2048010

NLP - Case Study

E-Commerce Clothing Review Classification with TF

In [1]:

```
import os
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import plotly.graph_objects as go
from wordcloud import WordCloud
from sklearn.model_selection import train_test_split
import numpy as np
import re
import plotly.express as px
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Dropout
from keras.layers import LSTM
from keras.models import Sequential
from keras.layers import Bidirectional
from keras.layers import Embedding
from keras.layers import GlobalAvgPool1D
import tensorflow as tf


for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tools\_testing.py:19: FutureWarning: pandas.util.te
sting is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [2]:

```
data = pd.read_csv("E:/Christ University/Semester-3/04_NLP/Case_Study/data/Womens Clothing E-Commerce R
data.head()
```

Out[2]:

	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Division Name	Department Name	Class Name
0	767	33	NaN	Absolutely wonderful - silky and sexy and comf...	4	1	0	Initmates	Intimate	Intimates
1	1080	34	NaN	Love this dress! it's sooo pretty. i happene...	5	1	4	General	Dresses	Dresses
2	1077	60	Some major design flaws	I had such high hopes for this dress and reall...	3	0	0	General	Dresses	Dresses
3	1049	50	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	5	1	0	General Petite	Bottoms	Pants
4	847	47	Flattering shirt	This shirt is very flattering to all due to th...	5	1	6	General	Tops	Blouses

As you can see from the head of the dataset, we have some unnecessary features such as Clothing ID, Title. First of all, I will drop this features.

In [3]:
```
data = data.drop(['Title', 'Clothing ID', 'Positive Feedback Count'], axis=1)
data.head()
```

Out[3]:

	Age	Review Text	Rating	Recommended IND	Division Name	Department Name	Class Name
0	33	Absolutely wonderful - silky and sexy and comf...	4	1	Initmates	Intimate	Intimates
1	34	Love this dress! it's sooo pretty. i happene...	5	1	General	Dresses	Dresses
2	60	I had such high hopes for this dress and reall...	3	0	General	Dresses	Dresses
3	50	I love, love, love this jumpsuit. it's fun, fl...	5	1	General Petite	Bottoms	Pants
4	47	This shirt is very flattering to all due to th...	5	1	General	Tops	Blouses

In [4]:
```
# Checking for the missing values
count_NaN = data.isna().sum()
count_NaN
```

Out[4]:
```
Age                  0
Review Text        845
Rating               0
Recommended IND      0
Division Name       14
Department Name     14
Class Name          14
dtype: int64
```

We will drop the missing rows from the dataset.

In [5]:

```
# Dropping the missing values in the rows
data = data.dropna(subset=['Review Text', 'Division Name', 'Department Name', 'Class Name'], axis=0)
data = data.reset_index(drop=True)

# Checking for the missing values after the drops
count_NaN_updated = data.isna().sum()
count_NaN_updated
```

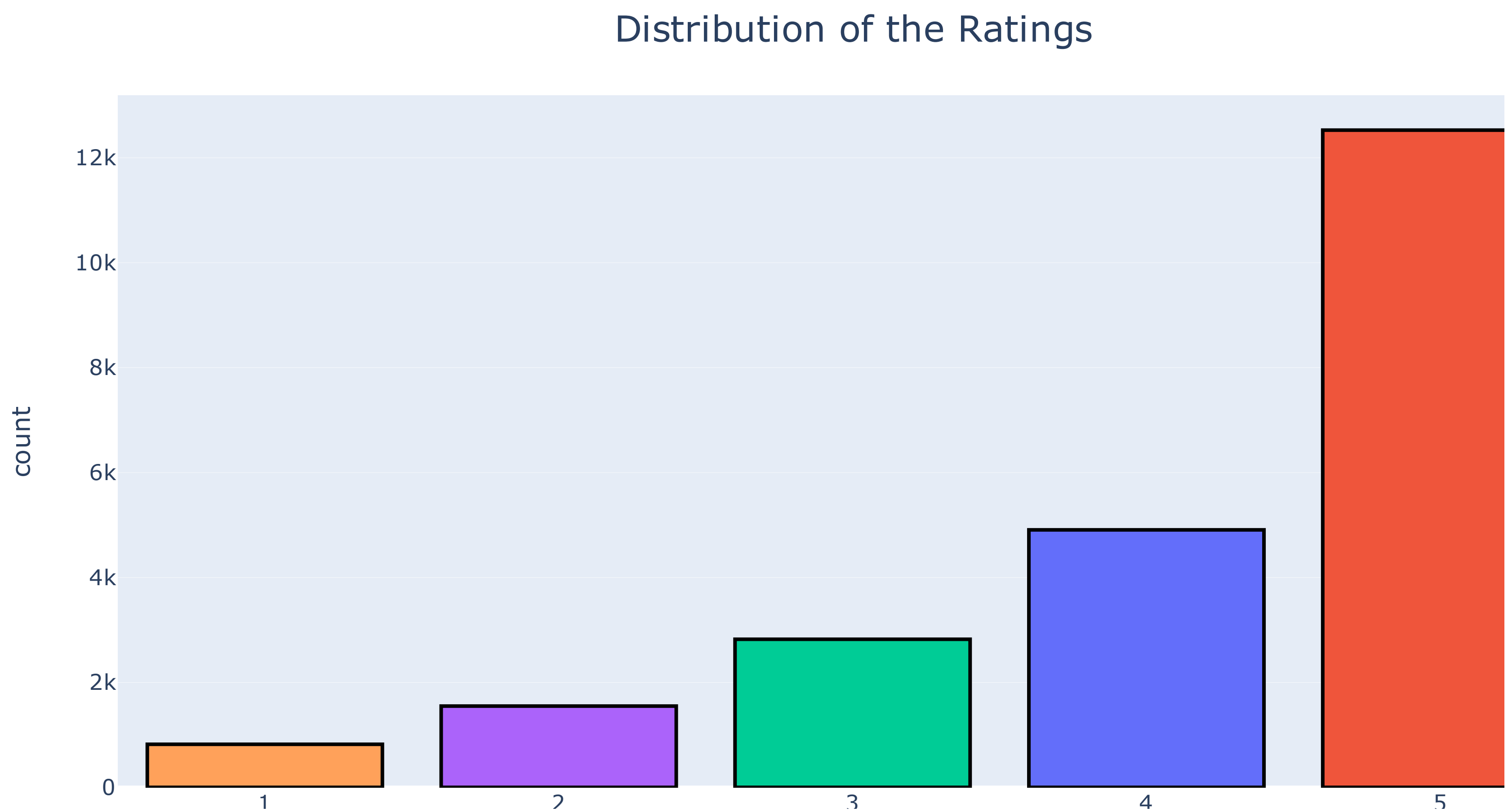
Out[5]:

```
Age                0
Review Text        0
Rating             0
Recommended IND    0
Division Name      0
Department Name    0
Class Name         0
dtype: int64
```

Distribution of the Ratings

In [6]:

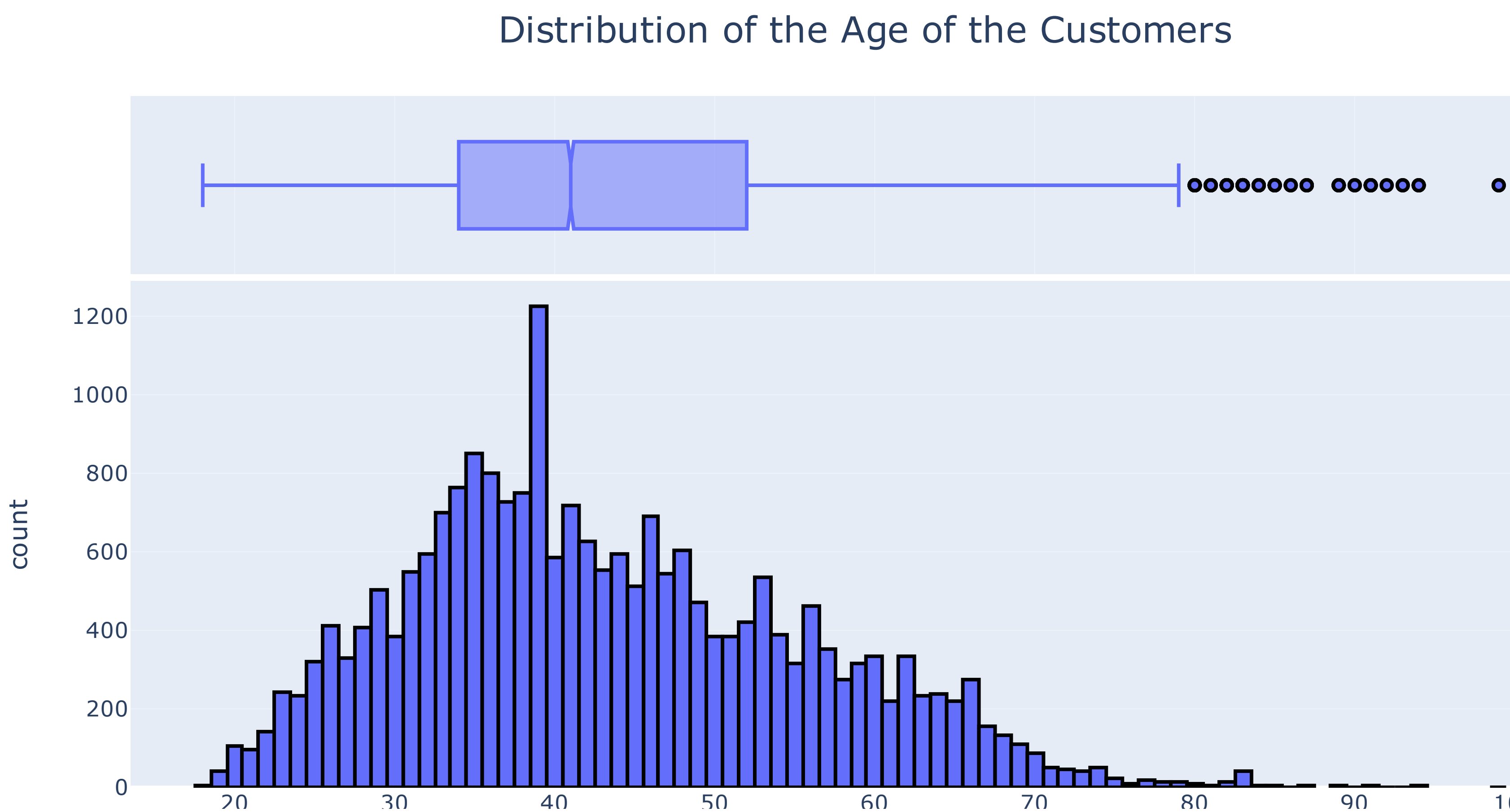
```
fig = px.histogram(data['Rating'],
                   labels={'value': 'Rating',
                           'count': 'Frequency',
                           'color': 'Rating'}, color=data['Rating'])
fig.update_layout(bargap=0.2)
fig.update_traces(marker=dict(line=dict(color='#000000', width=2)))
fig.update_layout(title_text='Distribution of the Ratings',
                  title_x=0.5, title_font=dict(size=20))
fig.show()
```



According to the graph above, frequency of the Rating 5 is pretty high compared to the others.

Distribution of the Age of the Customers

In [7]:
```
fig = px.histogram(data['Age'], marginal='box',
                   labels={'value': 'Age'})

fig.update_traces(marker=dict(line=dict(color='#000000', width=2)))
fig.update_layout(title_text='Distribution of the Age of the Customers',
                  title_x=0.5, title_font=dict(size=20))
fig.show()
```

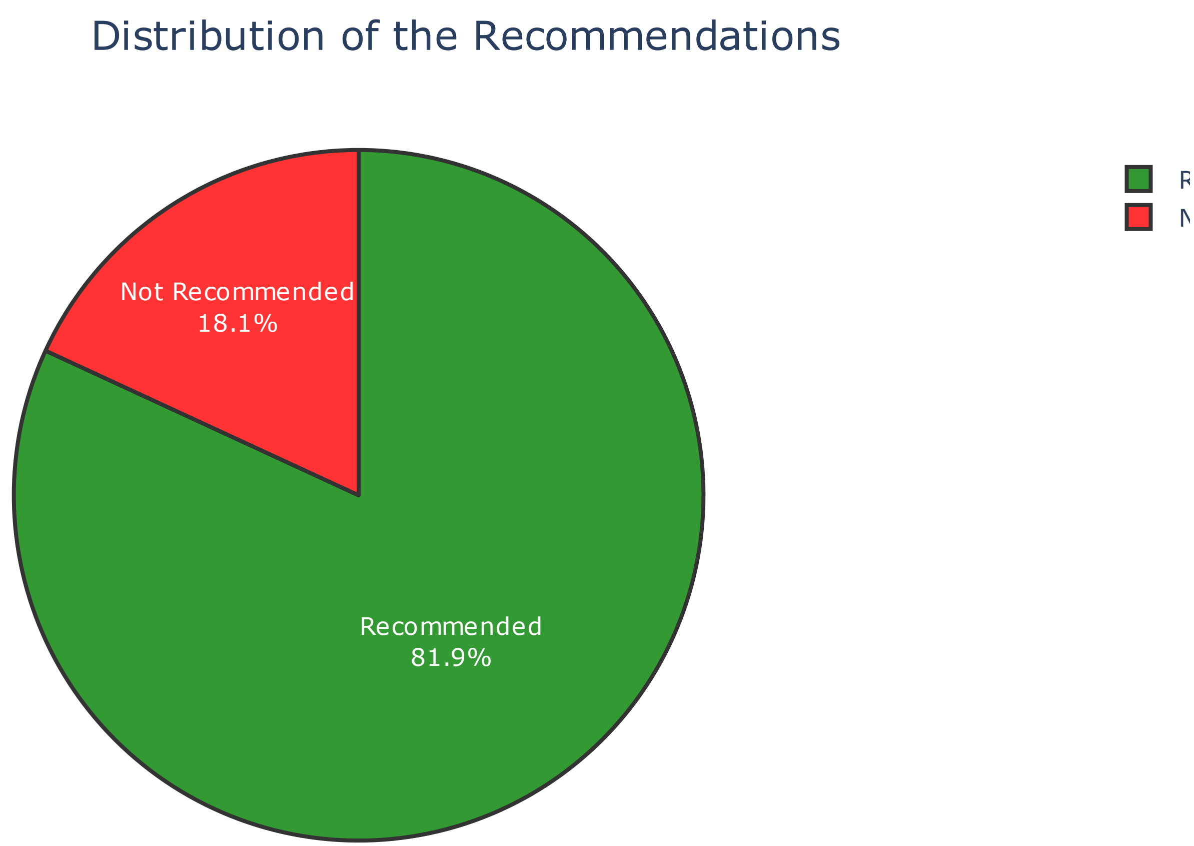


As you can see from the 'Distribution of the Age of the Customers' graph, the age of the customers is usually distributed between 34 and 52. We have outliers that customers older than 80.

Distribution of the Recommendations

In [8]:

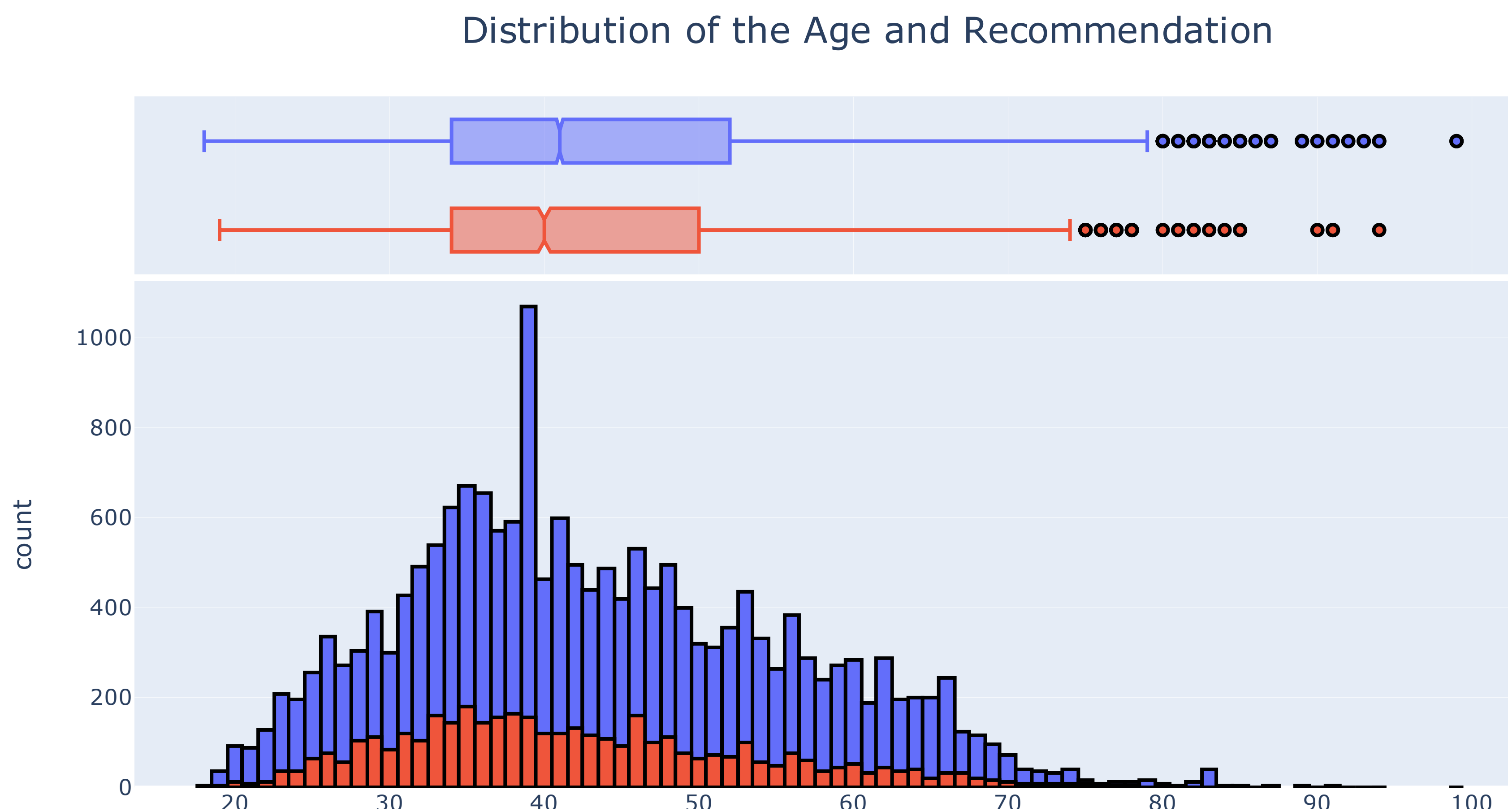
```
labels = ['Recommended', 'Not Recommended']
values = [data[data['Recommended IND'] == 1]['Recommended IND'].value_counts()[1],
          data[data['Recommended IND'] == 0]['Recommended IND'].value_counts()[0]]
colors = ['green', 'red']

fig = go.Figure(data=[go.Pie(labels=labels, values=values, opacity=0.8)])
fig.update_traces(textinfo='percent+label', marker=dict(line=dict(color='#000000', width=2), colors=col
fig.update_layout(title_text='Distribution of the Recommendations', title_x=0.5, title_font=dict(size=2
fig.show()
```



According to this pie chart, the most of the sales are Recommended.

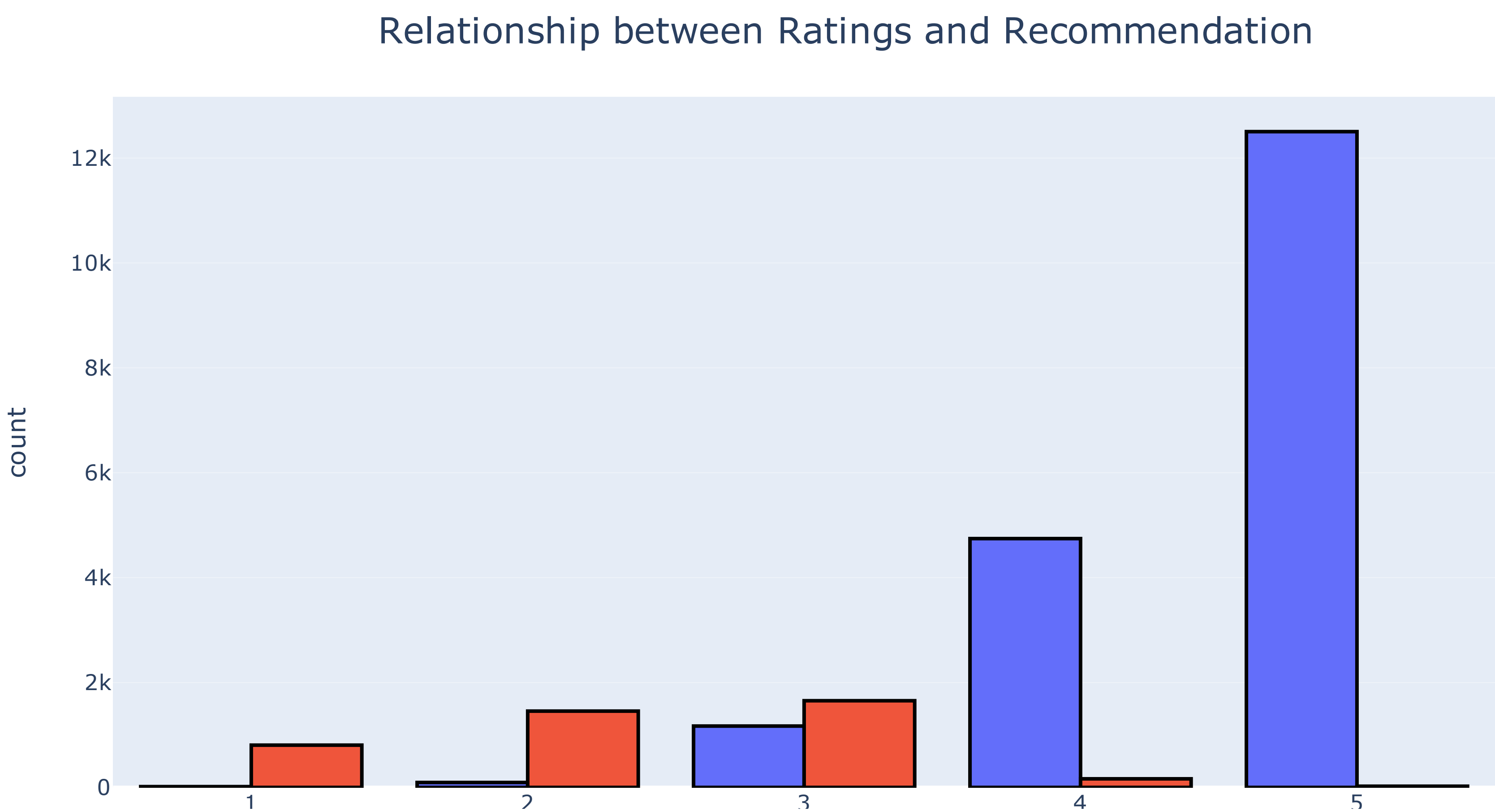
Distribution of the Age and Recommendation

In [9]:
```
fig = px.histogram(data['Age'], color=data['Recommended IND'],
                   labels={'value': 'Age',
                           'color': 'Recommended'}, marginal='box')
fig.update_traces(marker=dict(line=dict(color='#000000', width=2)))
fig.update_layout(title_text='Distribution of the Age and Recommendation',
                  title_x=0.5, title_font=dict(size=20))
fig.update_layout(barmode='overlay')
fig.show()
```



Relationship between Ratings and Recommendation

In [10]:
```
fig = px.histogram(data['Rating'], color=data['Recommended IND'],
                   labels={'value': 'Rating',
                           'color': 'Recommended?'})
fig.update_layout(bargap=0.2)
fig.update_traces(marker=dict(line=dict(color='#000000', width=2)))
fig.update_layout(title_text='Relationship between Ratings and Recommendation',
                  title_x=0.5, title_font=dict(size=20))
fig.update_layout(barmode='group')
fig.show()
```



According to this graph above, almost all the Rating 5 and Rating 4 data points are recommended.

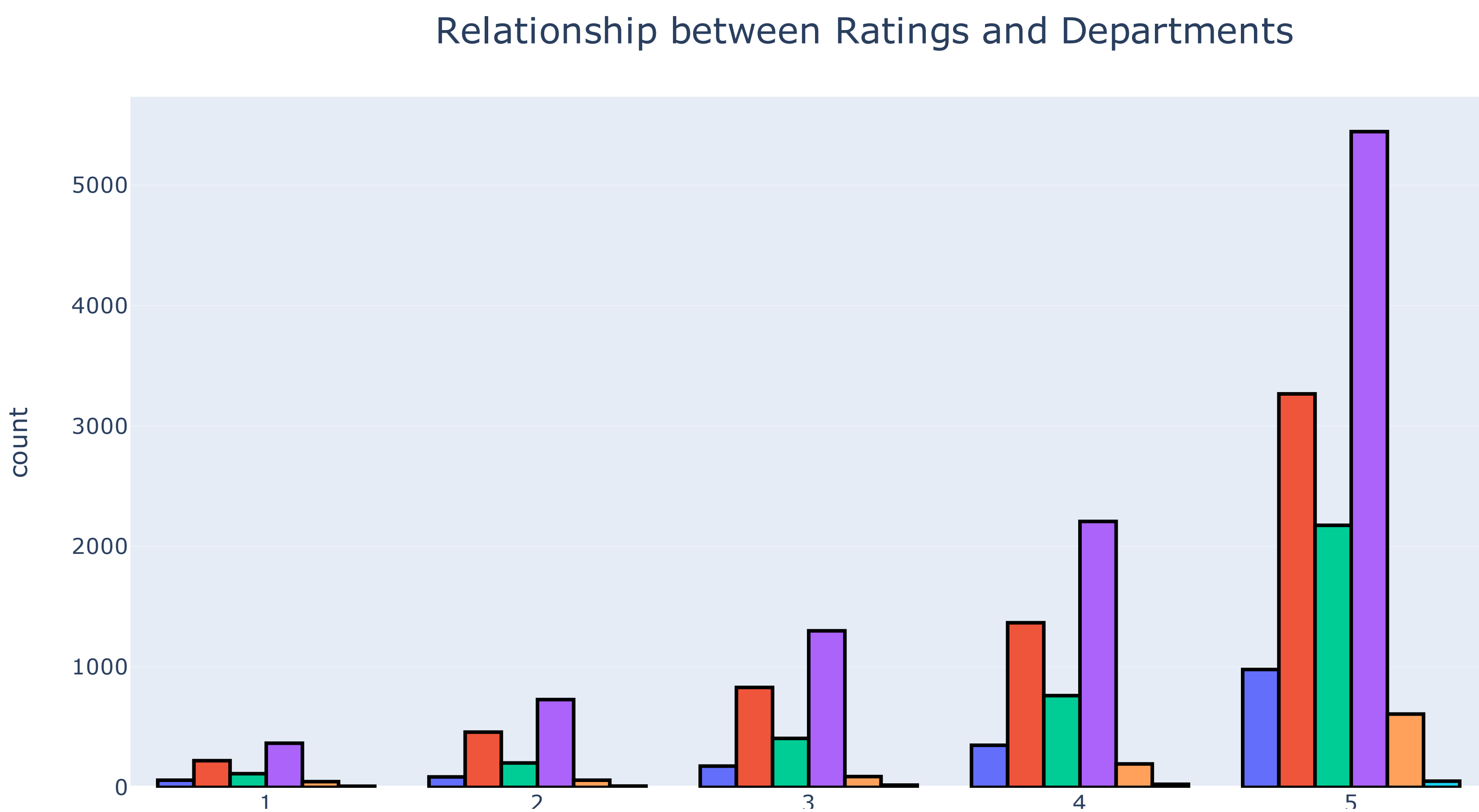
In addition, Rating 1 and Rating 2 data points have almost no recommendations.

For the further steps, I would create a common rating point with the Rating 4 and Rating 5 as well as Rating 1 and Rating 2. In this way, I would shrink the labels therefore, the model would perform better.

Relationship between Ratings and Departments

In [11]:

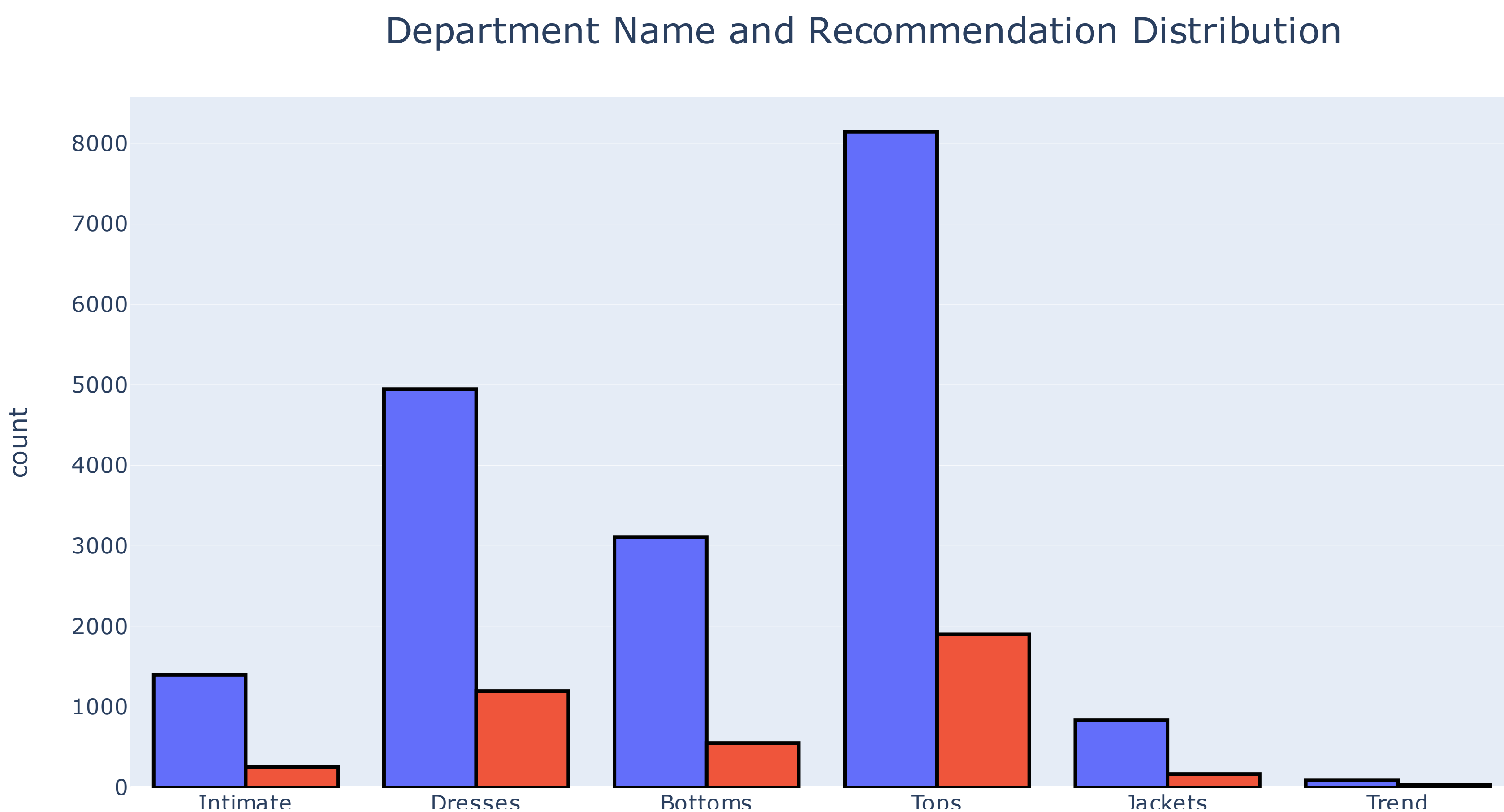
```
fig = px.histogram(data['Rating'], color=data['Department Name'],
                   labels={'value': 'Rating',
                           'color': 'Department Name'})
fig.update_layout(bargap=0.2)
fig.update_traces(marker=dict(line=dict(color='#000000', width=2)))
fig.update_layout(title_text='Relationship between Ratings and Departments',
                  title_x=0.5, title_font=dict(size=20))
fig.update_layout(barmode='group')
fig.show()
```



According to the graph above, Tops and Dresses have the most of the rating points. Trend and Jackets have the least.

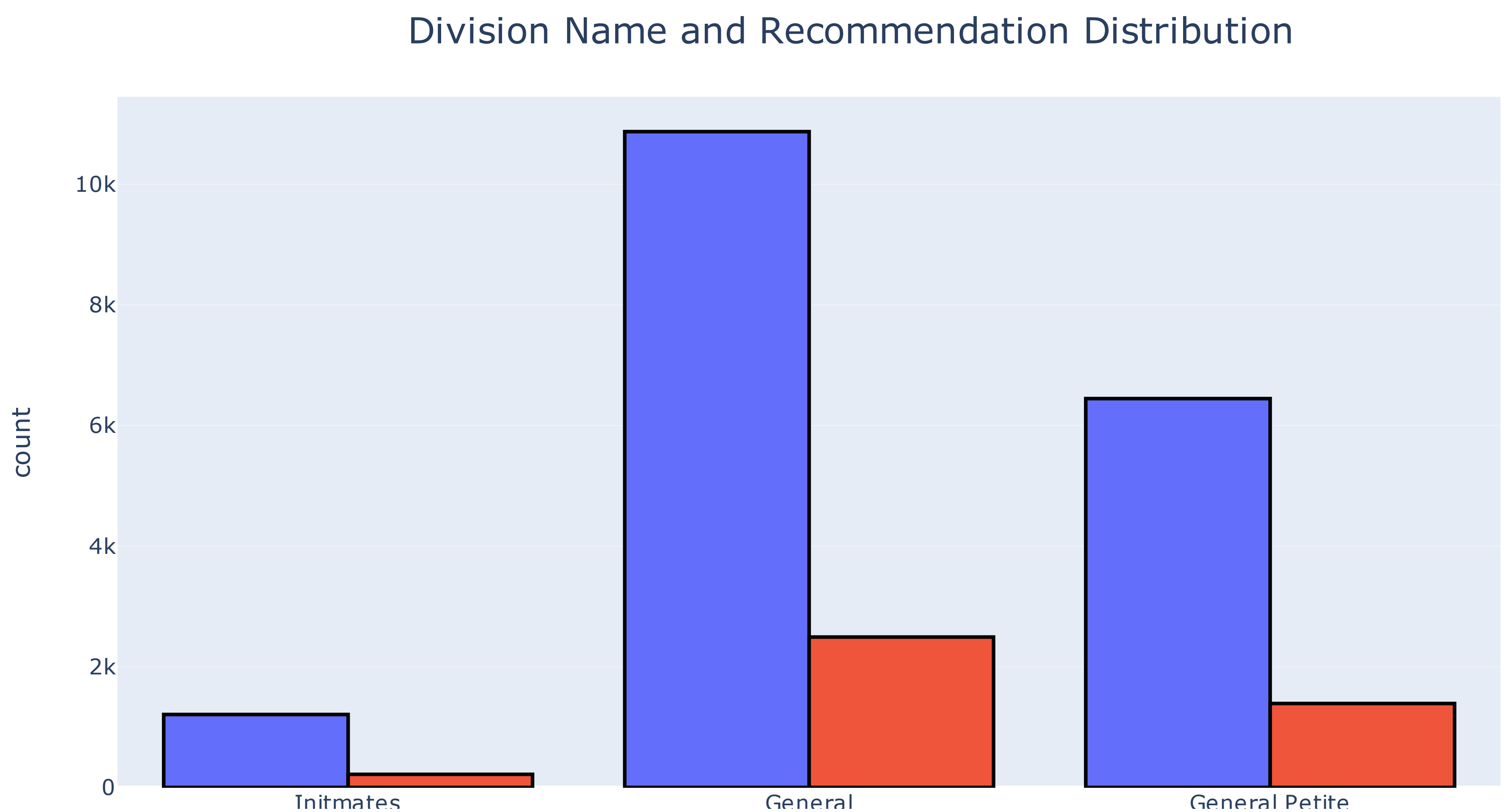
Department and Recommendation Distribution

In [12]:
```
fig = px.histogram(data['Department Name'], color=data['Recommended IND'],
                   labels={'value': 'Department Name',
                           'color': 'Recommended?'})
fig.update_layout(bargap=0.2)
fig.update_traces(marker=dict(line=dict(color='#000000', width=2)))
fig.update_layout(title_text='Department Name and Recommendation Distribution',
                  title_x=0.5, title_font=dict(size=20))
fig.update_layout(barmode='group')
fig.show()
```



Division and Recommendation Distribution

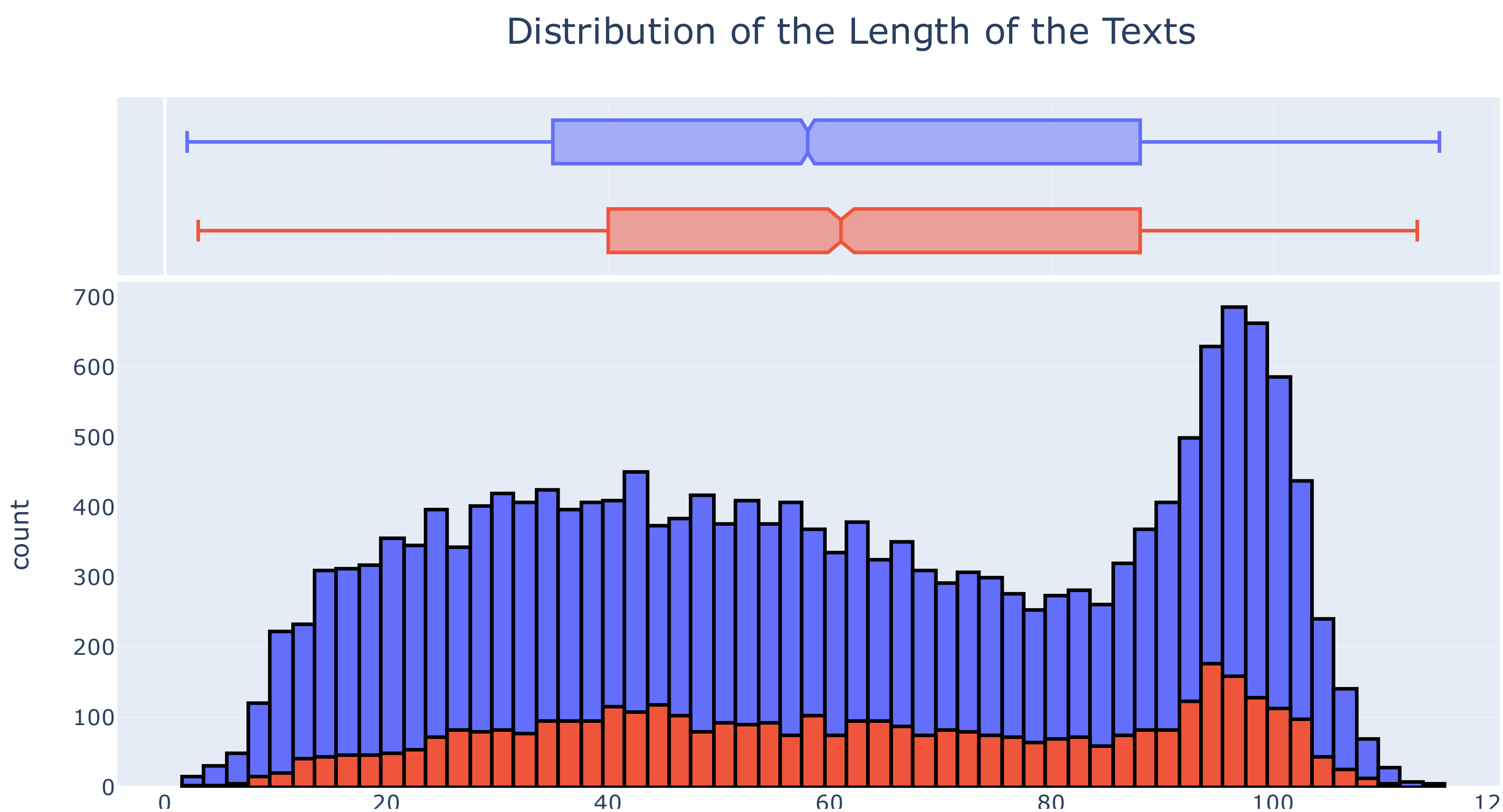
In [13]:
```
fig = px.histogram(data['Division Name'], color=data['Recommended IND'],
                   labels={'value': 'Division Name',
                           'color': 'Recommended?'})
fig.update_layout(bargap=0.2)
fig.update_traces(marker=dict(line=dict(color='#000000', width=2)))
fig.update_layout(title_text='Division Name and Recommendation Distribution',
                  title_x=0.5, title_font=dict(size=20))
fig.update_layout(barmode='group')
fig.show()
```



Distribution of the Length of the Texts

In [14]:

```
data['length_of_text'] = [len(i.split(' ')) for i in data['Review Text']]
fig = px.histogram(data['length_of_text'], marginal='box',
                   labels={"value": "Length of the Text",
                           "color": 'Recommended'},
                   color=data['Recommended IND'])

fig.update_traces(marker=dict(line=dict(color='#000000', width=2)))
fig.update_layout(title_text='Distribution of the Length of the Texts',
                  title_x=0.5, title_font=dict(size=20))
fig.update_layout(barmode='overlay')
fig.show()
```

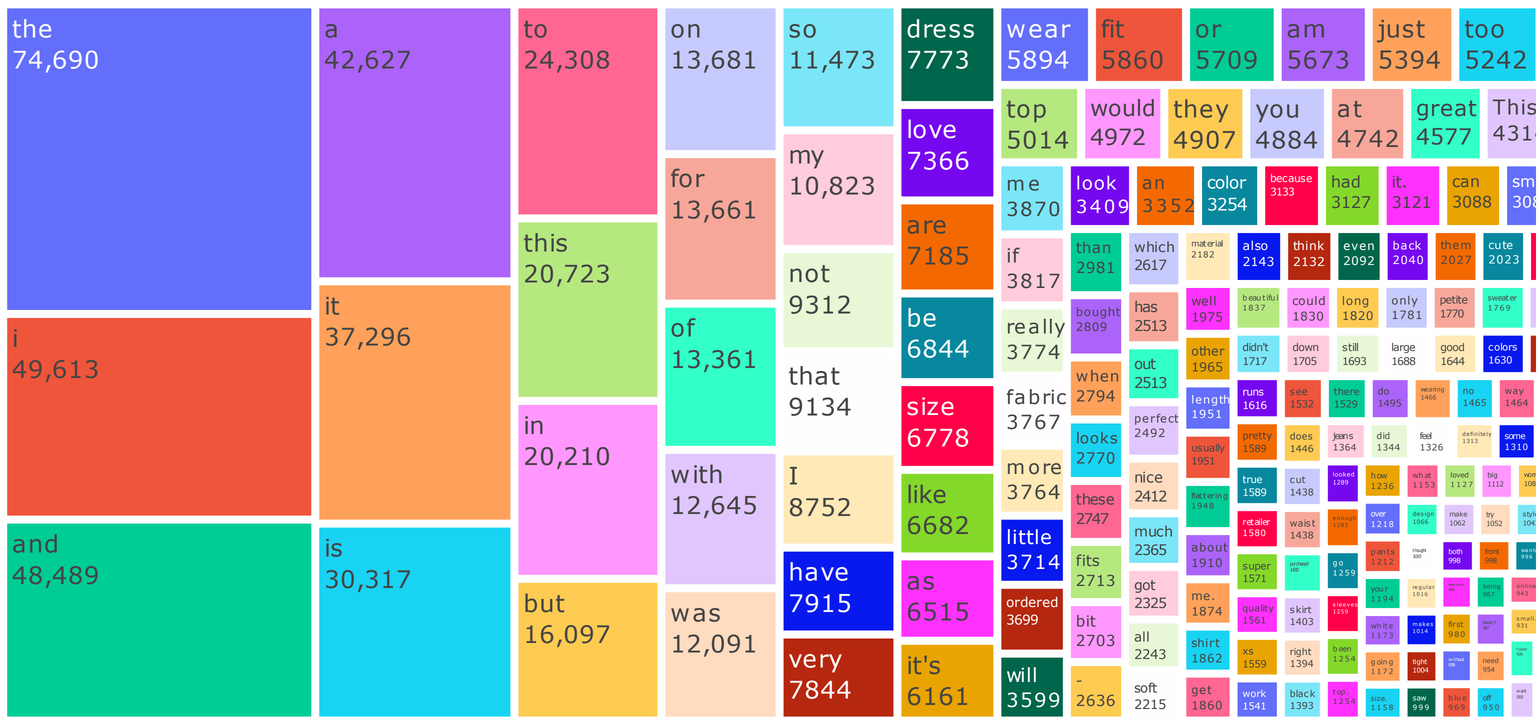


As you can see from the figure above, Recommended and not Recommended products almost have the same distribution length of text.

Top Frequent 200 Words in the Dataset (Before Cleaning)

In [15]:
```
FreqOfWords = data['Review Text'].str.split(expand=True).stack().value_counts()
FreqOfWords_top200 = FreqOfWords[:200]

fig = px.treemap(FreqOfWords_top200, path=[FreqOfWords_top200.index], values=0)
fig.update_layout(title_text='Top Frequent 200 Words in the Dataset (Before Cleaning)',
                  title_x=0.5, title_font=dict(size=20)
                  )
fig.update_traces(textinfo="label+value")
fig.show()
```

Top Frequent 200 Words in the Dataset (Before Cleaning)



According to this Treemap above, the top frequent 200 words usually include a stopword. For the further step of this notebook, I will remove them from the text.

Data Preprocessing

In [16]:
```
# Lower Character all the Texts
data['Review Text'] = data['Review Text'].str.lower()
data['Review Text'].head()
```

Out[16]:
```
0    absolutely wonderful - silky and sexy and comf...
1    love this dress!  it's sooo pretty.  i happene...
2    i had such high hopes for this dress and reall...
3    i love, love, love this jumpsuit. it's fun, fl...
4    this shirt is very flattering to all due to th...
Name: Review Text, dtype: object
```

In [17]:
```
# Removing Punctuations and Numbers from the Text
def remove_punctuations_numbers(inputs):
    return re.sub(r'[^a-zA-Z]', ' ', inputs)


data['Review Text'] = data['Review Text'].apply(remove_punctuations_numbers)
```

Now we will remove all punctuations and numbers from the all dataframe. They will be not usefull for my model training.

Tokenizing with NLTK

In [18]:
```
def tokenization(inputs):  # Ref.1
    return word_tokenize(inputs)


data['text_tokenized'] = data['Review Text'].apply(tokenization)
data['text_tokenized'].head()
```

Out[18]:
```
0    [absolutely, wonderful, silky, and, sexy, and,...
1    [love, this, dress, it, s, sooo, pretty, i, ha...
2    [i, had, such, high, hopes, for, this, dress, ...
3    [i, love, love, love, this, jumpsuit, it, s, f...
4    [this, shirt, is, very, flattering, to, all, d...
Name: text_tokenized, dtype: object
```

Stopwords Removal

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

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\NISHANTH\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
```

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

In [21]:
```
stop_words = set(stopwords.words('english'))
stop_words.remove('not')


def stopwords_remove(inputs):  # Ref.2
    return [k for k in inputs if k not in stop_words]


data['text_stop'] = data['text_tokenized'].apply(stopwords_remove)
data['text_stop'].head()
```

Out[21]:
```
0    [absolutely, wonderful, silky, sexy, comfortable]
1    [love, dress, sooo, pretty, happened, find, st...
2    [high, hopes, dress, really, wanted, work, ini...
3    [love, love, love, jumpsuit, fun, flirty, fabu...
4    [shirt, flattering, due, adjustable, front, ti...
Name: text_stop, dtype: object
```

Lemmatization

In [22]:
```
lemmatizer = WordNetLemmatizer()


def lemmatization(inputs):  # Ref.1
    return [lemmatizer.lemmatize(word=kk, pos='v') for kk in inputs]


data['text_lemmatized'] = data['text_stop'].apply(lemmatization)
data['text_lemmatized'].head()
```

Out[22]:
```
0    [absolutely, wonderful, silky, sexy, comfortable]
1    [love, dress, sooo, pretty, happen, find, stor...
2    [high, hop, dress, really, want, work, initial...
3    [love, love, love, jumpsuit, fun, flirty, fabu...
4    [shirt, flatter, due, adjustable, front, tie, ...
Name: text_lemmatized, dtype: object
```

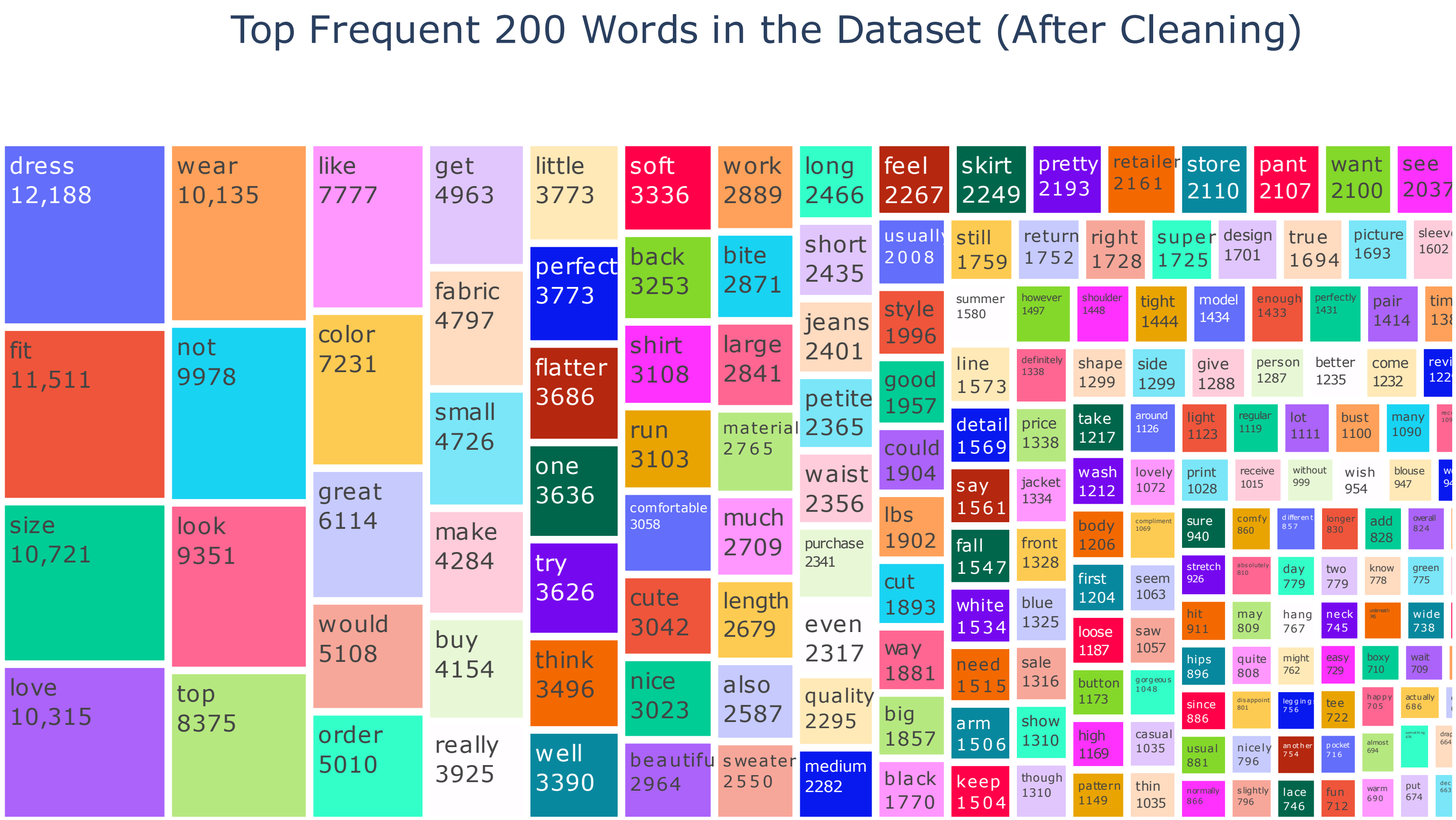
In [23]:
```
# Removing Words less than Length 2
def remove_less_than_2(inputs):  # Ref.1
    return [j for j in inputs if len(j) > 2]


data['final'] = data['text_lemmatized'].apply(remove_less_than_2)
```

In [24]:
```
# Joining Tokens into Sentences
data['final'] = data['final'].str.join(' ')
data['final'].head()
```

Out[24]:
```
0          absolutely wonderful silky sexy comfortable
1    love dress sooo pretty happen find store glad ...
2    high hop dress really want work initially orde...
3    love love love jumpsuit fun flirty fabulous ev...
4    shirt flatter due adjustable front tie perfect...
Name: final, dtype: object
```

Top Frequent 200 Words in the Dataset (After Cleaning)

In [25]:
```python
FreqOfWords = data['final'].str.split(expand=True).stack().value_counts()
FreqOfWords_top200 = FreqOfWords[:200]

fig = px.treemap(FreqOfWords_top200, path=[FreqOfWords_top200.index], values=0)
fig.update_layout(title_text='Top Frequent 200 Words in the Dataset (After Cleaning)',
                  title_x=0.5, title_font=dict(size=20)
                  )
fig.update_traces(textinfo="label+value")
fig.show()
```



As you can see from the Treemap above, all of the words are unique words and there are no stopwords in this set. Most words are 'dress', 'fit' and 'size'. Due to we are dealing with the clothing review dataset, this is pretty reasonable.

WordCloud of the Recommended Reviews

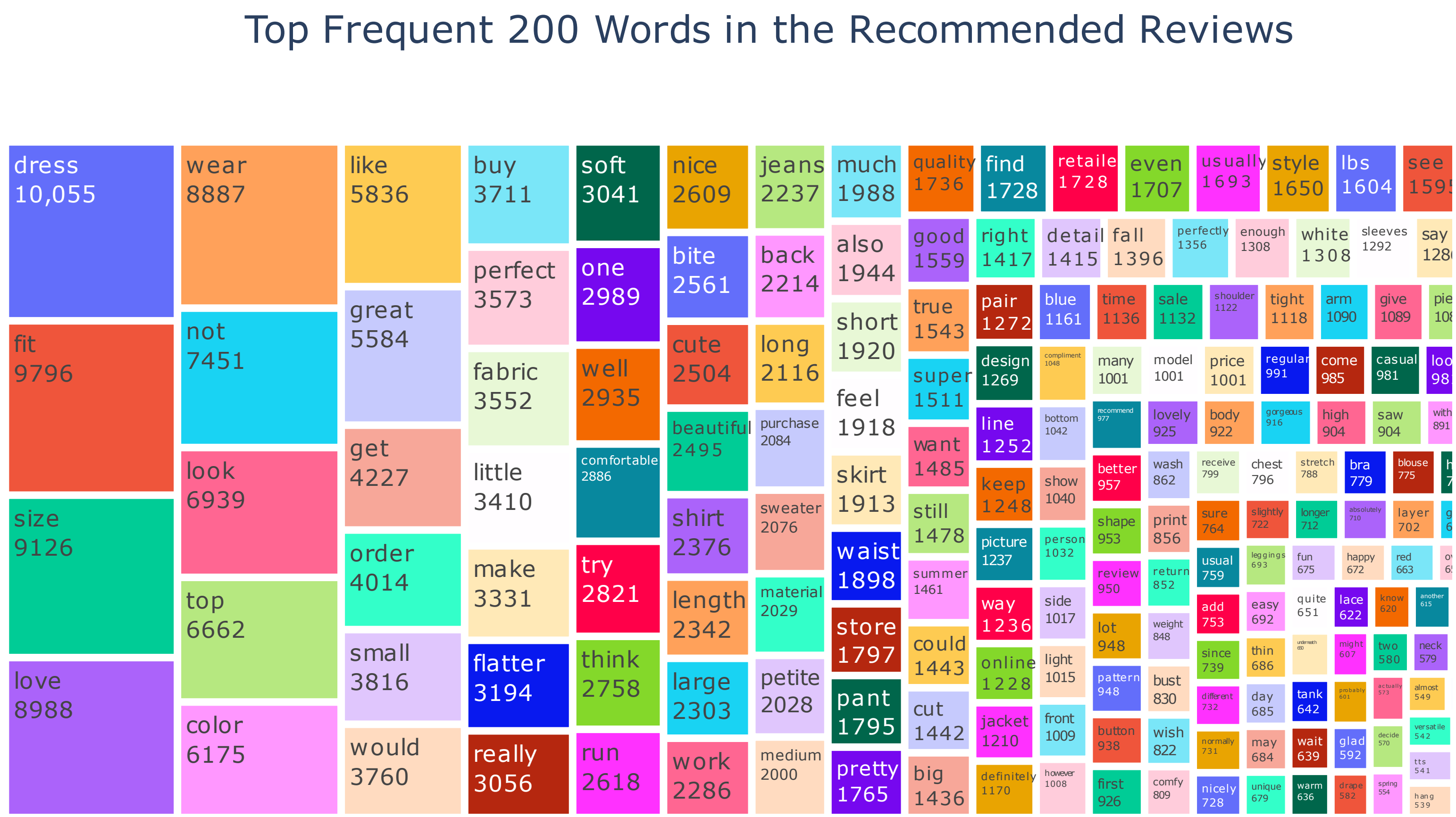
In [26]:

```
data_recommended = data[data['Recommended IND'] == 1]  # Dataframe that only includes recommended revie
data_not_recommended = data[data['Recommended IND'] == 0]  # # Dataframe that only includes not recomme

WordCloud_recommended = WordCloud(max_words=500,
                                  random_state=30,
                                  collocations=True).generate(str((data_recommended['final'])))

plt.figure(figsize=(15, 8))
plt.imshow(WordCloud_recommended, interpolation='bilinear')
plt.title('WordCloud of the Recommended Reviews', fontsize=20)
plt.axis("off")
plt.show()
```

WordCloud of the Recommended Reviews

In [27]:

```
FreqOfWords = data_recommended['final'].str.split(expand=True).stack().value_counts()
FreqOfWords_top200 = FreqOfWords[:200]

fig = px.treemap(FreqOfWords_top200, path=[FreqOfWords_top200.index], values=0)
fig.update_layout(title_text='Top Frequent 200 Words in the Recommended Reviews',
                  title_x=0.5, title_font=dict(size=20)
                  )
fig.update_traces(textinfo="label+value")
fig.show()
```



WordCloud of the Not Recommended Reviews

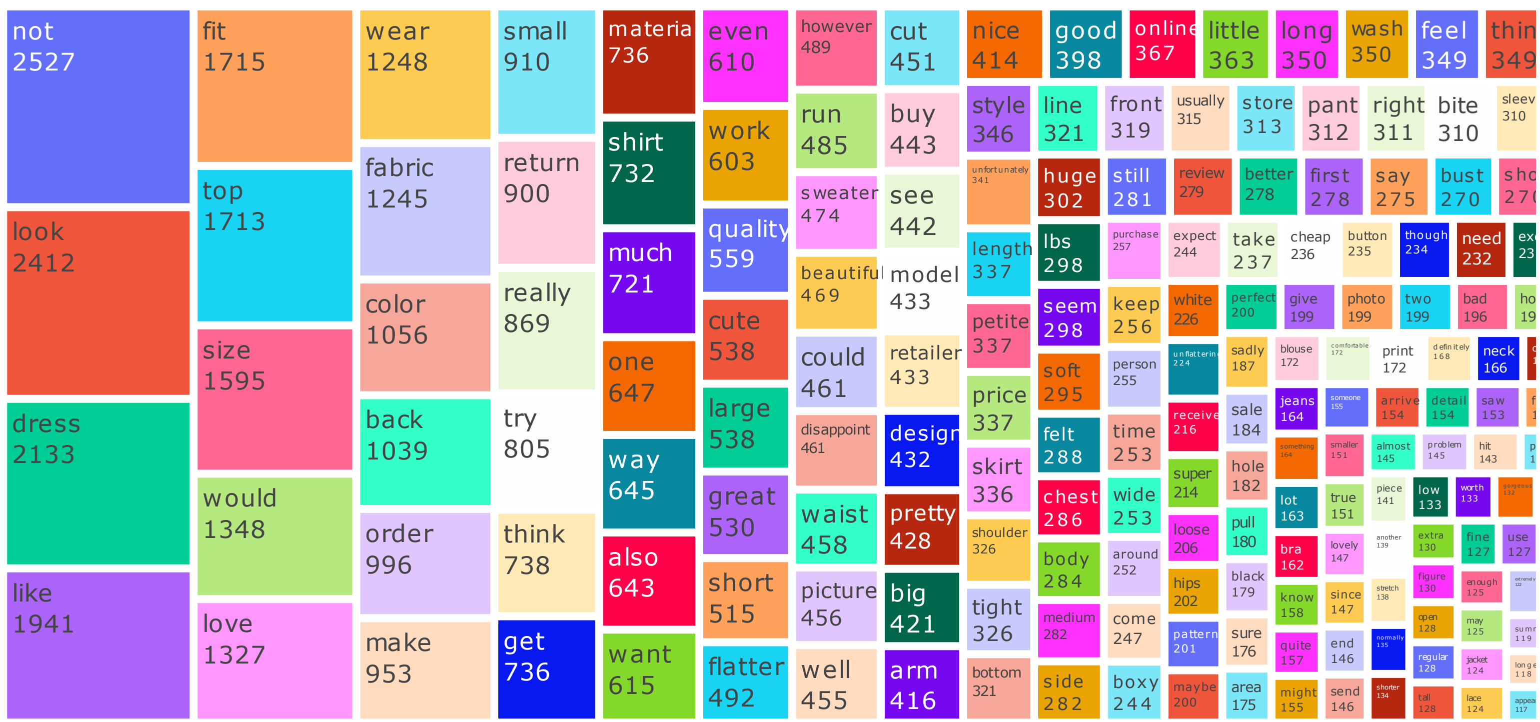
In [28]:

```
WordCloud_not_recommended = WordCloud(max_words=500,
                                      random_state=30,
                                      collocations=True).generate(str((data_not_recommended['final'])))

plt.figure(figsize=(15, 8))
plt.imshow(WordCloud_not_recommended, interpolation='bilinear')
plt.title('WordCloud of the Not Recommended Reviews', fontsize=20)
plt.axis("off")
plt.show()
```

WordCloud of the Not Recommended Reviews

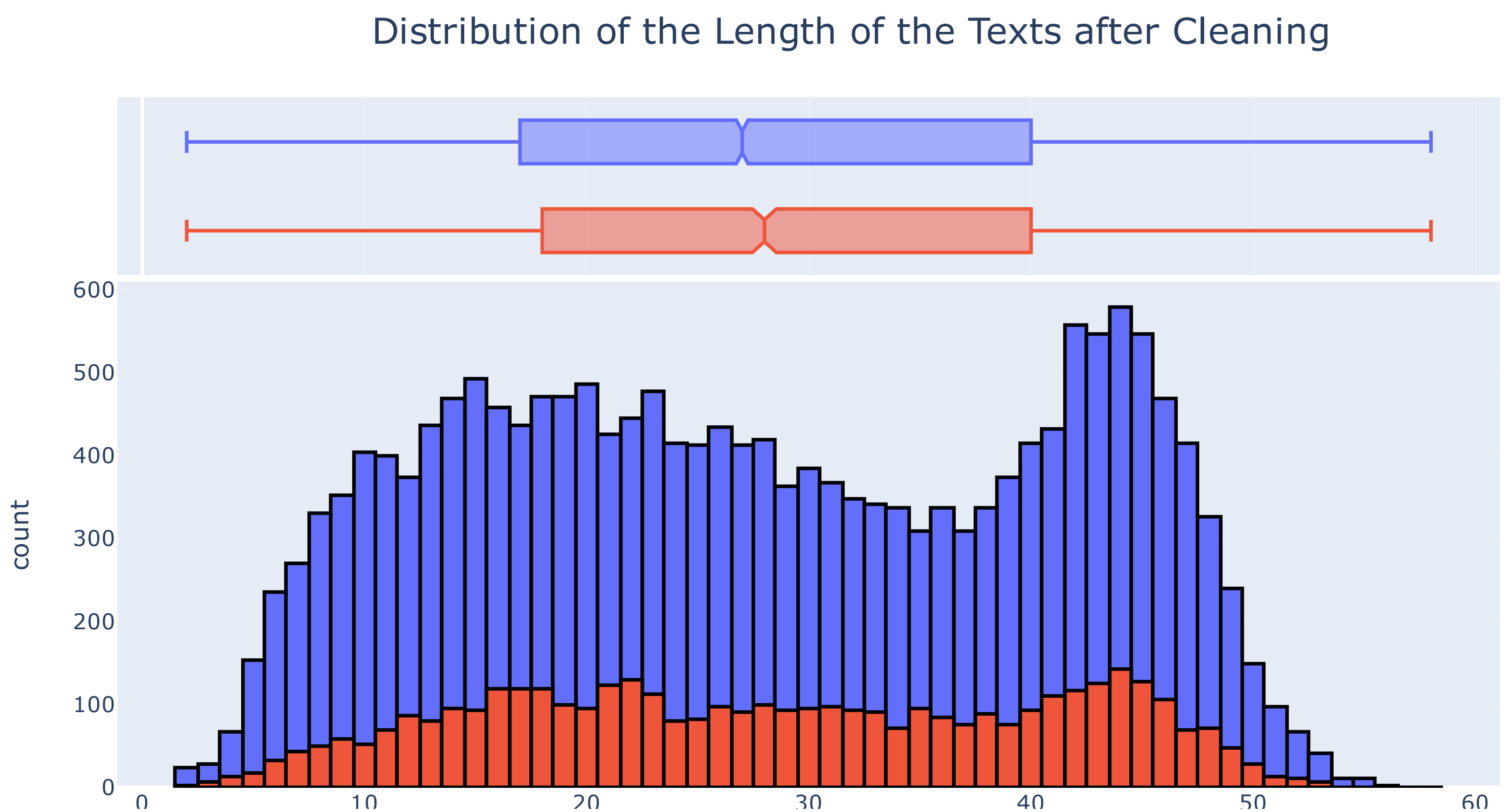
In [29]:
```
FreqOfWords = data_not_recommended['final'].str.split(expand=True).stack().value_counts()
FreqOfWords_top200 = FreqOfWords[:200]

fig = px.treemap(FreqOfWords_top200, path=[FreqOfWords_top200.index], values=0)
fig.update_layout(title_text='Top Frequent 200 Words in the Not Recommended Reviews',
                  title_x=0.5, title_font=dict(size=20)
                  )
fig.update_traces(textinfo="label+value")
fig.show()
```

Top Frequent 200 Words in the Not Recommended Reviews



Distribution of the Length of the Texts after Cleaning

In [30]:
```
data['length_of_text'] = [len(i.split(' ')) for i in data['final']]
fig = px.histogram(data['length_of_text'], marginal='box',
                   labels={"value": "Length of the Text",
                           "color": 'Recommended?'},
                   color=data['Recommended IND'])

fig.update_traces(marker=dict(line=dict(color='#000000', width=2)))
fig.update_layout(title_text='Distribution of the Length of the Texts after Cleaning',
                  title_x=0.5, title_font=dict(size=20))
fig.update_layout(barmode='overlay')
fig.show()
```



Train-Test-Validation Split

In [31]:
```
# I will only use Text data to predict Recommendation
y = data['Recommended IND']
X = data['final']

X.head()
```

Out[31]:
```
0          absolutely wonderful silky sexy comfortable
1    love dress sooo pretty happen find store glad ...
2    high hop dress really want work initially orde...
3    love love love jumpsuit fun flirty fabulous ev...
4    shirt flatter due adjustable front tie perfect...
Name: final, dtype: object
```

In [32]:
```
# Train-Test-Validation Split
x, X_test, y, y_test = train_test_split(X, y, test_size=0.2, random_state=13)  # Test: %20

X_train, X_val, y_train, y_val = train_test_split(x, y, test_size=0.25, random_state=13)  # Val: %20

print('Shape of the X_train:', X_train.shape)
print('Shape of the X_test:', X_test.shape)
print('Shape of the X_val:', X_val.shape)
print('--'*20)
print('Shape of the y_train:', y_train.shape)
print('Shape of the y_test:', y_test.shape)
print('Shape of the y_val:', y_val.shape)
```

```
Shape of the X_train: (13576,)
Shape of the X_test: (4526,)
Shape of the X_val: (4526,)
----------------------------------------
Shape of the y_train: (13576,)
Shape of the y_test: (4526,)
Shape of the y_val: (4526,)
```

Tokenizing with Tensorflow

In [33]:
```
num_words = 10000
tokenizer = Tokenizer(num_words=num_words, oov_token='<OOV>')
tokenizer.fit_on_texts(X_train)

Tokenized_train = tokenizer.texts_to_sequences(X_train)
Tokenized_val = tokenizer.texts_to_sequences(X_val)

print('Non-tokenized Version: ', X_train[0])
print('Tokenized Version: ', tokenizer.texts_to_sequences([X_train[0]]))
print('--'*20)
print('Non-tokenized Version: ', X_train[80])
print('Tokenized Version: ', tokenizer.texts_to_sequences([X_train[80]]))
```

```
Non-tokenized Version:  absolutely wonderful silky sexy comfortable
Tokenized Version:  [[161, 366, 748, 445, 33]]
----------------------------------------
Non-tokenized Version:  usually petite since dress not come petites try fit lbs dress hit knee hem bite no
t overwhelm dress look stun great vibrant color dark hair make classic elegant dress look contemporary sty
lish try store salesperson others happen see rave tell grab glad plan wear spring daughte
Tokenized Version:  [[61, 47, 150, 2, 7, 109, 769, 23, 3, 68, 2, 146, 269, 223, 38, 7, 746, 2, 8, 397, 12,
356, 11, 278, 1025, 18, 342, 459, 2, 8, 2151, 344, 23, 57, 1885, 350, 601, 62, 1723, 432, 631, 218, 313,
6, 212, 6315]]
```

Padding the Datasets

In [34]:
```
maxlen = 50
Padded_train = pad_sequences(Tokenized_train, maxlen=maxlen, padding='pre')
Padded_val = pad_sequences(Tokenized_val, maxlen=maxlen, padding='pre')
```

ANN Model Creation

In [35]:

```
# Creating the Model
model = Sequential()

model.add(Embedding(num_words, 16, input_length=maxlen))
model.add(Dropout(0.2))

model.add(GlobalAvgPool1D())
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

opt = tf.optimizers.Adam(lr=0.55e-3)  # Learning Rate

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

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, 50, 16)            160000    
_________________________________________________________________
dropout (Dropout)            (None, 50, 16)            0         
_________________________________________________________________
global_average_pooling1d (Gl (None, 16)                0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 16)                0         
_________________________________________________________________
dense (Dense)                (None, 1)                 17        
=================================================================
Total params: 160,017
Trainable params: 160,017
Non-trainable params: 0
_________________________________________________________________
```

In [36]:
```
# Training the Model
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', mode='auto', patience=5,
                                                  restore_best_weights=True)

epochs = 100
hist = model.fit(Padded_train, y_train, epochs=epochs,
                 validation_data=(Padded_val, y_val),
                 callbacks=[early_stopping], batch_size=32)
```

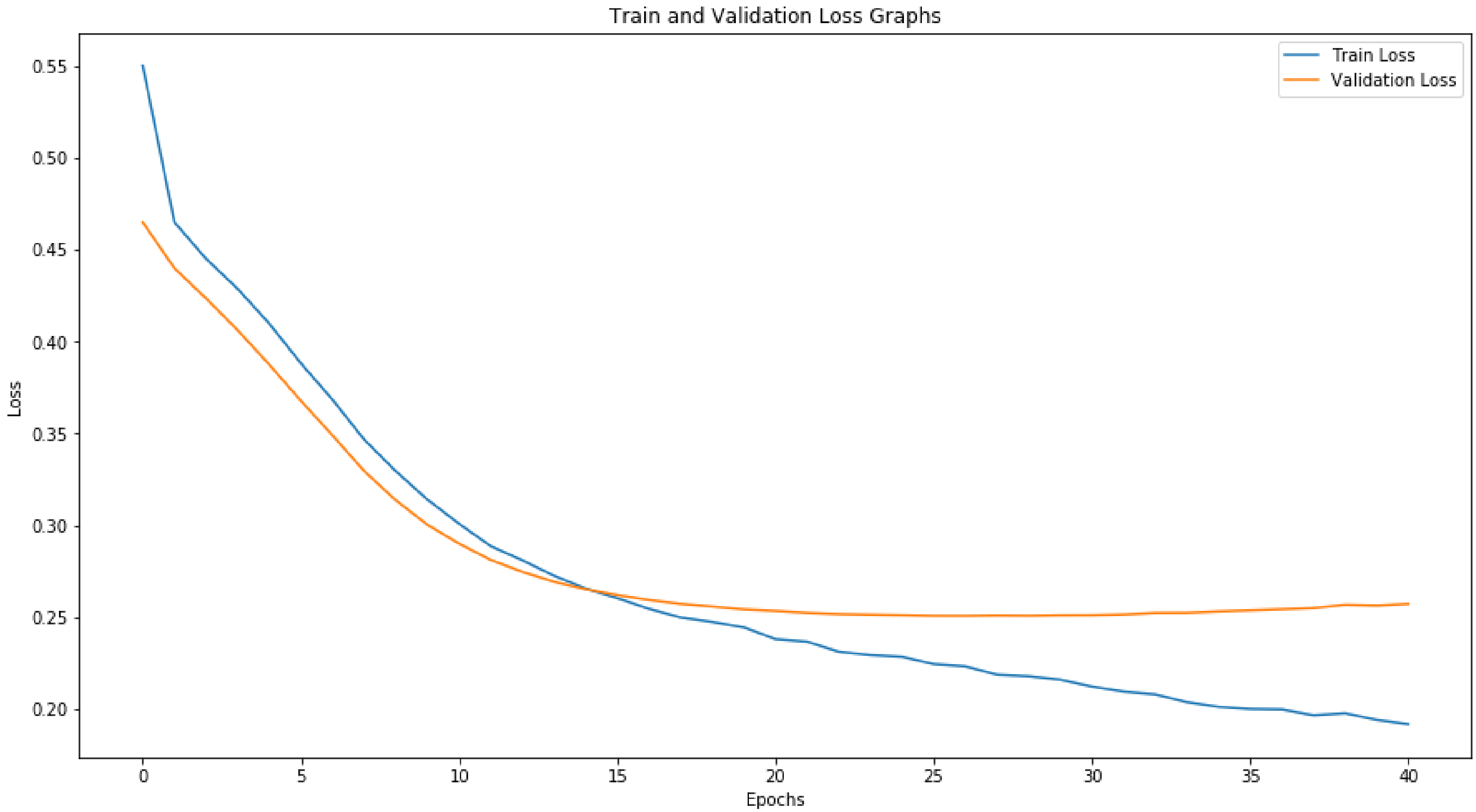
```
Epoch 1/100
425/425 [==============================] - 6s 7ms/step - loss: 0.6074 - accuracy: 0.8130 - val_loss: 0.
4649 - val_accuracy: 0.8239
Epoch 2/100
425/425 [==============================] - 2s 5ms/step - loss: 0.4773 - accuracy: 0.8132 - val_loss: 0.
4400 - val_accuracy: 0.8239
Epoch 3/100
425/425 [==============================] - 2s 5ms/step - loss: 0.4556 - accuracy: 0.8144 - val_loss: 0.
4235 - val_accuracy: 0.8239
Epoch 4/100
425/425 [==============================] - 2s 5ms/step - loss: 0.4340 - accuracy: 0.8194 - val_loss: 0.
4062 - val_accuracy: 0.8239
Epoch 5/100
425/425 [==============================] - 2s 5ms/step - loss: 0.4161 - accuracy: 0.8183 - val_loss: 0.
3875 - val_accuracy: 0.8243
Epoch 6/100
425/425 [==============================] - 2s 5ms/step - loss: 0.3985 - accuracy: 0.8161 - val_loss: 0.
3677 - val_accuracy: 0.8261
Epoch 7/100
```

Train and Validation Loss Graphs

In [38]:

```
plt.figure(figsize=(15, 8))
plt.plot(hist.history['loss'], label='Train Loss')
plt.plot(hist.history['val_loss'], label='Validation Loss')
plt.title('Train and Validation Loss Graphs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

Out[38]: <matplotlib.legend.Legend at 0x15f24dae448>



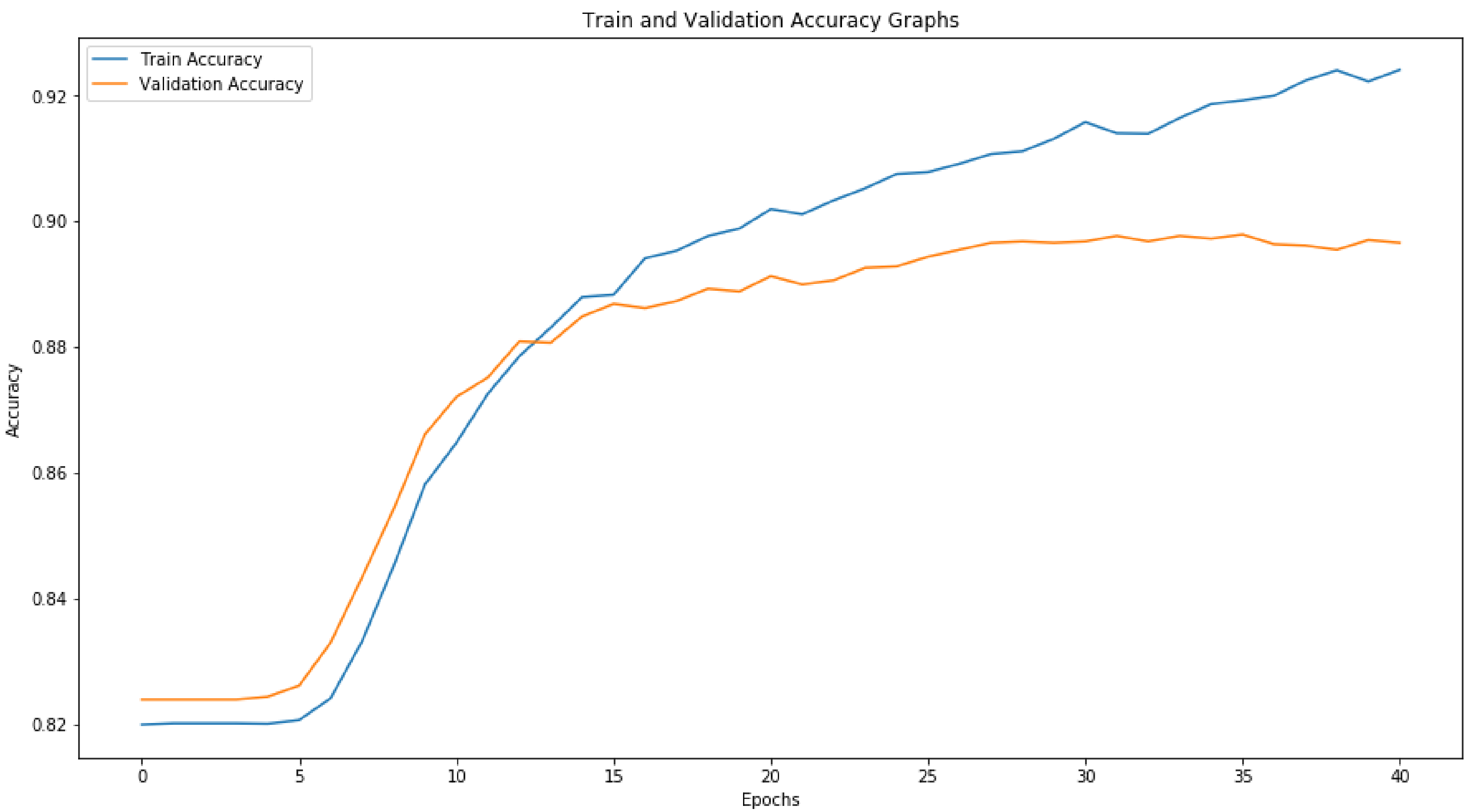
Train and Validation Accuracy Graphs

In [39]:

```
plt.figure(figsize=(15, 8))
plt.plot(hist.history['accuracy'], label='Train Accuracy')
plt.plot(hist.history['val_accuracy'], label='Validation Accuracy')
plt.title('Train and Validation Accuracy Graphs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

Out[39]:

```
<matplotlib.legend.Legend at 0x15f254c6ec8>
```



Preparing the Test Data

In [40]:
```
X_test = X_test.apply(tokenization)
X_test = X_test.apply(stopwords_remove)
X_test = X_test.apply(lemmatization)
X_test = X_test.str.join(' ')

X_test.head()
```

Out[40]:
```
10818                              low waisted weird liner
779      shirt not good look gal hips fit top tight ord...
10907    love dress long enough dramatic graze feet wit...
17442    understand pencil skirt gon body hug however r...
832      order shirt wear pair pant return one reason t...
Name: final, dtype: object
```

In [41]:
```
Tokenized_test = tokenizer.texts_to_sequences(X_test)
Padded_test = pad_sequences(Tokenized_test, maxlen=maxlen, padding='pre')

test_evaluate = model.evaluate(Padded_test, y_test)
```

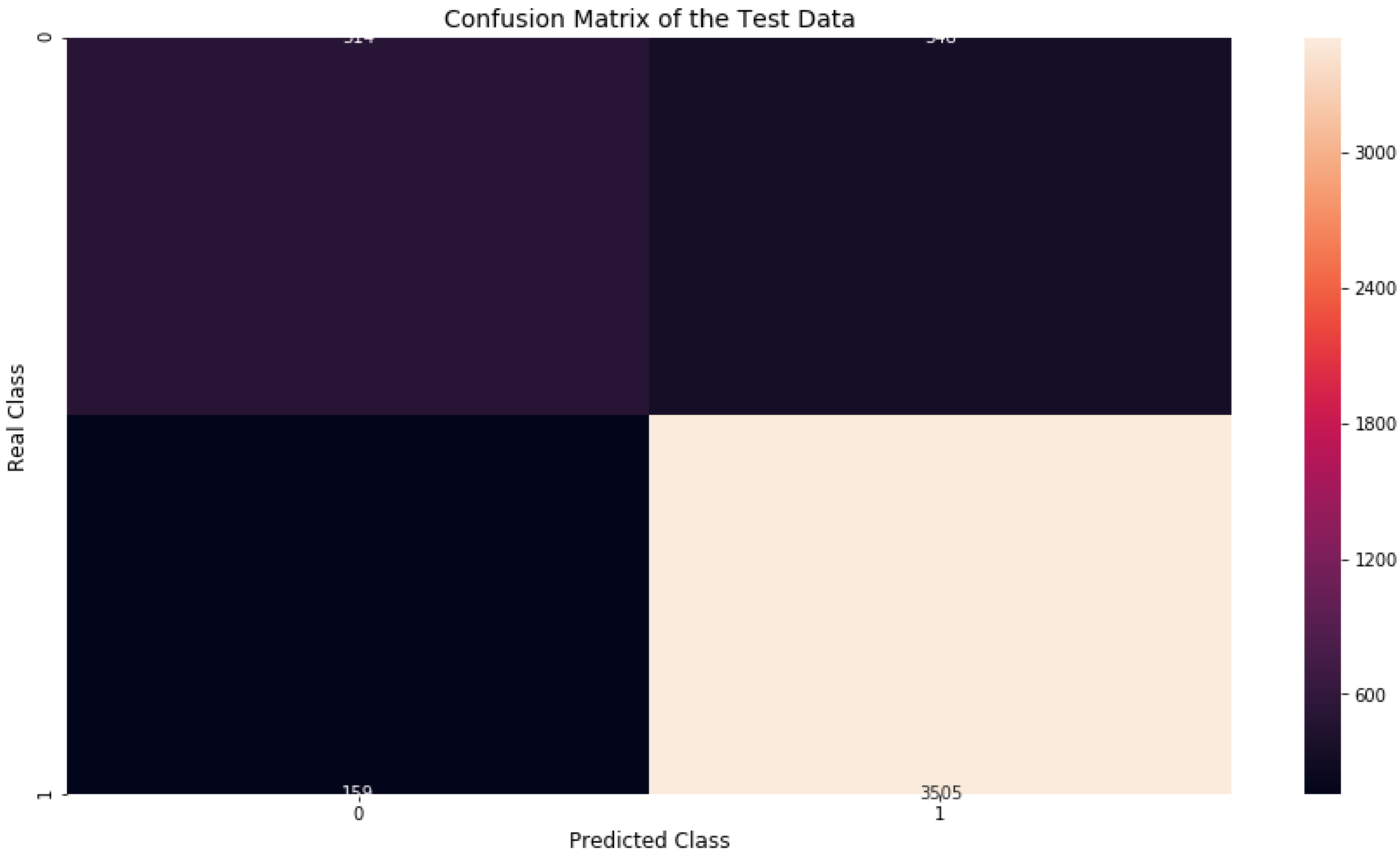
```
142/142 [==============================] - 0s 1ms/step - loss: 0.2700 - accuracy: 0.8889
```

Confusion Matrix of the Test Data

In [42]:
```
pred_train_lstm = model.predict(Padded_train)
pred_test_lstm = model.predict(Padded_test)
```

In [43]:

```
for i, x in enumerate(pred_test_lstm):
    if 0 <= x < 0.49:
        pred_test_lstm[i] = 0
    else:
        pred_test_lstm[i] = 1

for i, x in enumerate(pred_train_lstm):
    if 0 <= x < 0.49:
        pred_train_lstm[i] = 0
    else:
        pred_train_lstm[i] = 1

conf_mat = confusion_matrix(y_true=y_test, y_pred=pred_test_lstm)
plt.figure(figsize=(15, 8))
sns.heatmap(conf_mat, annot=True, fmt='g')
plt.title('Confusion Matrix of the Test Data', fontsize=14)
plt.ylabel('Real Class', fontsize=12)
plt.xlabel('Predicted Class', fontsize=12)
plt.show()
```



Evaluation Metrics of the LSTM Model

In [44]:
```
# Accuracy
train_acc_lstm = round(accuracy_score(y_train, pred_train_lstm) * 100, 2)
print('Train Accuracy of the LSTM: %', train_acc_lstm)
test_acc_lstm = round(accuracy_score(y_test, pred_test_lstm) * 100, 2)
print('Test Accuracy of the LSTM: %', test_acc_lstm)
print('--' * 20)

# Precision
train_precision_lstm = round(precision_score(y_train, pred_train_lstm) * 100, 2)
print('Train Precision of the LSTM: %', train_precision_lstm)
precision_lstm = round(precision_score(y_test, pred_test_lstm) * 100, 2)
print('Test Precision of the LSTM: %', precision_lstm)
print('--' * 20)

# Recall
train_recall_lstm = round(recall_score(y_train, pred_train_lstm) * 100, 2)
print('Train Recall of the LSTM: %', train_recall_lstm)
recall_lstm = round(recall_score(y_test, pred_test_lstm) * 100, 2)
print('Test Recall of the LSTM: %', recall_lstm)
```

```
Train Accuracy of the LSTM: % 92.35
Test Accuracy of the LSTM: % 88.8
----------------------------------------
Train Precision of the LSTM: % 93.94
Test Precision of the LSTM: % 90.97
----------------------------------------
Train Recall of the LSTM: % 96.93
Test Recall of the LSTM: % 95.66
```

Having Fun with the LSTM Model

In [45]:

```
def predict_recommendation(input_text):  # The function for doing all the previous steps
    input_text = input_text.lower()
    input_text = re.sub(r'[^a-zA-Z]', ' ', input_text)
    input_text = tokenization(input_text)
    input_text = stopwords_remove(input_text)
    input_text = lemmatization(input_text)
    input_text = ' '.join(input_text)
    input_text = tokenizer.texts_to_sequences([input_text])
    input_text = pad_sequences(input_text, maxlen=maxlen, padding='pre')
    input_text = model.predict(input_text)
    if input_text >= 0.5:
        input_text = f'Recommended with %{round(float(input_text*100), 2)}'
    else:
        input_text = f'Not Recommended with %{round(float(input_text*100), 2)}'

    return print(input_text)
```

In [46]:

```
# This reviews above are taken from several websites for testing the model with real world data. You ca
predict_recommendation("The clothes are such poor quality and look nothing like they do on the website.
```

```
Not Recommended with %16.57
```

In [47]:

```
predict_recommendation("Beautiful colour of lemon great fit and length here in three days all l need is
```

```
Recommended with %94.93
```

In [48]:

```
predict_recommendation("As usual the clothes I ordered arrived quickly and were all a good fit, except
```

```
Recommended with %83.63
```

In [49]:

```
predict_recommendation("I should've checked reviews before ordering... each item they sent was much wor
```

```
Not Recommended with %2.59
```

In [50]:
```
predict_recommendation("cheap material that falls apart in seconds. Clothes look nothing like the pictu
```

```
Not Recommended with %19.39
```

In [51]:
```
predict_recommendation("Very fast dispatch and delivery. Clothes are always a consistent fit, good qual
```

```
Recommended with %89.66
```

In [52]:
```
predict_recommendation("I have no complaints whatsoever, from ordering to getting my goods were excelle
```

```
Recommended with %94.01
```

In [53]:
```
predict_recommendation("My dress had blue ink and biro stains on which was a real shame. I needed it fo
```

```
Not Recommended with %41.35
```

In [54]:
```
# Ref. 6 from now on
predict_recommendation("Sizes varied despite allegedly being the same size. Some of the quality was poo
```

```
Not Recommended with %14.46
```

In [55]:
```
predict_recommendation("I do really like yours clothing, just find the sizing is slightly off, a normal
```

```
Recommended with %89.15
```

In [56]:
```
predict_recommendation("I really love this dress. I ordered a large and it fits perfectly. There’s abou
```

```
Recommended with %99.85
```

In [57]:
```
predict_recommendation("I don't like writing negative reviews but this one pissed me off the second I p
```

Not Recommended with %2.33

In [58]:
```
predict_recommendation("The cheapest material I've ever seen. It was like someone wove paper napkins fr
```

Not Recommended with %16.9

In [59]:
```
# Ref. 7 from now on
predict_recommendation("I was so excited to receive this dress in the mail! The first day I wore it, I
```

Recommended with %91.75

In [60]:
```
predict_recommendation("The dress does not look like the dress pictures. The material seems cheaper and
```

Not Recommended with %2.57

In [61]:
```
predict_recommendation("I love this item it's was not dark blue like the picture but i love i it's very
```

Recommended with %73.58

In [62]:
```
predict_recommendation("This kaftan is NOT a silky material at all, it is a slightly transparent and du
```

Not Recommended with %4.94

In [ ]:
```

```