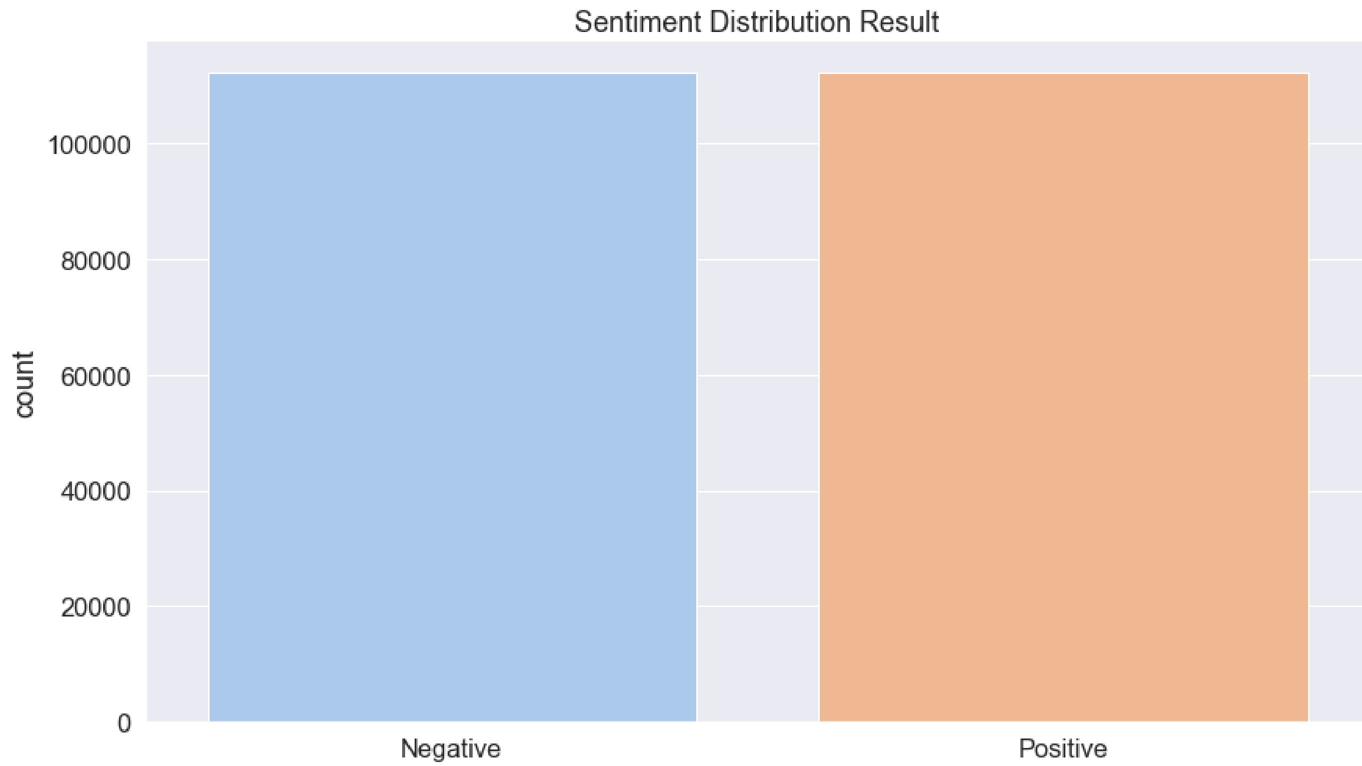
In [2]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import seaborn as sns
from pylab import rcParams
import string
import re
import matplotlib.pyplot as plt
import math
from matplotlib import rc
from sklearn.model_selection import train_test_split
from collections import Counter, defaultdict
from bs4 import BeautifulSoup
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
import nltk
from nltk.corpus import stopwords
from wordcloud import WordCloud


%matplotlib inline

sns.set(style='darkgrid', palette='pastel', font_scale=1.5)

rcParams['figure.figsize'] = 14, 8

RANDOM_SEED = 50

np.random.seed(RANDOM_SEED)
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\esber\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[2]: True

In [7]:
```
train = pd.read_csv("/Users/esber/Documents/PythonProjects/data/TrainDataReview.csv")
test = pd.read_csv("/Users/esber/Documents/PythonProjects/data/TestDataReview.csv")
```

In [8]:
```
f = sns.countplot(x='sentiment', data=train)
f.set_title("Sentiment Distribution Result")
f.set_xticklabels(['Negative', 'Positive'])
plt.xlabel("");
```



In [9]:
```python
text = " ".join(review for review in train.review)

wordcloud = WordCloud(max_font_size=50, max_words=100, background_color="pink", stopwords=
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show();
```

In [10]:
```python
class Tokenizer:

  def clean(self, text):
      no_html = BeautifulSoup(text).get_text()
      clean = re.sub("[^a-z\s]+", " ", no_html, flags=re.IGNORECASE)
      return re.sub("(\s+)", " ", clean)


  def tokenize(self, text):
      clean = self.clean(text).lower()
      stopwords_en = stopwords.words("english")
      return [w for w in re.split("\W+", clean) if not w in stopwords_en]
```

In [11]:

```
class MultinomialNaiveBayes:

    def __init__(self, classes, tokenizer):
      self.tokenizer = tokenizer
      self.classes = classes

    def group_by_class(self, X, y):
      data = dict()
      for c in self.classes:
        data[c] = X[np.where(y == c)]
      return data

    def fit(self, X, y):
        self.n_class_items = {}
        self.log_class_priors = {}
        self.word_counts = {}
        self.vocab = set()

        n = len(X)

        grouped_data = self.group_by_class(X, y)

        for c, data in grouped_data.items():
          self.n_class_items[c] = len(data)
          self.log_class_priors[c] = math.log(self.n_class_items[c] / n)
          self.word_counts[c] = defaultdict(lambda: 0)

          for text in data:
            counts = Counter(self.tokenizer.tokenize(text))
            for word, count in counts.items():
                if word not in self.vocab:
                    self.vocab.add(word)

                self.word_counts[c][word] += count

        return self

    def laplace_smoothing(self, word, text_class):
      num = self.word_counts[text_class][word] + 1
      denom = self.n_class_items[text_class] + len(self.vocab)
      return math.log(num / denom)

    def predict(self, X):
        result = []
        for text in X:

          class_scores = {c: self.log_class_priors[c] for c in self.classes}

          words = set(self.tokenizer.tokenize(text))
          for word in words:
              if word not in self.vocab: continue

              for c in self.classes:

                log_w_given_c = self.laplace_smoothing(word, c)
                class_scores[c] += log_w_given_c

          result.append(max(class_scores, key=class_scores.get))

        return result
```

In [12]:
```
X = train['review'].values
y = train['sentiment'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=RAND(
```

In [14]:
```
MNB = MultinomialNaiveBayes(
    classes=np.unique(y),
    tokenizer=Tokenizer()
).fit(X_train, y_train)
```

In [15]:
```
y_hat = MNB.predict(X_test)
```

In [16]:
```
accuracy_score(y_test, y_hat)
```

Out[16]:
```
0.747365372189916
```

In [17]:
```
print(classification_report(y_test, y_hat))
```

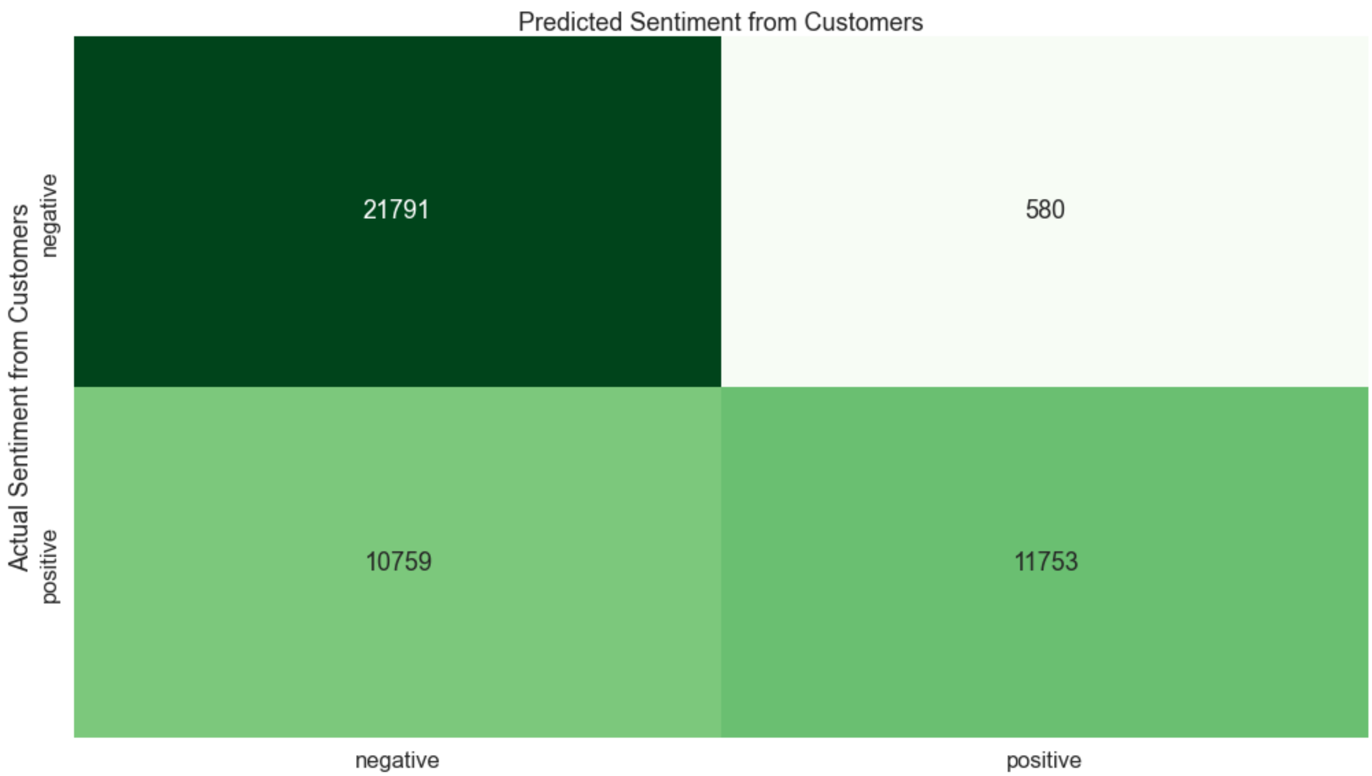
```
              precision    recall  f1-score   support

           0       0.67      0.97      0.79     22371
           1       0.95      0.52      0.67     22512

    accuracy                           0.75     44883
   macro avg       0.81      0.75      0.73     44883
weighted avg       0.81      0.75      0.73     44883
```

In [18]:
```
cnf_matrix = confusion_matrix(y_test, y_hat)
cnf_matrix
```

Out[18]:
```
array([[21791,   580],
       [10759, 11753]], dtype=int64)
```

In [29]:

```
class_names = ["negative", "positive"]
fig,ax = plt.subplots()


sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, fmt="d", cmap="Greens", cbar=False, xticl
ax.xaxis.set_label_position('top')
plt.tight_layout()
plt.ylabel('Actual Sentiment from Customers')
plt.xlabel('Predicted Sentiment from Customers');
```



In [ ]:

In [ ]:

In [ ]: