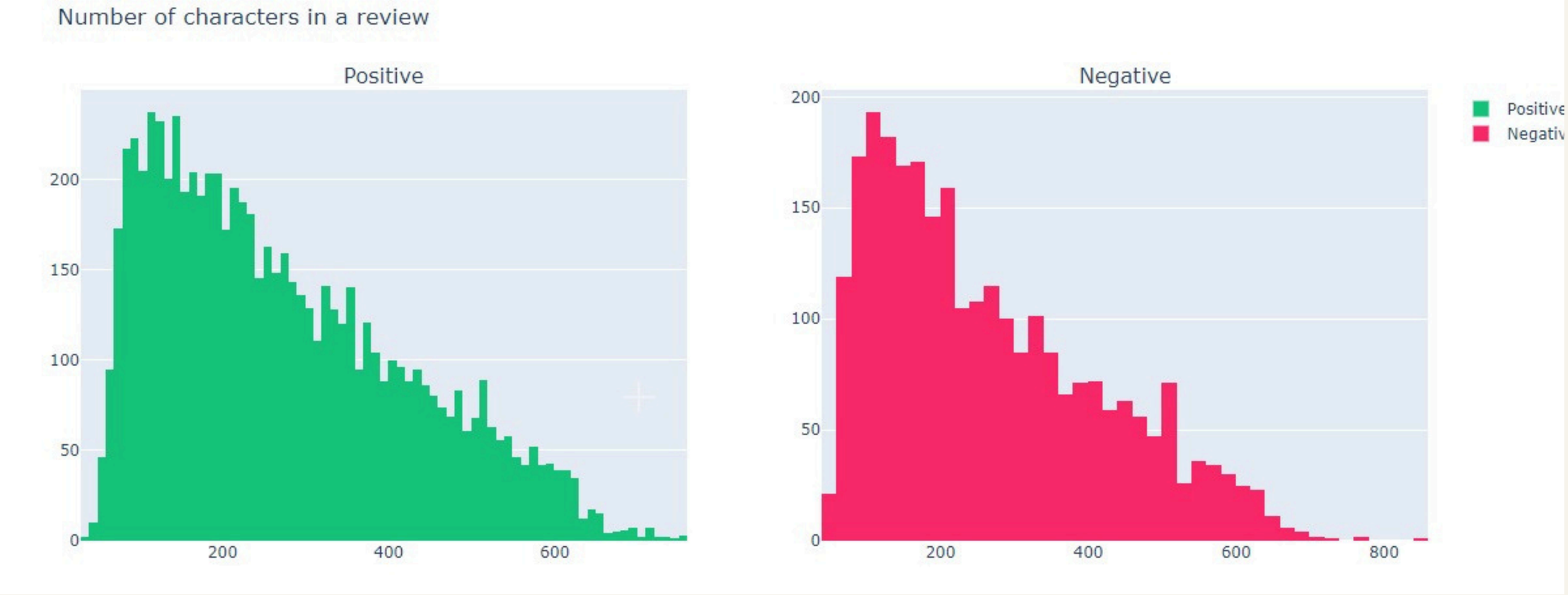
API

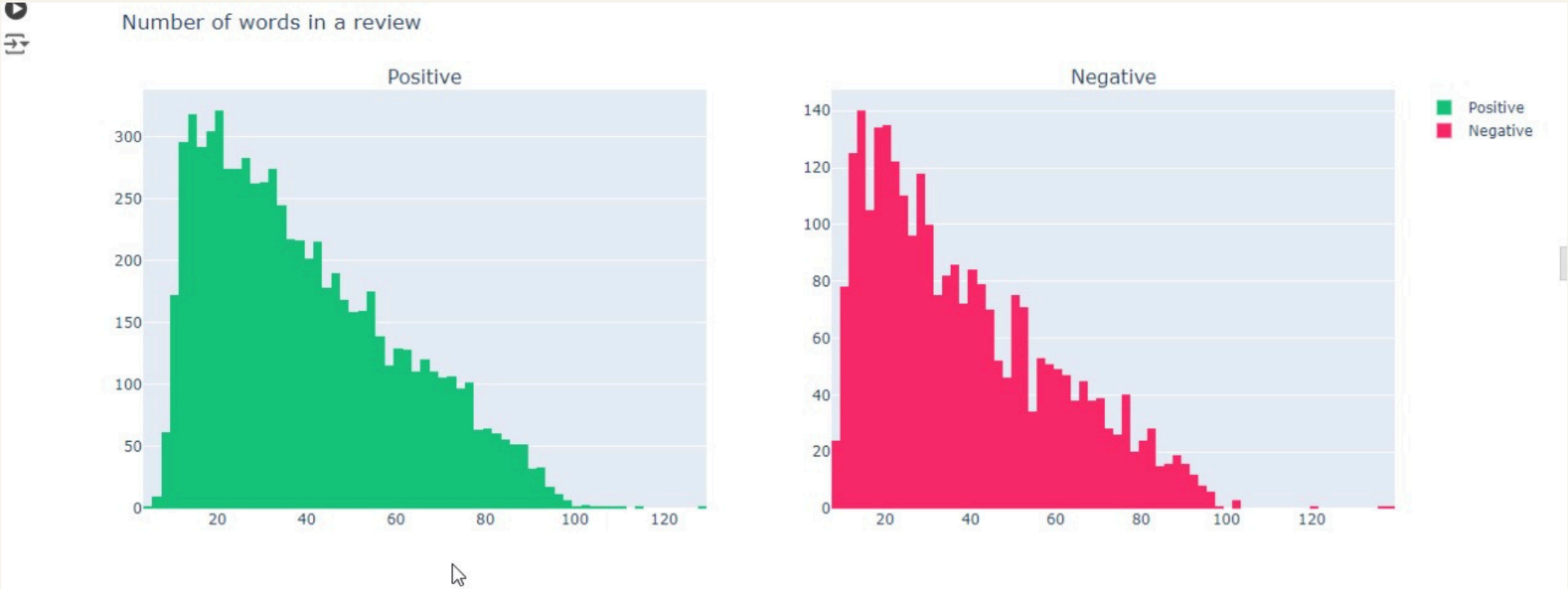
DATA ANALYSIS

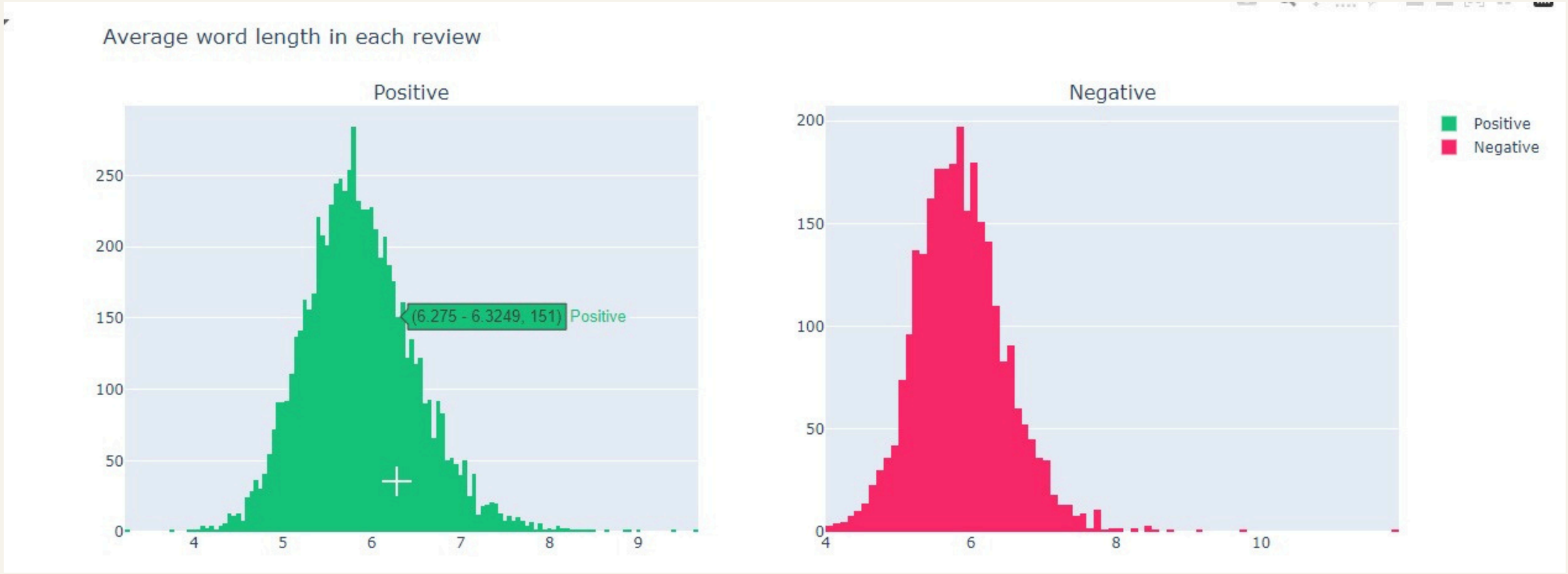
1. Finding the dataset from kaggle and import it
2. Read and convert dataset to data frame
3. Visualize the data.

VISUALIZATION



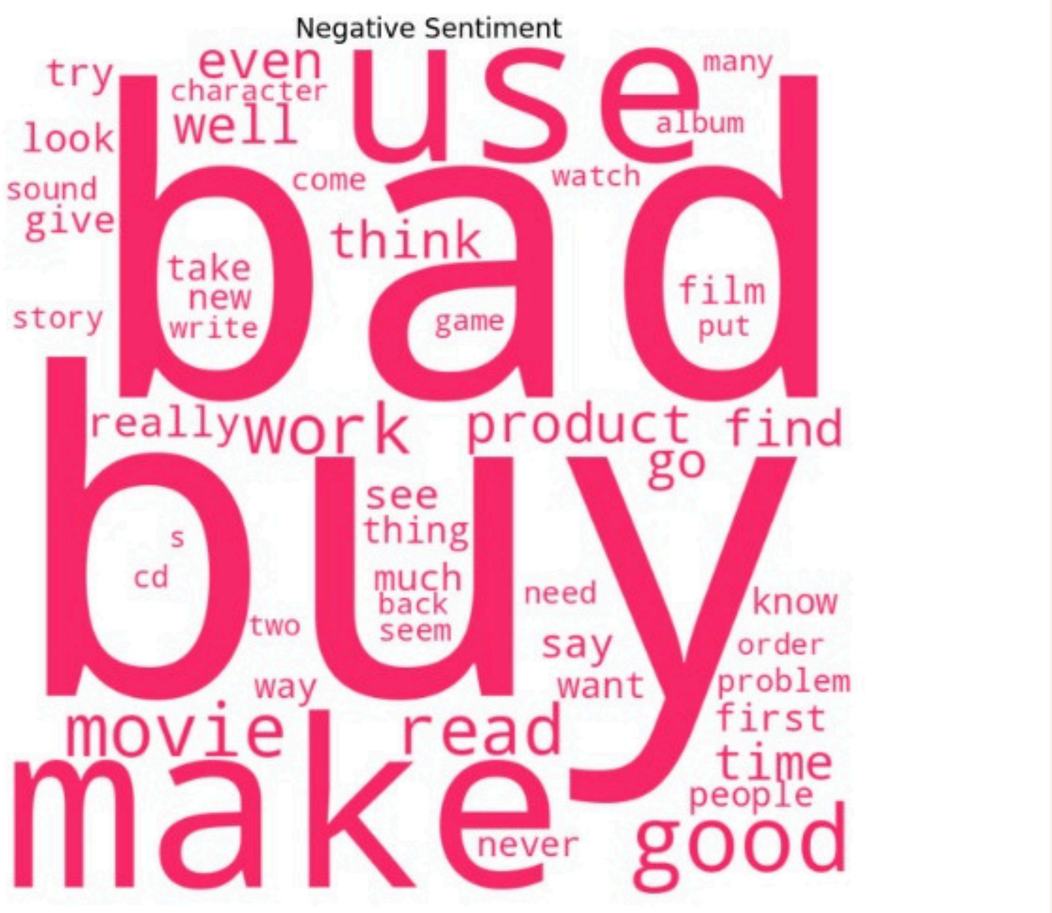

Number of characters in a review
Positive
Negative
Positive
Negativ
0
50
100
150
200
200
400
600
800


Number of words in a review
Positive
Negative
300
250
200
150
100
50
0
20
40
60
80
100
120
140
120
100
80
60
40
20
0
20
40
60
80
100
120
Positive
Negative


Average word length in each review
Positive
Negative
(6.275 - 6.3249, 151) Positive
Positive
Negative
0
50
100
150
200
250
4
5
6
7
8
9
10


Most common words by sentiment
Positive
4000
3000
2000
1000
0
not
good
great
get
do
well
would
use
buy
read
make
work
love
time
movie
Positive
Negative
Negative
1500
1000
500
0
not
do
get
buy
bad
would
make
time
movie
use
read
work
good
well
even


Positive Sentiment
well
good
great
use
love
read
think
cd
know
product
work
buy
try
go
music
new
much
take
look
little
book
even
album
many
thing
give
keep
way
feel
see
play
movie
first
time
song
make
lot
find
purchase
say
seem
put
come
want
s
still
story
really
need


Negative Sentiment
bad
buy
make
use
try
even
many
character
well
look
album
sound
come
watch
give
think
take
new
write
story
game
film
put
really
work
product
find
go
see
thing
s
cd
much
back
seem
need
know
two
say
order
way
want
problem
movie
read
first
time
people
never
good

DATA CLEANING & PREPROCESSING

1. Converting characters to lower case.
2. Remove special character, punctuation, stop words, frequent words,and rare words.
3. Applying Spelling Correction.
4. Applying Tokenization.
5. Applying Stemming.
6. Applying Lemmatization.

	review	label	review_corrected
2208675	freaking fantastiic love little heater monster...	2	freaking fantastiic love little heater monster...
3195595	planet earth bluray stunning picture quality h...	2	planet earth bluray stunning picture quality h...
274452	do not go unless fell look phone past month tm...	1	go unless fall look phone past month tmobile f...
949069	martina mcbandit strike second time cd rerelea...	1	martina mcbandit strike second time cd rerelea...
2088168	hub reliable buy hub could easily connect disc...	1	hub reliable buy hub could easily connect disc...

DATA BALANCE

```
X_train_p, _, y_train_p, _ = train_test_split(
    X_train,                    # Features
    y_train,                    # Labels
    train_size=0.005,                   # Percentage of data to use for training
    random_state=42,                  # Random seed for reproducibility
    stratify=y_train            # Perform stratified sampling based on the labels
)
X_val_p, _, y_val_p, _ = train_test_split(
    X_val,  # Features
    y_val,                    # Labels
    train_size=0.005,                   # Percentage of data to use for training
    random_state=42,                  # Random seed for reproducibility
    stratify=y_val            # Perform stratified sampling based on the labels
)
X_test_p, _, y_test_p, _ = train_test_split(
    X_test,  # Features
    y_test,                     # Labels
    train_size=0.005,                   # Percentage of data to use for training
    random_state=42,                  # Random seed for reproducibility
    stratify=y_test             # Perform stratified sampling based on the labels
)
```

```
[ ] cnt = pd.Series(y_train_p)
    cnt.value_counts()

label
1    7200
2    7200
Name: count, dtype: int64
```

```
[ ] cnt = pd.Series(y_val_p)
    cnt.value_counts()

label
1    1800
2    1800
Name: count, dtype: int64
```

```
[ ] Start coding or generate with AI.
```

```
cnt = pd.Series(y_test_p)
cnt.value_counts()

0
1    1000
2    1000
Name: count, dtype: int64
```

DATA PREPROCESSING

```
[ ]  # Remove rows with empty reviews
     train_data = train_data.dropna(subset=['review'])

     # if the review column contains whitespace strings but not NaN values
     train_data = train_data[train_data['review'].str.strip() != '']

     # Reset index after removing rows
     train_data = train_data.reset_index(drop=True)
```

```
     # Remove rows with empty reviews
     test_data = test_data.dropna(subset=['review'])

     # if the review column contains whitespace strings but not NaN values
     test_data = test_data[test_data['review'].str.strip() != '']

     # Reset index after removing rows
     test_data = test_data.reset_index(drop=True)
```

```
[ ]  !pip install langdetect
     from langdetect import detect

     # Function to detect language of a text
     def detect_language(text):
         try:
             lang = detect(text)
             return lang == 'en'  # Return True if la
         except:
             return False  # Return False if language
```

MODEL SELECTION

MODEL ONE

Naive Bayes

MODEL TWO

CNN

MODEL THREE

RNN

MODEL FOUR

Roberta

NAIVE BAYES

```
[ ]  from sklearn.feature_extraction.text import CountVectorizer
     # Conversion of text to vector

     v = CountVectorizer(stop_words='english')
     X_train = v.fit_transform(X_train_p)
     X_test = v.transform(X_test_p)
     X_val = v.transform(X_val_p)
```

```
[ ]  from sklearn.naive_bayes import MultinomialNB
     multNB = MultinomialNB()
     multNB.fit(X_train,y_train_p)
```

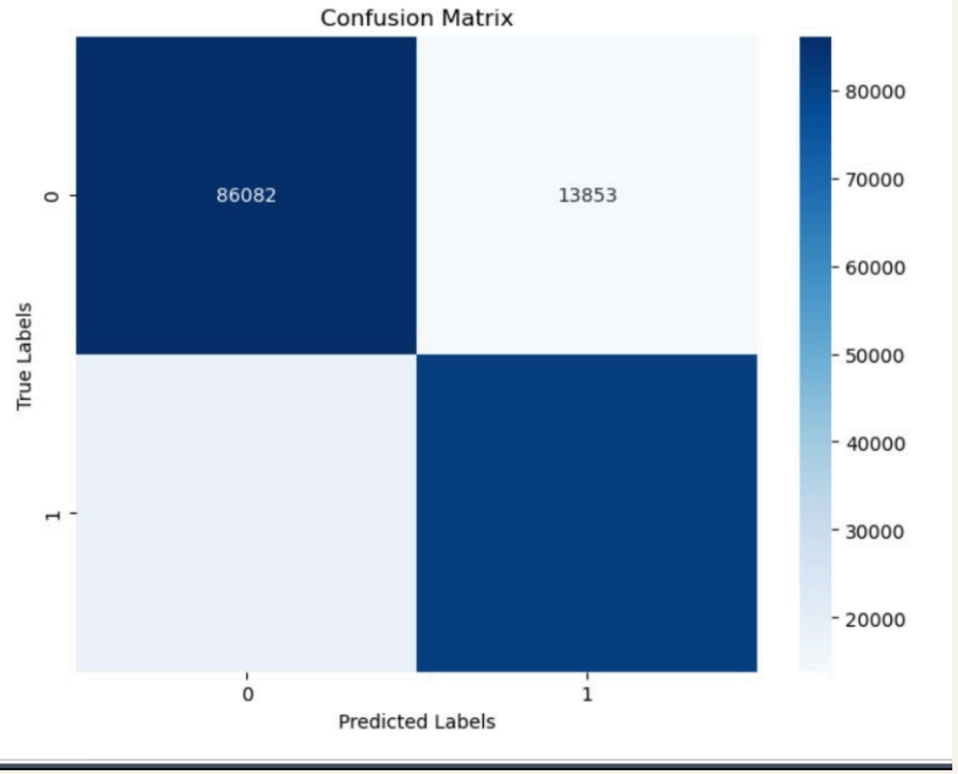
```
MultinomialNB
MultinomialNB()
```

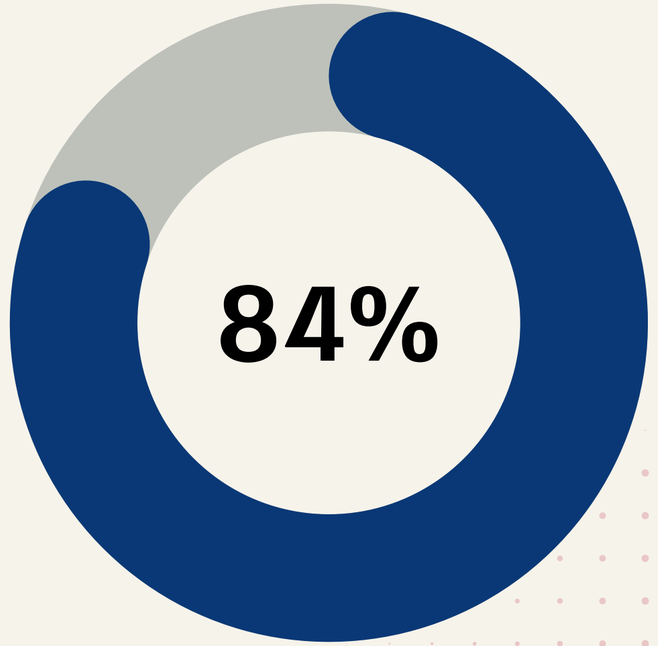
```
     Y_pred = multNB.predict(X_test)
```

```
[ ]  # Model performance
     from sklearn.metrics import accuracy_score
     accuracy_score(y_test_p,Y_pred)
```

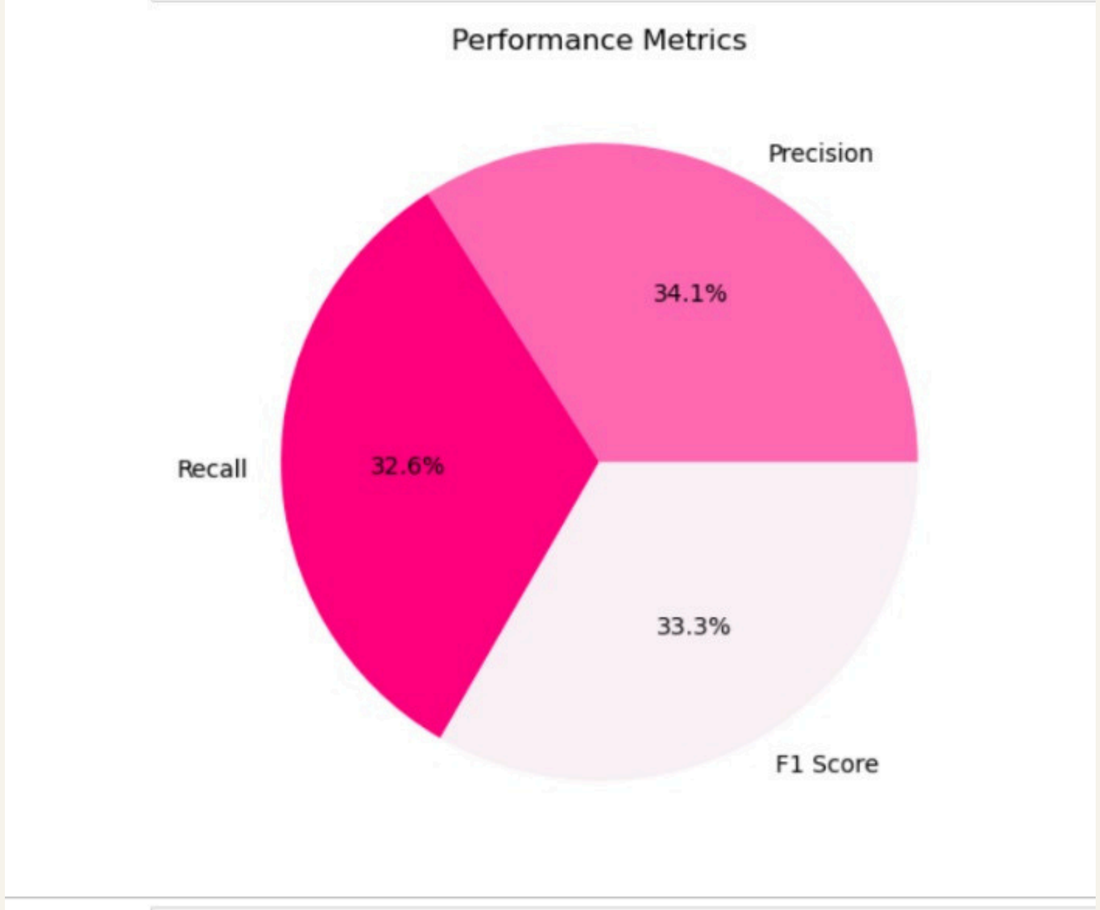
NAIVE BAYES

```
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```





NAIVE BAYES



Classification Report:

	precision	recall	f1-score	support
0	0.82	0.86	0.84	99935
1	0.86	0.82	0.84	100065
accuracy			0.84	200000
macro avg	0.84	0.84	0.84	200000
weighted avg	0.84	0.84	0.84	200000

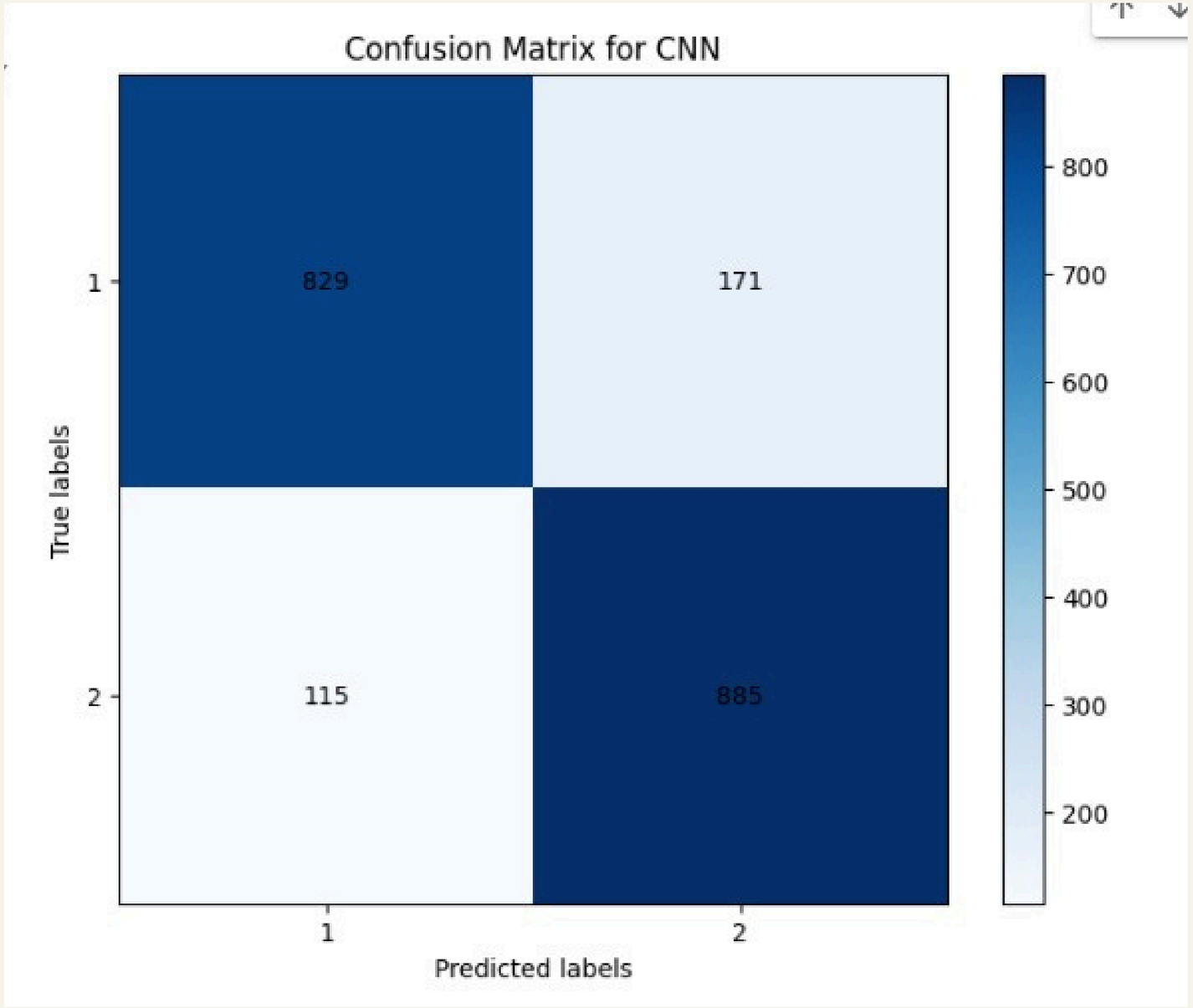
Precision: 0.86
Recall: 0.82
F1 Score: 0.84

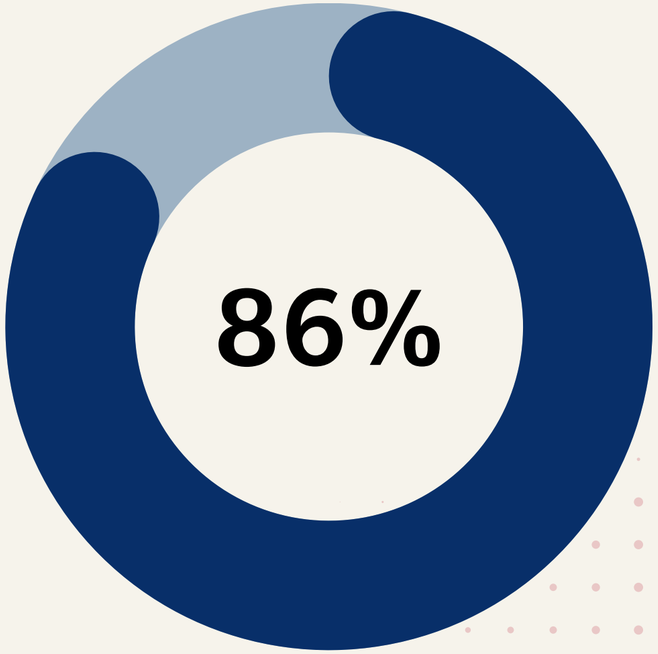
CNN

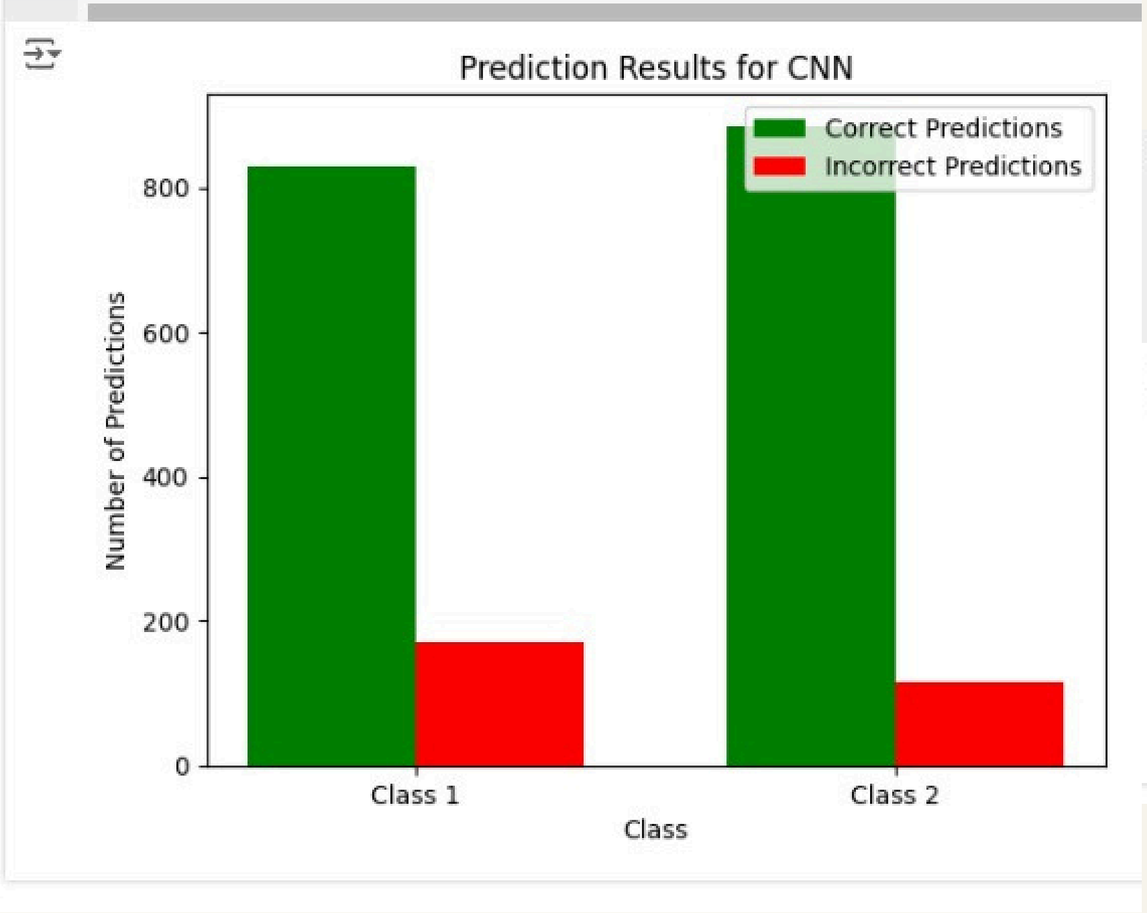

```
#create model arch
model = Sequential()
model.add(Embedding(VOCAB_SIZE, EMBED_SIZE, input_length=MAX_SEQUENCE_LENGTH))
model.add(Conv1D(filters=32, kernel_size=4, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(rate=0.10))
model.add(Conv1D(filters=64, kernel_size=4, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(rate=0.10))
model.add(Conv1D(filters=64, kernel_size=4, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(rate=0.10))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=loss, optimizer=Adam, metrics=['accuracy'])
model.summary()
```

```
# Fit the model
with tf.device('/GPU:0'):
    history1 = model.fit(X_train, y_train, validation_data=(X_val,y_val),epochs=10, batch_size=32, verbose=1, callbacks=[callback])
```

CNN


Confusion Matrix for CNN
True labels
Predicted labels
1
2
829
171
115
885
800
700
600
500
400
300
200


86%

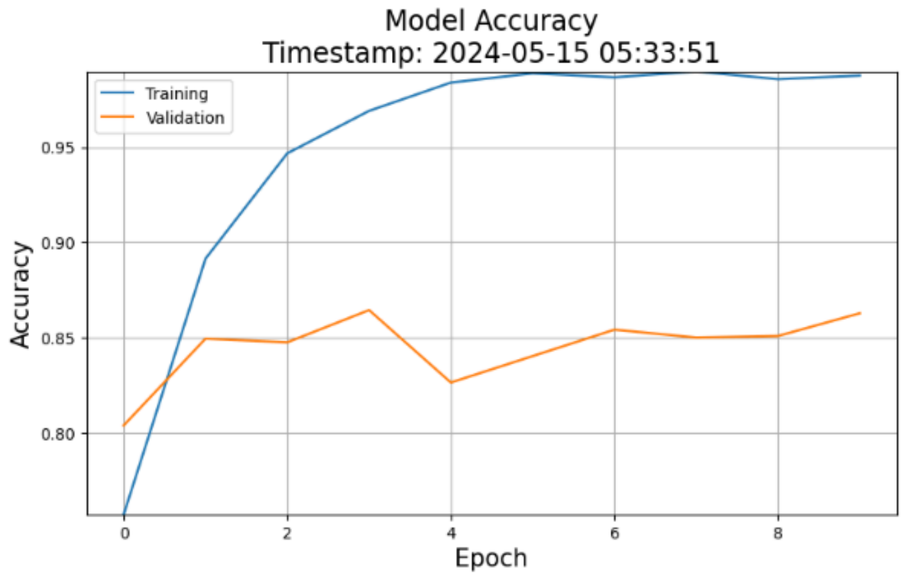


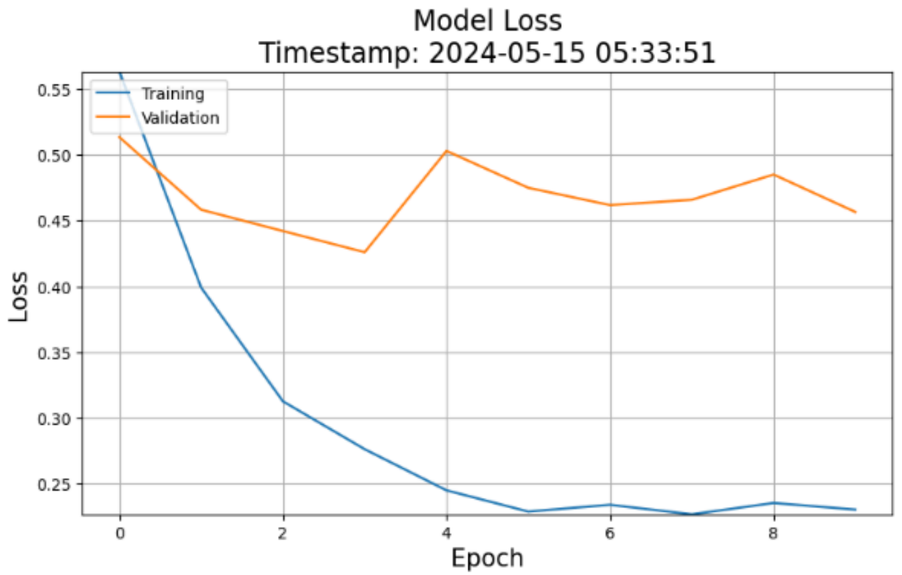

```
# prompt: how to get classification report

print(classification_report(y_true, y_pred, target_names=['1','2']))
```

```
              precision    recall  f1-score   support

           1       0.88      0.83      0.85      1000
           2       0.84      0.89      0.86      1000

    accuracy                           0.86      2000
   macro avg       0.86      0.86      0.86      2000
weighted avg       0.86      0.86      0.86      2000
```

Visualizing and Evaluating the Results

In [ ]:
```
plot_performance(history=history1)
```





RNN

```
voc_size=20000
max_length=100
tokenizer=Tokenizer(num_words=voc_size)
tokenizer.fit_on_texts(train)
word_index=tokenizer.word_index
with open('/kaggle/working/tokenizer.pkl', 'wb') as f:
    pickle.dump(tokenizer, f)
```

```
train=tokenizer.texts_to_sequences(train)
train=pad_sequences(train,maxlen=max_length)
test=tokenizer.texts_to_sequences(test)
test=pad_sequences(test,maxlen=max_length)
```

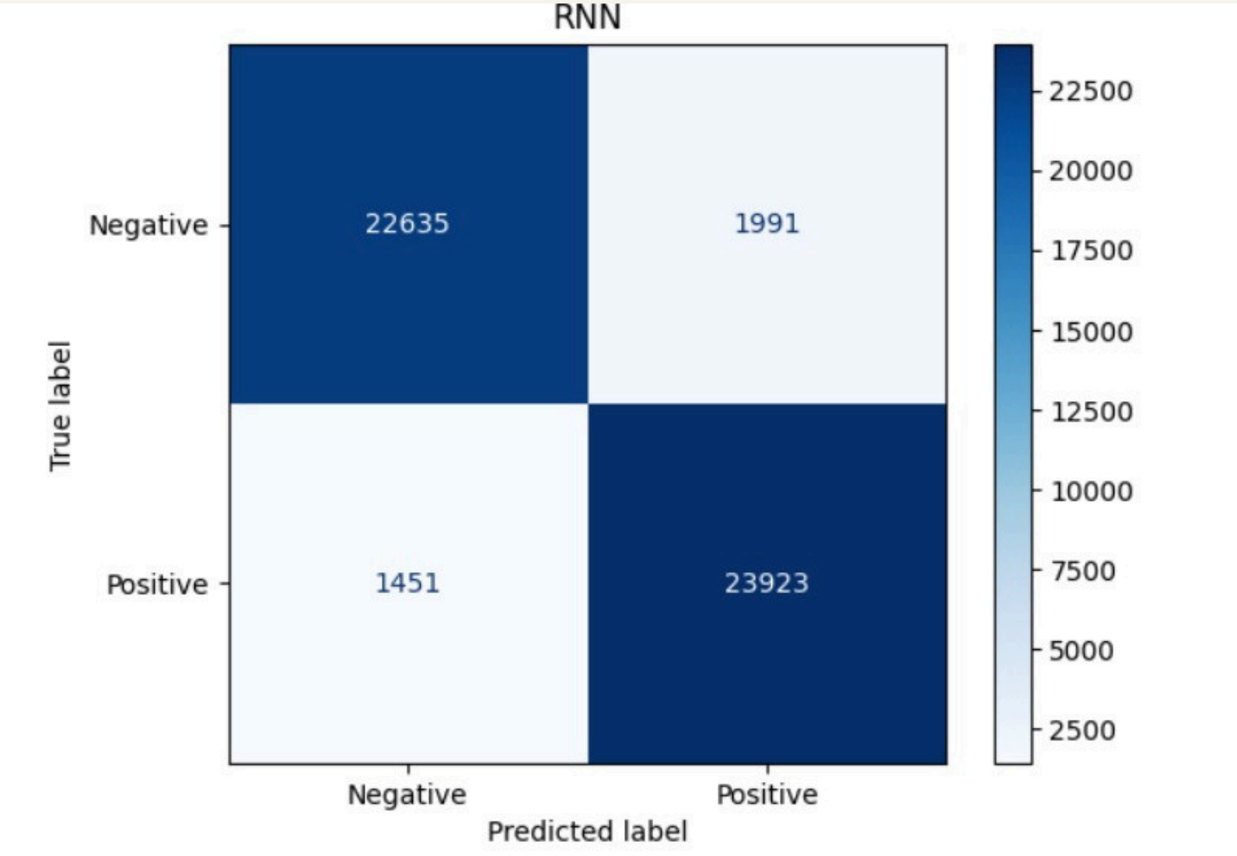
```
model=Sequential()
model.add(Embedding(input_dim=voc_size, output_dim=64, input_length=max_length))

model.add(LSTM(units=32,return_sequences=True))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(units=32))
model.add(Dense(1,activation='sigmoid'))
model.summary()
```

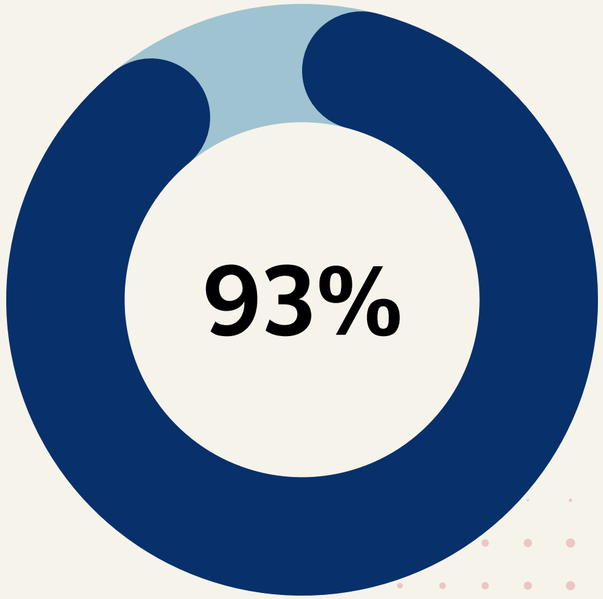
RNN

```
model=Sequential()
model.add(Embedding(input_dim=voc_size, output_dim=64, input_length=max_length))

model.add(LSTM(units=32,return_sequences=True))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(units=32))
model.add(Dense(1,activation='sigmoid'))
model.summary()
```

```
checkpoint_cb=ModelCheckpoint('amazon_model.h5',save_best_only=True)
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
history=model.fit(train,train_lab,epochs=2,validation_split=.2,
                  callbacks=[checkpoint_cb])
```

RNN


RNN
Negative
Positive
True label
22635
1991
1451
23923
Negative
Positive
Predicted label
22500
20000
17500
15000
12500
10000
7500
5000
2500


93%

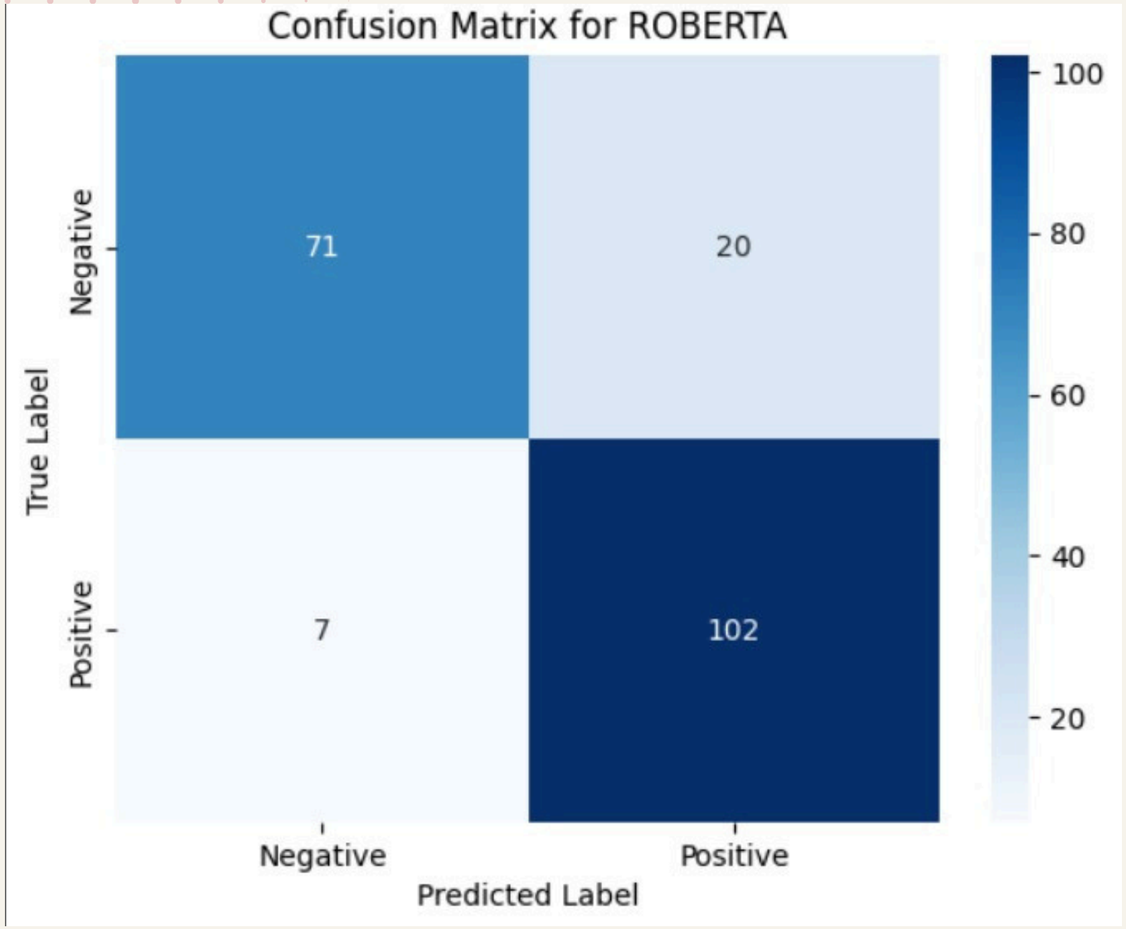
RNN

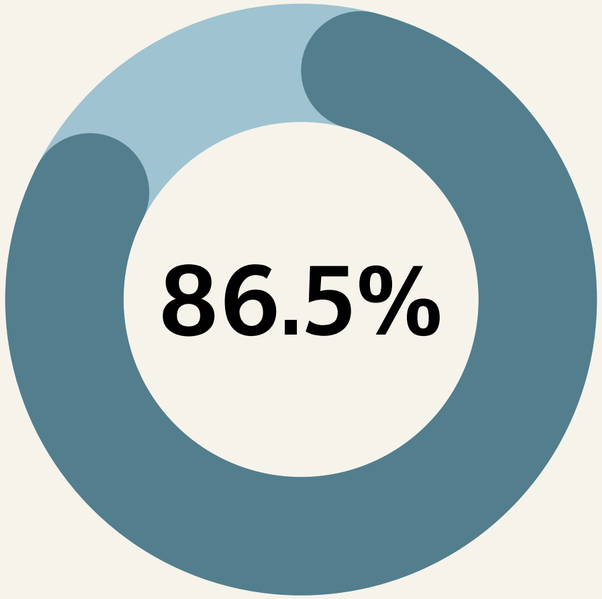
	precision	recall	f1-score	support
0	0.94	0.92	0.93	24626
1	0.92	0.94	0.93	25374
accuracy			0.93	50000
macro avg	0.93	0.93	0.93	50000
weighted avg	0.93	0.93	0.93	50000

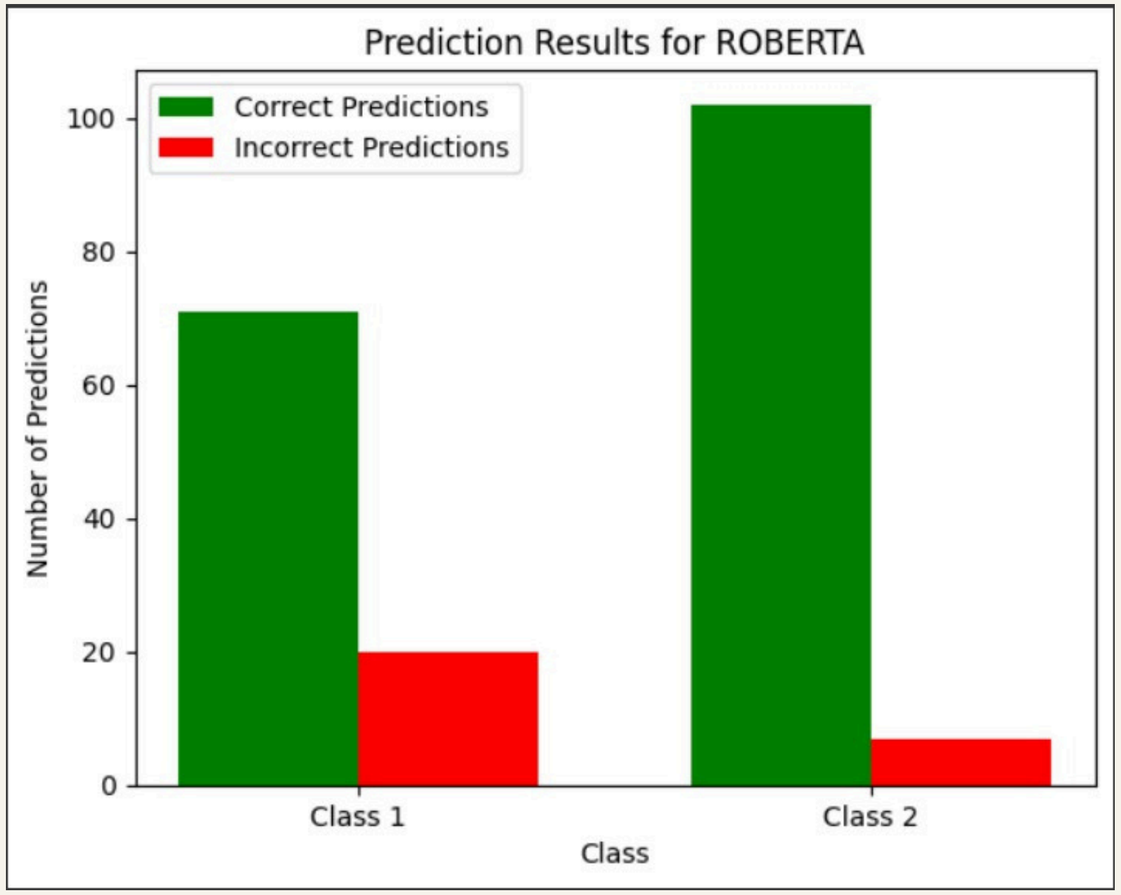
ROBERTA


```
# Assuming test_data is your DataFrame containing 'review' and 'label' columns
reviews = test_data['review'].tolist()
labels = test_data['label']
labels = labels.replace({'1': 1, '2': 2})


# Tokenize the text data
tokenized_data = tokenizer(reviews, padding=True, truncation=True, return_tensors="pt")
with torch.no_grad():
    outputs = model(**tokenized_data)
logits = outputs.logits

# Convert logits to probabilities using softmax
probs = torch.softmax(logits, dim=1)

# Extract the model scores (probabilities for each class)
model_scores = probs.cpu().numpy()
print (model_scores)

def map_scores_to_labels(scores):
    predicted_labels = []
    for score in scores:
        if score[1] > score[0]:  # Check if positive score is higl
            predicted_labels.append(2)  # Map to positive label
        else:
            predicted_labels.append(1)  # Map to negative label
    return predicted_labels

# Map model scores to predicted labels
predicted_labels = map_scores_to_labels(model_scores)
```

```
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification
from scipy.special import softmax

MODEL = f"cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)
```


Confusion Matrix for ROBERTA
Negative
Positive
True Label
71
20
7
102
Negative
Positive
Predicted Label
100
80
60
40
20


86.5%



	precision	recall	f1-score	support
1	0.91	0.78	0.84	91
2	0.84	0.94	0.88	109
accuracy			0.86	200
macro avg	0.87	0.86	0.86	200
weighted avg	0.87	0.86	0.86	200

THANK YOU