# *TECHNICAL REPORTS* – RECOMMENDATION SYSTEM

Disusun untuk memenuhi tugas UAS mata kuliah Komputasi Lanjut dan Pengelolaan Data SCMA801007

Dosen : Dr. Risman Adnan, S.Si., M.Si.

Muhammad Ridho (2206130776)

PROGRAM MAGISTER MATEMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS INDONESIA

DEPOK

2023

## Contents

1. **Introduction** ........ 3
2. **Background** ........ 3
   - 2.1 **Traditional Drug Prescribing Practices** ........ 3
   - 2.2 **Limitations of Traditional Approaches** ........ 4
   - 2.3 **Role of Technology and Data-driven Approaches** ........ 4
   - 2.4 **Existing Drug Recommendation Systems** ........ 4
3. **Data Collection and Preprocessing** ........ 5
   - 3.1 **Data Collection** ........ 5
   - 3.2 **Exploration Data Analysis** ........ 6
   - 3.3 **Date Preprocessing** ........ 23
4. **Models** ........ 28
   - 4.1 **Collaborative Filtering** ........ 28
   - 4.2 **Deep Learning Model Using N-gram** ........ 38
   - 4.3 **Lightgbm** ........ 42
5. **Conclusion** ........ 51

## 1. Introduction

In modern healthcare, the process of drug prescribing is a complex and critical task that heavily influences patient outcomes. Physicians face the challenge of navigating an ever-expanding array of pharmaceutical options, each with its own unique efficacy, safety profile, and patient-specific considerations. The consequences of incorrect or suboptimal drug prescriptions can be severe, leading to adverse reactions, treatment inefficacy, increased healthcare costs, and even patient harm.

To address these challenges, drug recommendation systems have emerged as a promising solution. These systems leverage the power of technology and data-driven approaches to assist healthcare professionals in making informed and personalized drug prescribing decisions. By analyzing vast amounts of patient data, clinical guidelines, drug databases, and other relevant sources, these systems can provide evidence-based recommendations tailored to individual patients' needs, medical histories, and demographic factors.

The primary objective of this technical report is to explore the landscape of drug recommendation systems, examine their underlying algorithms, and evaluate their effectiveness in improving prescribing practices. We aim to provide a comprehensive overview of the key components, methodologies, and challenges involved in building and implementing such systems. Additionally, we seek to highlight the potential benefits of drug recommendation systems in enhancing patient safety, optimizing treatment outcomes, and reducing healthcare costs.

By delving into the intricacies of drug recommendation systems, we hope to contribute to the broader understanding of this innovative approach to healthcare decision-making. Through critical analysis and evaluation, we aim to identify the strengths and limitations of existing systems, explore areas for improvement, and propose future research directions.

In the subsequent sections of this report, we will provide a detailed examination of the background, data collection and preprocessing methods, recommendation algorithms, evaluation metrics, implementation details, experimental results, and potential future directions for drug recommendation systems. By shedding light on this rapidly evolving field, we aspire to pave the way for advancements that have the potential to revolutionize the practice of drug prescribing and improve patient care.

## 2. Background

### 2.1 Traditional Drug Prescribing Practices

Traditional drug prescribing practices have long relied on the expertise and experience of healthcare professionals, such as physicians, pharmacists, and specialists. While this approach has been effective to some extent, it is not without limitations. The vast and constantly expanding knowledge base in pharmacology, along with the increasing number of available drugs and their interactions, makes it challenging for healthcare professionals to stay up-to-date and make optimal prescribing decisions solely based on their individual expertise.

Furthermore, the variability in patients' characteristics, medical histories, and medication tolerances introduces a level of complexity that demands personalized and evidence-based prescriptions. Traditional prescribing practices often struggle to account for the nuances of each patient's condition and the evolving landscape of drug efficacy and safety profiles.

## 2.2 Limitations of Traditional Approaches

Several limitations of traditional drug prescribing approaches have been identified, highlighting the need for more advanced methods:

- Limited access to comprehensive patient data: Healthcare professionals often have limited access to a patient's complete medical history, including past diagnoses, treatments, and medication usage. This lack of comprehensive data can hinder their ability to make informed prescribing decisions tailored to the patient's specific needs and potential contraindications.
- Time and resource constraints: Healthcare professionals operate in a fast-paced environment, with limited time for in-depth research on every patient's condition and available drug options. This time constraint can lead to suboptimal or inadequate prescribing decisions.
- Human error and cognitive biases: Even experienced healthcare professionals are susceptible to cognitive biases and errors, which can affect their prescribing decisions. Factors such as fatigue, workload, and individual biases may impact the accuracy and consistency of drug prescriptions.

## 2.3 Role of Technology and Data-driven Approaches

The integration of technology and data-driven approaches has the potential to revolutionize drug prescribing practices. By leveraging large-scale data collection, storage, and analysis, technology can support healthcare professionals in making evidence-based, personalized prescribing decisions. Data sources such as electronic health records (EHRs), clinical trials, genetic information, and drug databases can provide valuable insights into patient characteristics, treatment outcomes, and drug effectiveness.

Furthermore, advancements in computational techniques, machine learning, and artificial intelligence have opened up new avenues for developing sophisticated drug recommendation systems. These systems can analyze vast amounts of data, identify patterns, and generate personalized recommendations for specific patients based on their medical history, demographics, and other relevant factors.

## 2.4 Existing Drug Recommendation Systems

Over the years, several drug recommendation systems have been developed and deployed in various healthcare settings. These systems utilize diverse methodologies, ranging from collaborative filtering and content-based filtering to knowledge-based systems and hybrid approaches. Collaborative filtering leverages the collective wisdom of a group of similar patients to make recommendations, while content-based filtering focuses on the similarity between drugs and patients' profiles. Knowledge-based systems utilize expert knowledge and guidelines, and hybrid approaches combine multiple techniques to enhance recommendation accuracy and coverage.

These existing drug recommendation systems have demonstrated promising results in improving prescribing practices, enhancing patient safety, and optimizing treatment outcomes. However, there are still challenges to overcome, such as the need for better integration with healthcare workflows, ensuring data privacy and security, and addressing the interpretability of recommendation results.

## 3. Data Collection and Preprocessing

### 3.1 Data Collection

We obtained the Drug dataset from the https://www.kaggle.com/datasets/jessicali9530/kuc-hackathon-winter-2018. This dataset contains 160.684 observations of drugs and each observation has 7 features describing the characteristics of drugs. First we will import Train data and Test data. The sizes of the two data are as follows: It was data from https://archive.ics.uci.edu/ml/datasets/Drug+Review+Dataset+%28Drugs.com%29 and crawled reviews from online pharmaceutical review sites. Here is the codes.

```
import pandas as pd #Analysis
import matplotlib.pyplot as plt #Visulization
import seaborn as sns #Visulization
import numpy as np #Analysis
from scipy.stats import norm #Analysis
from sklearn.preprocessing import StandardScaler #Analysis
from scipy import stats #Analysis
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
import gc

import os
import string
color = sns.color_palette()

%matplotlib inline

from plotly import tools
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go

from sklearn import model_selection, preprocessing, metrics, ensemble,
naive_bayes, linear_model
from sklearn.feature_extraction.text import TfidfVectorizer,
CountVectorizer
from sklearn.decomposition import TruncatedSVD
import lightgbm as lgb

pd.options.mode.chained_assignment = None
pd.options.display.max_columns = 999
```

```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)


from google.colab import drive
drive.mount('/content/gdrive')
```

```
# preprocessing the data file
#  read the data
df1 = pd.read_csv("/content/gdrive/My Drive/UAS Data
Mining/drugsComTrain_raw.csv", parse_dates=['date'])
df2 = pd.read_csv("/content/gdrive/My Drive/UAS Data
Mining/drugsComTest_raw.csv", parse_dates=['date'])
```

## 3.2 Exploration Data Analysis

We performed some EDA steps to ensure the data was in a suitable format. Here is the codes.

```
print("Train shape :" ,df_train.shape)
print("Test shape :", df_test.shape)
Train shape : (161297, 7)
Test shape : (53766, 7)
```

This is the result of looking at the data through the head () command. There are six variables except for the unique ID that identifies the individual, and review is the key variable.

```
df_train.head()
```

| | uniqueID | drugName | condition | review | rating | date | usefulCount |
|---|---|---|---|---|---|---|---|
| 0 | 206461 | Valsartan | Left Ventricular Dysfunction | "It has no side effect, I take it in combinati... | 9 | 2012-05-20 | 27 |
| 1 | 95260 | Guanfacine | ADHD | "My son is halfway through his fourth week of ... | 8 | 2010-04-27 | 192 |
| 2 | 92703 | Lybrel | Birth Control | "I used to take another oral contraceptive, wh... | 5 | 2009-12-14 | 17 |
| 3 | 138000 | Ortho Evra | Birth Control | "This is my first time using any form of birth... | 8 | 2015-11-03 | 10 |
| 4 | 35696 | Buprenorphine / naloxone | Opiate Dependence | "Suboxone has completely turned my life around... | 9 | 2016-11-27 | 37 |

These are additional explanations for variables.

1. drugName (categorical): name of drug
2. condition (categorical): name of condition
3. review (text): patient review
4. rating (numerical): 10 star patient rating

5. date (date): date of review entry
6. usefulCount (numerical): number of users who found review useful

The structure of the data is that a patient with a unique ID purchases a drug that meets his condition and writes a review and rating for the drug he/she purchased on the date. Afterwards, if the others read that review and find it helpful, they will click usefulCount, which will add 1 for the variable.

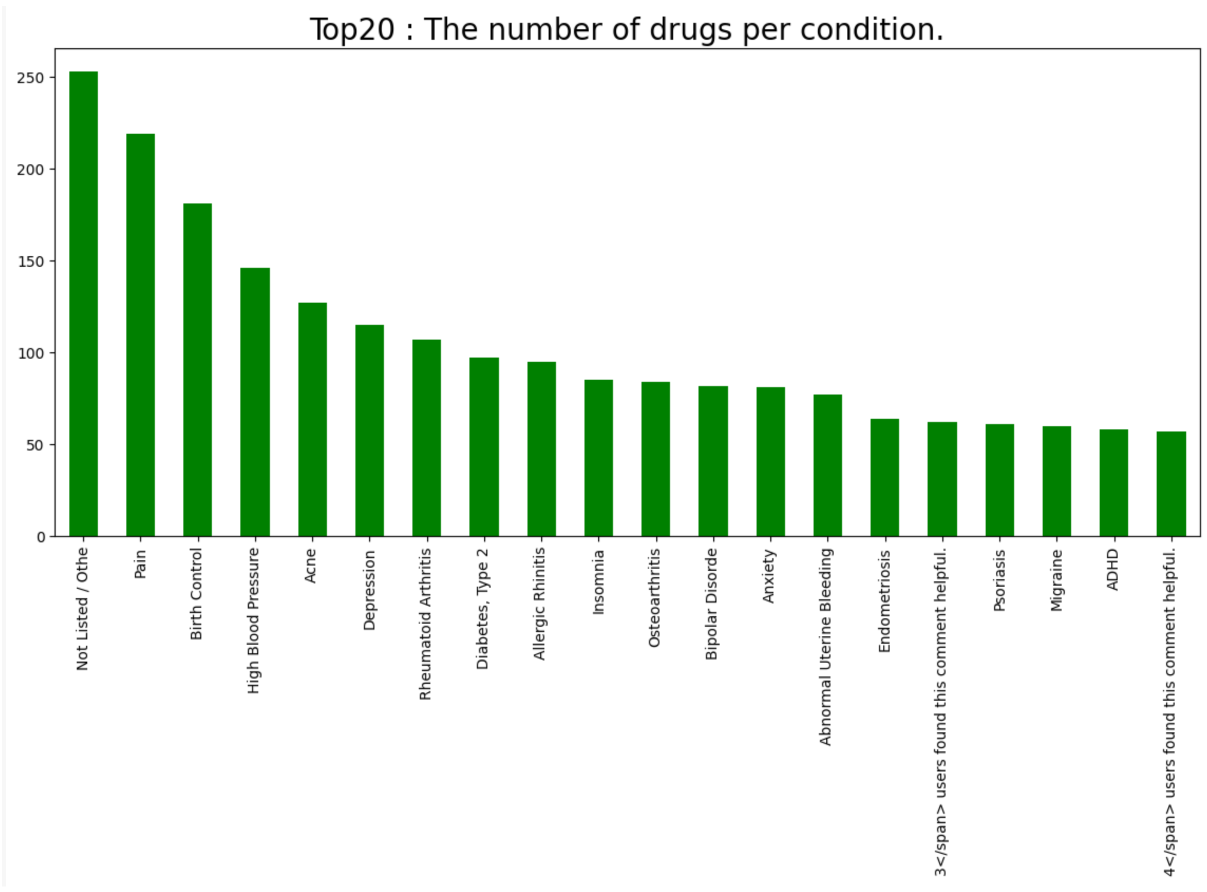
First, we will start exploring variables, starting from uniqueID. We compared the unique number of unique IDs and the length of the train data to see if the same customer has written multiple reviews, and there weren't more than one reviews for one customer.

```
print("unique values count of train : "
,len(set(df_train['uniqueID'].values)))
print("length of train : " ,df_train.shape[0])
```

```
unique values count of train :  161297
length of train :  161297
```

DrugName is closely related to condition, so we have analyzed them together. The unique values of the two variables are 3671 and 917, respectively, and there are about 4 drugs for each condition. Let's go ahead and visualize this in more detail.

```
df_all = pd.concat([df_train,df_test])
condition_dn =
df_all.groupby(['condition'])['drugName'].nunique().sort_values(as
cending=False)
condition_dn[0:20].plot(kind="bar", figsize = (14,6), fontsize =
10,color="green")
plt.xlabel("", fontsize = 20)
plt.ylabel("", fontsize = 20)
plt.title("Top20 : The number of drugs per condition.", fontsize =
20)
```



As you can see from the picture above, the number of drugs for top eight conditions is about 100 for each condition. On the other hand, it should be noted that the phrase "3</span> users found this comment helpful" appears in the condition, which seems like an error in the crawling process. I have looked into it to see in more details.

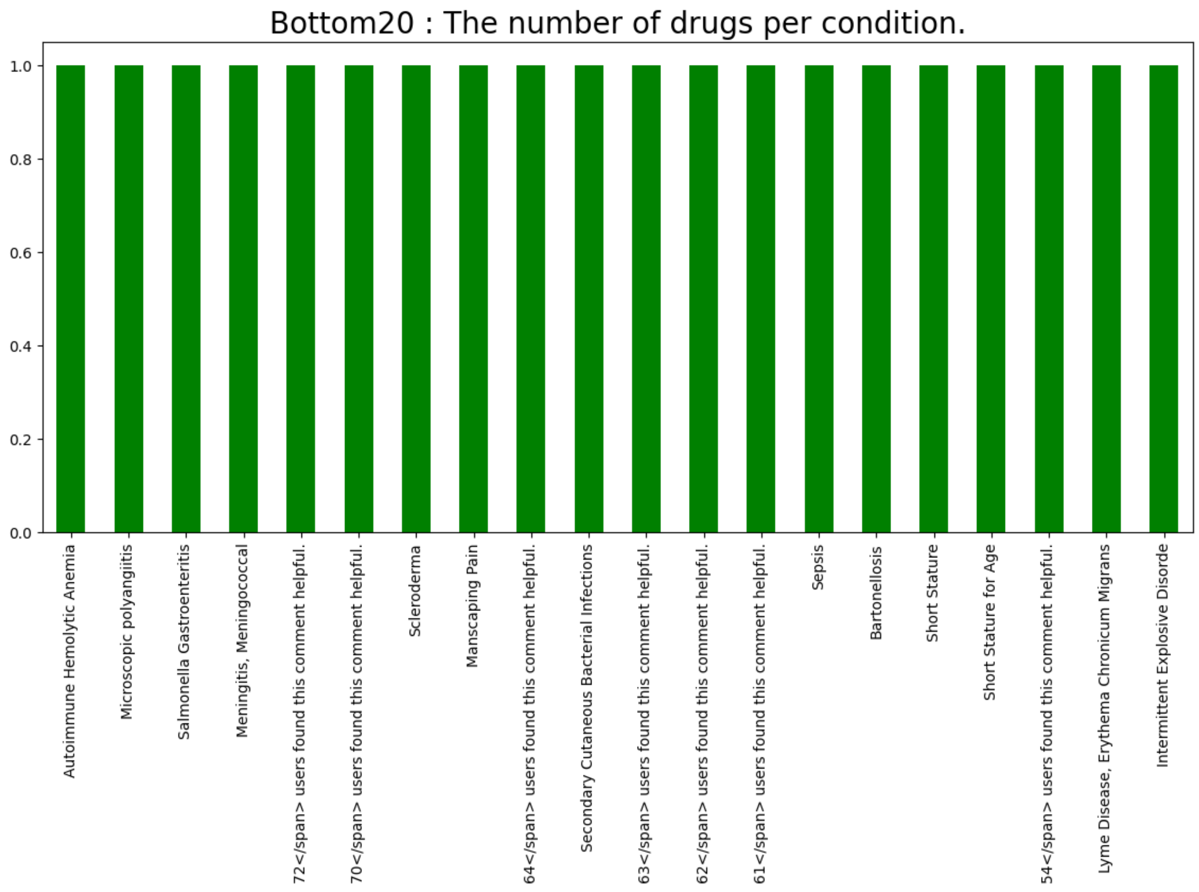
```
df_all[df_all['condition']=='3</span> users found this comment 
helpful.'].head(3)
```

| | uniqueID | drugName | condition | review | rating | date | usefulCount |
|---|---|---|---|---|---|---|---|
| **243** | 81588 | Yaz | 3</span> users found this comment helpful. | "I took Yaz for a little over 2 years. From a... | 3 | 2010-06-01 | 3 |
| **1864** | 124318 | Skyla | 3</span> users found this comment helpful. | "Never pregnant,28,retroverted small (6cm) ute... | 1 | 2015-12-16 | 3 |
| **3322** | 202848 | ProAir HFA | 3</span> users found this comment helpful. | "I get chest colds and asthmatic symptoms in t... | 9 | 2015-12-12 | 3 |

It is expected that for structure of '</ span> users found this comment helpful.' phrase, there will be not only 3, but also 4 as shown above, and other numbers as well. We will remove these data in the future preprocessing.

The following are the low 20 conditions of 'drugs per condition'. As you can see, the number is all 1. Considering the recommendation system, it is not feasible to recommend with that when there is only one product. Therefore, we will analyze only the conditions that have at least 2 drugs per condition.

```
condition_dn =
df_all.groupby(['condition'])['drugName'].nunique().sort_values(as
cending=False)

condition_dn[condition_dn.shape[0]-
20:condition_dn.shape[0]].plot(kind="bar", figsize = (14,6),
fontsize = 10,color="green")
plt.xlabel("", fontsize = 20)
plt.ylabel("", fontsize = 20)
plt.title("Bottom20 : The number of drugs per condition.",
fontsize = 20)
```



Next, let's have a look at the review. First, noticeable parts are the html strings like \ r \ n, and the parts that express emotions in parentheses such as (very unusual for him) and (a good thing) and words in capital letters like MUCH.

```
df_train['review'][1]
```

```
'"My son is halfway through his fourth week of Intuniv. We became concerned when he began this last week, when he started taking
the highest dose he will be on. For two days, he could hardly get out of bed, was very cranky, and slept for nearly 8 hours on a
drive home from school vacation (very unusual for him.) I called his doctor on Monday morning and she said to stick it out a few
days. See how he did at school, and with getting up in the morning. The last two days have been problem free. He is MUCH more agr
eeable than ever. He is less emotional (a good thing), less cranky. He is remembering all the things he should. Overall his behav
ior is better. \r\nWe have tried many different medications and so far this is the most effective."'
```

In addition, there were some words with errors like didn&# 039;t for didn't, and also characters like ...

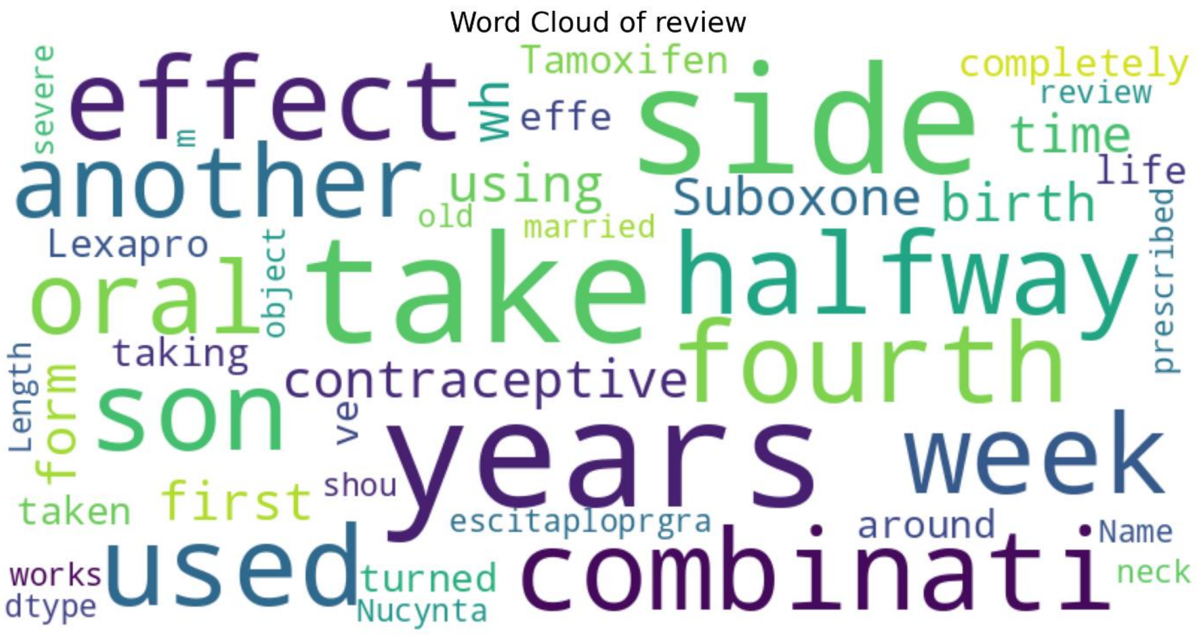
```
df_train['review'][2]
```

```
'"I used to take another oral contraceptive, which had 21 pill cycle, and was very happy- very light periods, max 5 days, no othe
r side effects. But it contained hormone gestodene, which is not available in US, so I switched to Lybrel, because the ingredient
s are similar. When my other pills ended, I started Lybrel immediately, on my first day of period, as the instructions said. And
the period lasted for two weeks. When taking the second pack- same two weeks. And now, with third pack things got even worse- my
third period lasted for two weeks and now it&#039;s the end of the third week- I still have daily brown discharge.\r\nThe positiv
e side is that I didn&#039;t have any other side effects. The idea of being period free was so tempting... Alas."'
```

We will delete these parts in preprocessing as well.

Next up, it's Word Cloud.

```
#https://www.kaggle.com/sudalairajkumar/simple-exploration-
notebook-qiqc kernel
from wordcloud import WordCloud, STOPWORDS

# Thanks : https://www.kaggle.com/aashita/word-clouds-of-various-
shapes ##
def plot_wordcloud(text, mask=None, max_words=200,
max_font_size=100, figure_size=(24.0,16.0),
                   title = None, title_size=40,
image_color=False):
    stopwords = set(STOPWORDS)
    more_stopwords = {'one', 'br', 'Po', 'th', 'sayi', 'fo',
'Unknown'}
    stopwords = stopwords.union(more_stopwords)

    wordcloud = WordCloud(background_color='white',
                    stopwords = stopwords,
                    max_words = max_words,
                    max_font_size = max_font_size,
                    random_state = 42,
                    width=800,
                    height=400,
                    mask = mask)
    wordcloud.generate(str(text))

    plt.figure(figsize=figure_size)
    if image_color:
        image_colors = ImageColorGenerator(mask);
        plt.imshow(wordcloud.recolor(color_func=image_colors),
interpolation="bilinear");
        plt.title(title, fontdict={'size': title_size,
                                  'verticalalignment': 'bottom'})
    else:
        plt.imshow(wordcloud);
        plt.title(title, fontdict={'size': title_size, 'color':
'black',
                                  'verticalalignment': 'bottom'})
    plt.axis('off');
    plt.tight_layout()

plot_wordcloud(df_all["review"], title="Word Cloud of review")
```

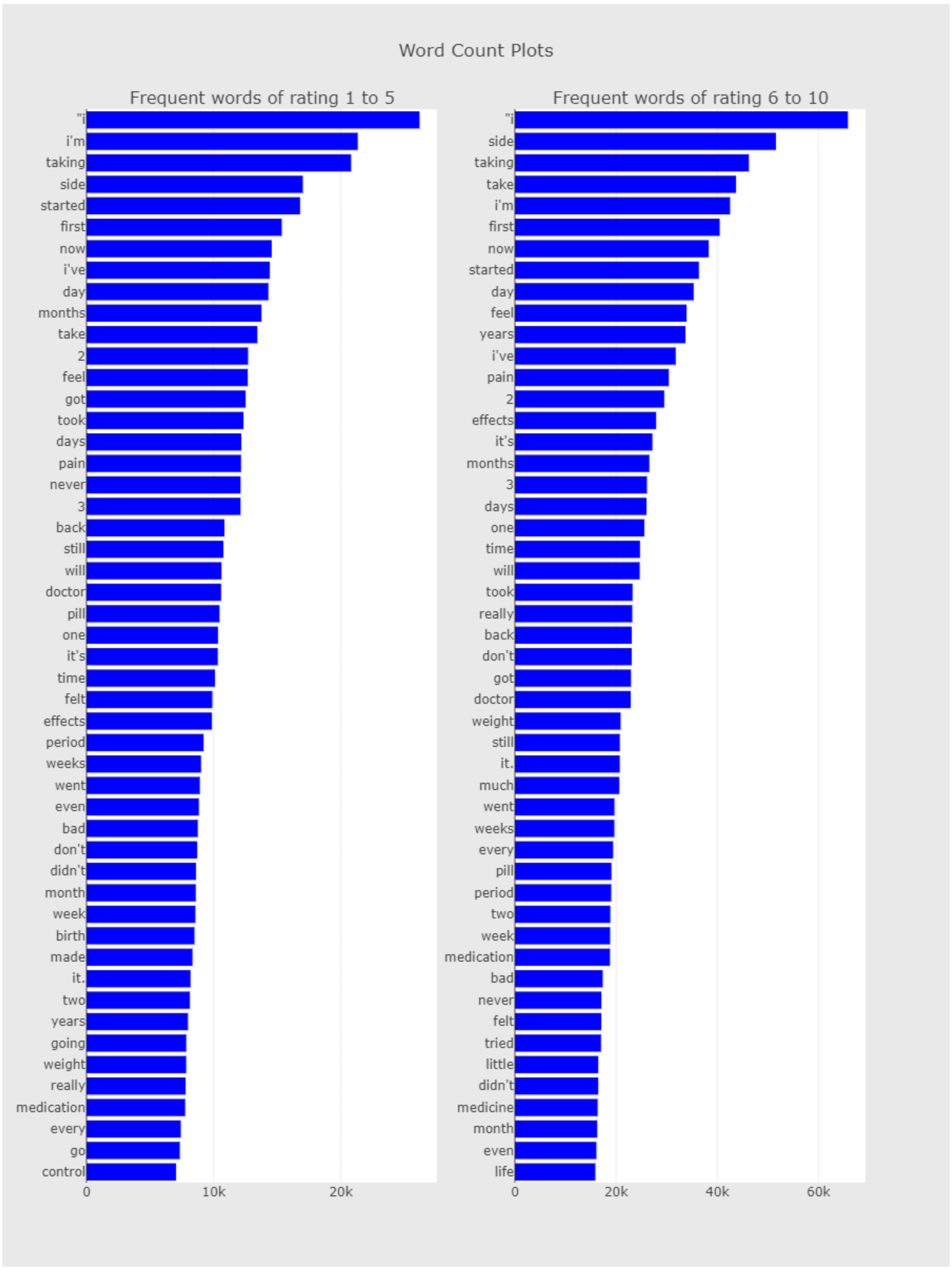


Next, we will classify 1 ~ 5 as negative, and 6 ~ 10 as positive, and we will check through 1 ~ 4 grams which corpus best classifies emotions.

```
from collections import defaultdict
df_all_6_10 = df_all[df_all["rating"]>5]
df_all_1_5 = df_all[df_all["rating"]<6]
```

```
## custom function for ngram generation ##
def generate_ngrams(text, n_gram=1):
    token = [token for token in text.lower().split(" ") if token
!= "" if token not in STOPWORDS]
    ngrams = zip(*[token[i:] for i in range(n_gram)])
    return [" ".join(ngram) for ngram in ngrams]

## custom function for horizontal bar chart ##
def horizontal_bar_chart(df, color):
    trace = go.Bar(
        y=df["word"].values[::-1],
        x=df["wordcount"].values[::-1],
        showlegend=False,
        orientation = 'h',
        marker=dict(
            color=color,
        ),
    )
    return trace
```

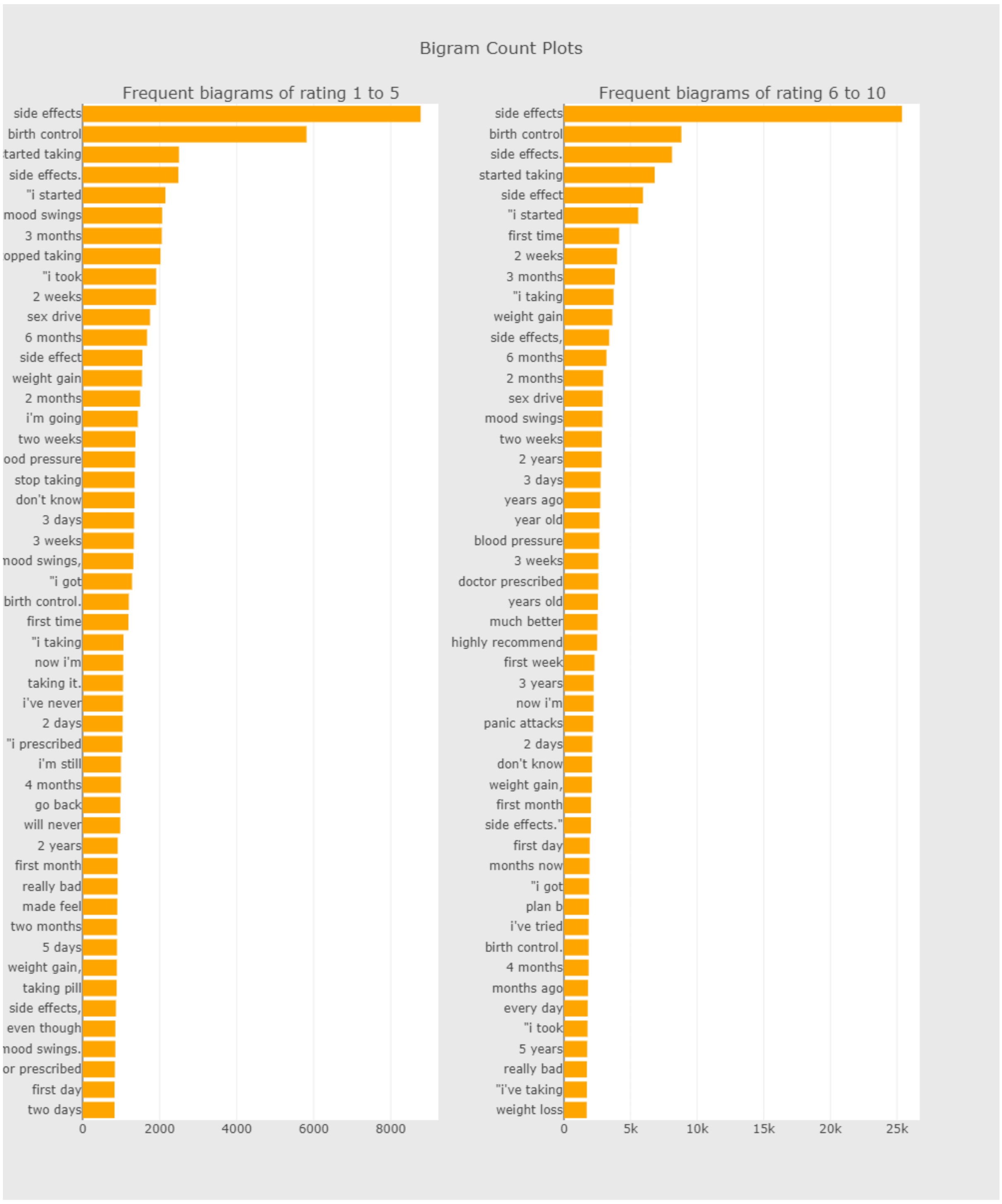
```
## Get the bar chart from rating  8 to 10 review ##
freq_dict = defaultdict(int)
for sent in df_all_1_5["review"]:
    for word in generate_ngrams(sent):
        freq_dict[word] += 1
fd_sorted = pd.DataFrame(sorted(freq_dict.items(), key=lambda x:
x[1])[::-1])
fd_sorted.columns = ["word", "wordcount"]
trace0 = horizontal_bar_chart(fd_sorted.head(50), 'blue')

## Get the bar chart from rating  4 to 7 review ##
freq_dict = defaultdict(int)
for sent in df_all_6_10["review"]:
    for word in generate_ngrams(sent):
        freq_dict[word] += 1
fd_sorted = pd.DataFrame(sorted(freq_dict.items(), key=lambda x:
x[1])[::-1])
fd_sorted.columns = ["word", "wordcount"]
trace1 = horizontal_bar_chart(fd_sorted.head(50), 'blue')

# Creating two subplots
fig = tools.make_subplots(rows=1, cols=2, vertical_spacing=0.04,
                          subplot_titles=["Frequent words of
rating 1 to 5",
                                          "Frequent words of
rating 6 to 10"])
fig.append_trace(trace0, 1, 1)
fig.append_trace(trace1, 1, 2)
fig['layout'].update(height=1200, width=900,
paper_bgcolor='rgb(233,233,233)', title="Word Count Plots")
py.iplot(fig, filename='word-plots')
```



When you use 1-gram, you can see that the top 5 words have the same contents, although the order of left (negative) and right (positive) are different. This means when we analyze the text with a single corpus, it does not classify the emotion well. So, we will expand the corpus.

```
freq_dict = defaultdict(int)
```

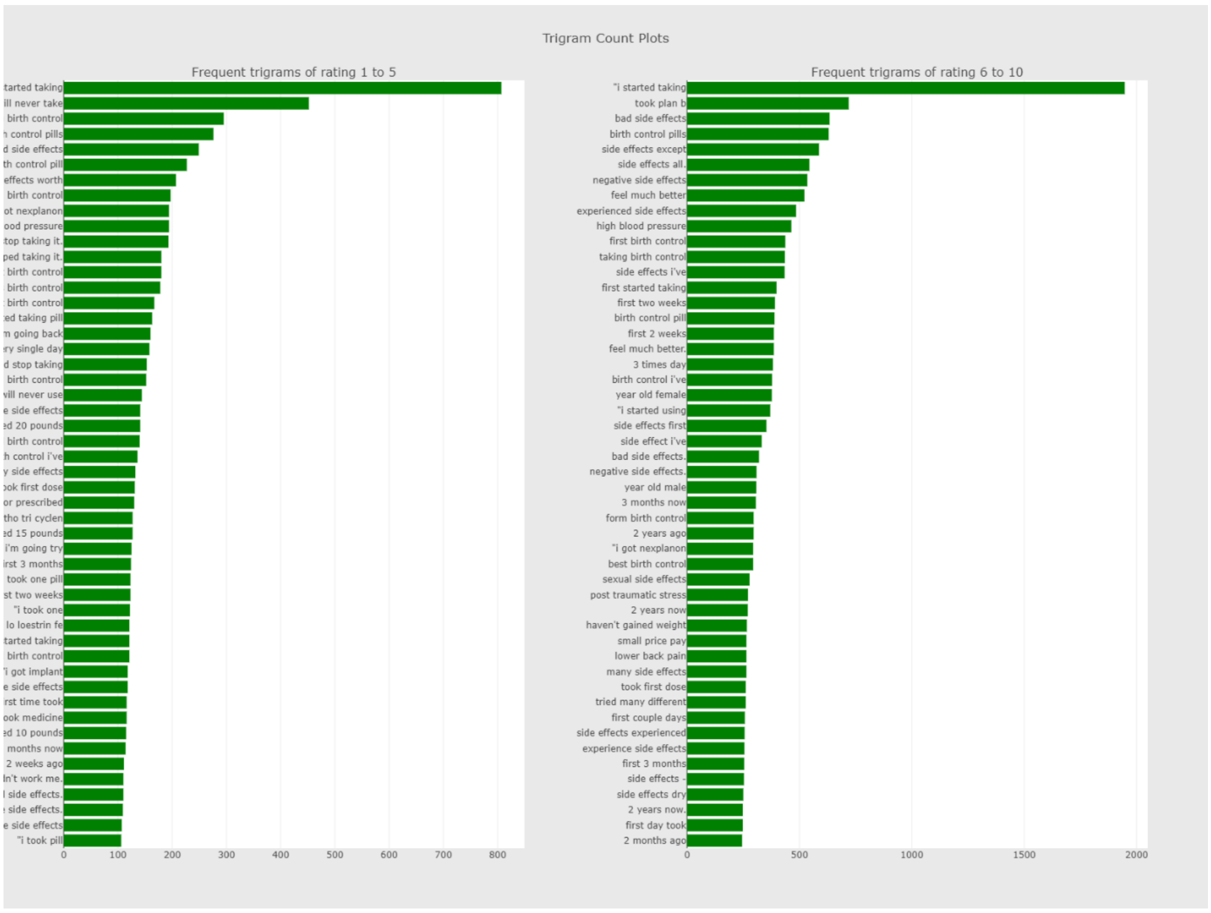
```
for sent in df_all_1_5["review"]:
    for word in generate_ngrams(sent,2):
        freq_dict[word] += 1
fd_sorted = pd.DataFrame(sorted(freq_dict.items(), key=lambda x: x
[1])[::-1])
fd_sorted.columns = ["word", "wordcount"]
trace1 = horizontal_bar_chart(fd_sorted.head(50), 'orange')

freq_dict = defaultdict(int)
for sent in df_all_6_10["review"]:
    for word in generate_ngrams(sent,2):
        freq_dict[word] += 1
fd_sorted = pd.DataFrame(sorted(freq_dict.items(), key=lambda x: x
[1])[::-1])
fd_sorted.columns = ["word", "wordcount"]
trace2 = horizontal_bar_chart(fd_sorted.head(50), 'orange')

# Creating two subplots
fig = tools.make_subplots(rows=1, cols=2, vertical_spacing=0.04,ho
rizontal_spacing=0.15,
                          subplot_titles=["Frequent biagrams of ra
ting 1 to 5",
                                          "Frequent biagrams of ra
ting 6 to 10"])
fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 1, 2)
fig['layout'].update(height=1200, width=1000, paper_bgcolor='rgb(2
33,233,233)', title="Bigram Count Plots")
py.iplot(fig, filename='word-plots')
```



Likewise, in 2-gram, the contents of the top five corpus are similar, and it is hard to classify positive and negative. In addition, 'side effects' and 'side effects.' are interpreted differently, which means preprocessing of review data is necessary. However, you can see that this is better to classify emotions rather than previous 1-grams, like side effects, weight gain, and highly recommend.

```
freq_dict = defaultdict(int)
for sent in df_all_1_5["review"]:
    for word in generate_ngrams(sent,3):
        freq_dict[word] += 1
```

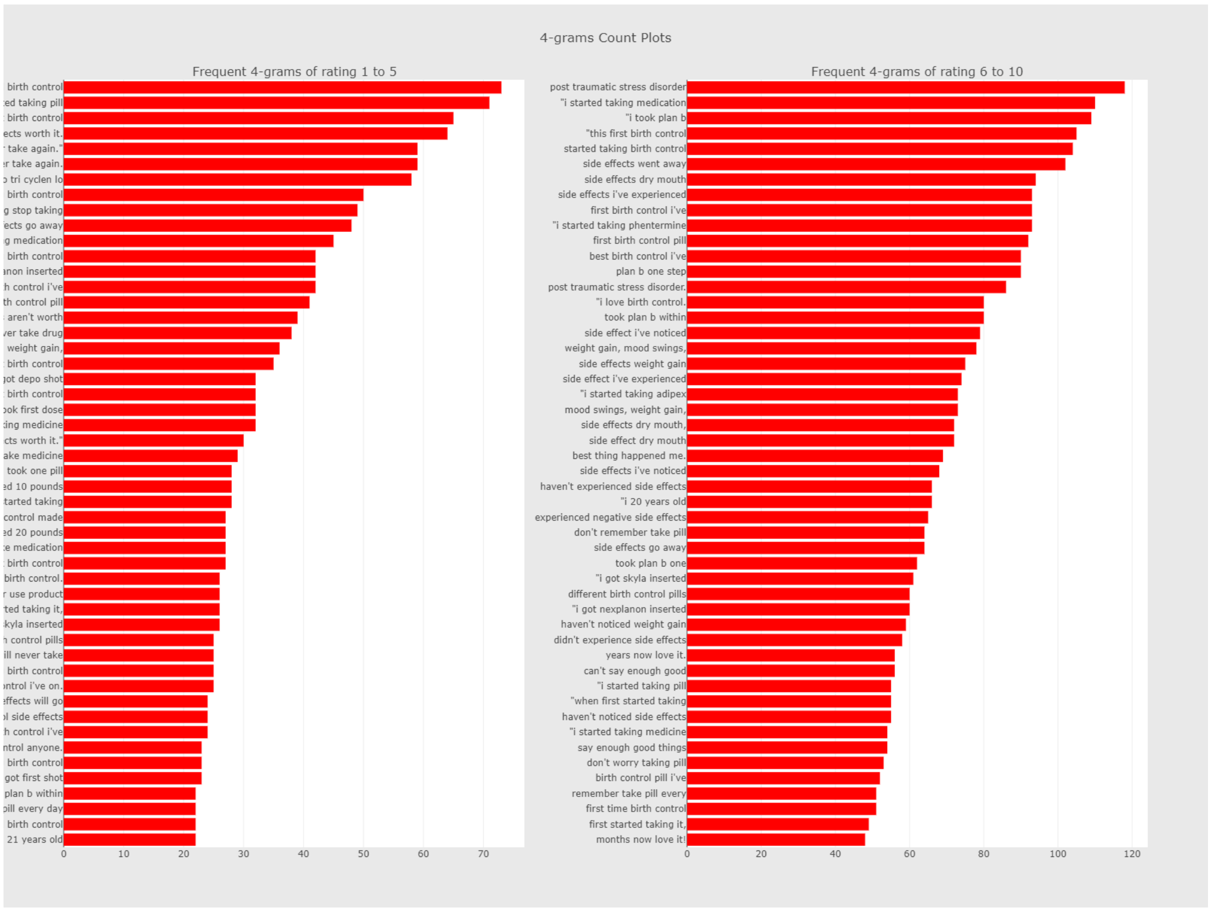
```
fd_sorted = pd.DataFrame(sorted(freq_dict.items(), key=lambda x: x
[1])[::-1])
fd_sorted.columns = ["word", "wordcount"]
trace1 = horizontal_bar_chart(fd_sorted.head(50), 'green')

freq_dict = defaultdict(int)
for sent in df_all_6_10["review"]:
    for word in generate_ngrams(sent,3):
        freq_dict[word] += 1
fd_sorted = pd.DataFrame(sorted(freq_dict.items(), key=lambda x: x
[1])[::-1])
fd_sorted.columns = ["word", "wordcount"]
trace2 = horizontal_bar_chart(fd_sorted.head(50), 'green')

# Creating two subplots
fig = tools.make_subplots(rows=1, cols=2, vertical_spacing=0.04,ho
rizontal_spacing=0.15,
                          subplot_titles=["Frequent trigrams of ra
ting 1 to 5",
                                          "Frequent trigrams of ra
ting 6 to 10"])
fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 1, 2)
fig['layout'].update(height=1200, width=1600, paper_bgcolor='rgb(2
33,233,233)', title="Trigram Count Plots")
py.iplot(fig, filename='word-plots')
```



From 3-gram you can see that there is a difference between positive and negative corpus. Bad side effects, birth control pills, negative side effects are corpus that classify positive and negative. However, both positive and negative parts can be thought that it has missing parts that reverses the context, such as' not' in front of a corpus.

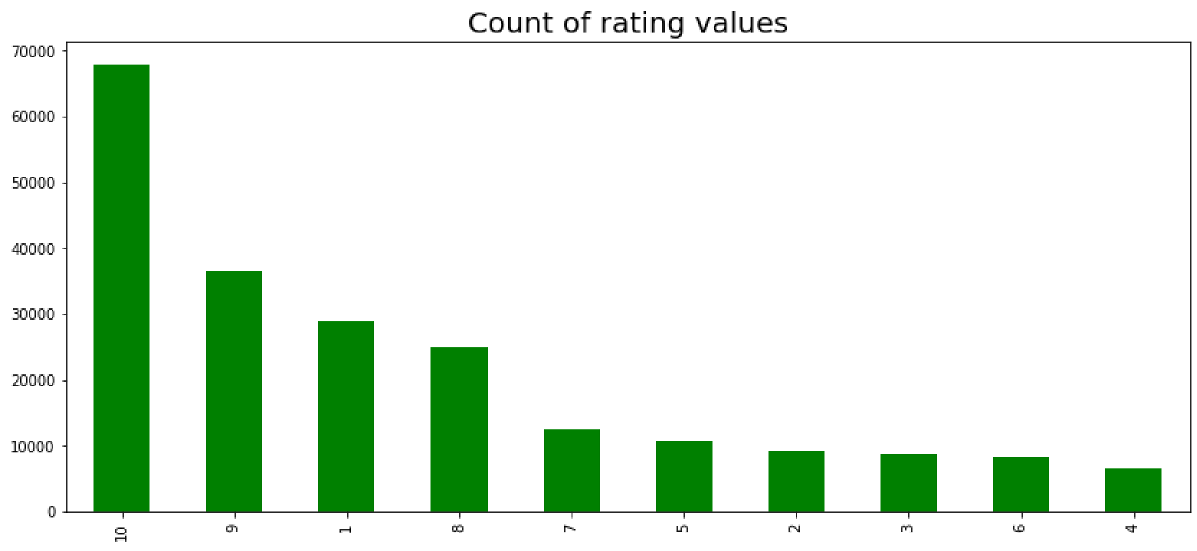
```
freq_dict = defaultdict(int)
for sent in df_all_1_5["review"]:
    for word in generate_ngrams(sent,4):
        freq_dict[word] += 1
fd_sorted = pd.DataFrame(sorted(freq_dict.items(), key=lambda x: x
[1])[::-1])
fd_sorted.columns = ["word", "wordcount"]
trace1 = horizontal_bar_chart(fd_sorted.head(50), 'red')

freq_dict = defaultdict(int)
for sent in df_all_6_10["review"]:
    for word in generate_ngrams(sent,4):
        freq_dict[word] += 1
fd_sorted = pd.DataFrame(sorted(freq_dict.items(), key=lambda x: x
[1])[::-1])
fd_sorted.columns = ["word", "wordcount"]
trace2 = horizontal_bar_chart(fd_sorted.head(50), 'red')

# Creating two subplots
```

```
fig = tools.make_subplots(rows=1, cols=2, vertical_spacing=0.04,ho
rizontal_spacing=0.15,
                          subplot_titles=["Frequent 4-grams of rat
ing 1 to 5",
                                          "Frequent 4-grams of rat
ing 6 to 10"])
fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 1, 2)
fig['layout'].update(height=1200, width=1600, paper_bgcolor='rgb(2
33,233,233)', title="4-grams Count Plots")
py.iplot(fig, filename='word-plots')
```



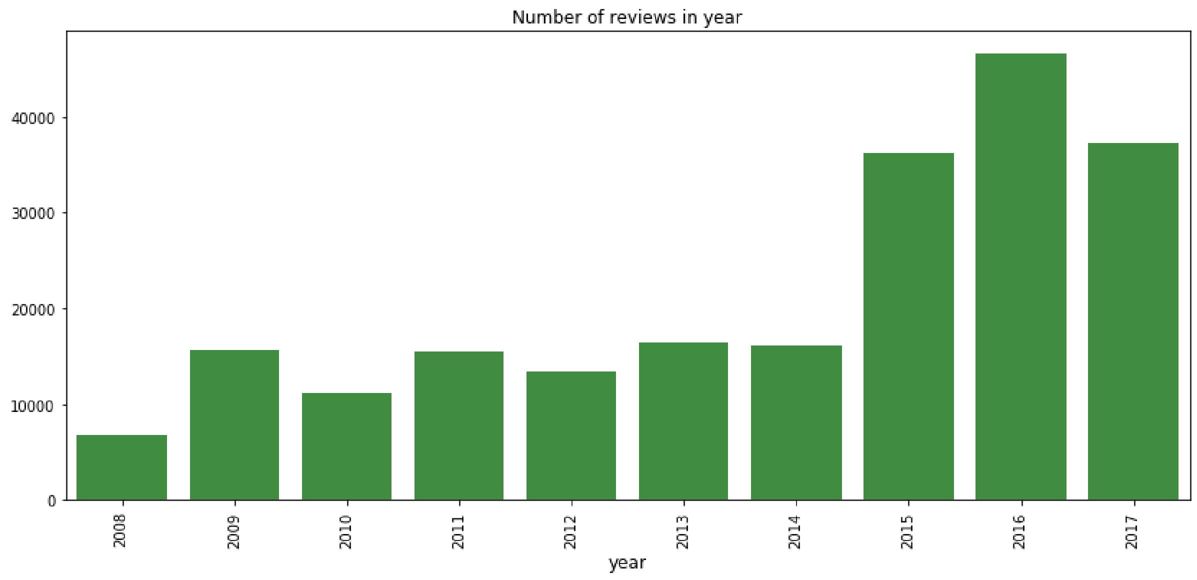
Clearly, 4-gram classifies emotions much betther than other grams. Therefore, we will use 4-gram to build deep learning model. Next, we will look for relationship between rating and weather. First of all, we will count the number of ratings.

```
rating = df_all['rating'].value_counts().sort_values(ascending=Fal
se)
rating.plot(kind="bar", figsize = (14,6), fontsize = 10,color="gre
en")
plt.xlabel("", fontsize = 20)
plt.ylabel("", fontsize = 20)
plt.title("Count of rating values", fontsize = 20)
```

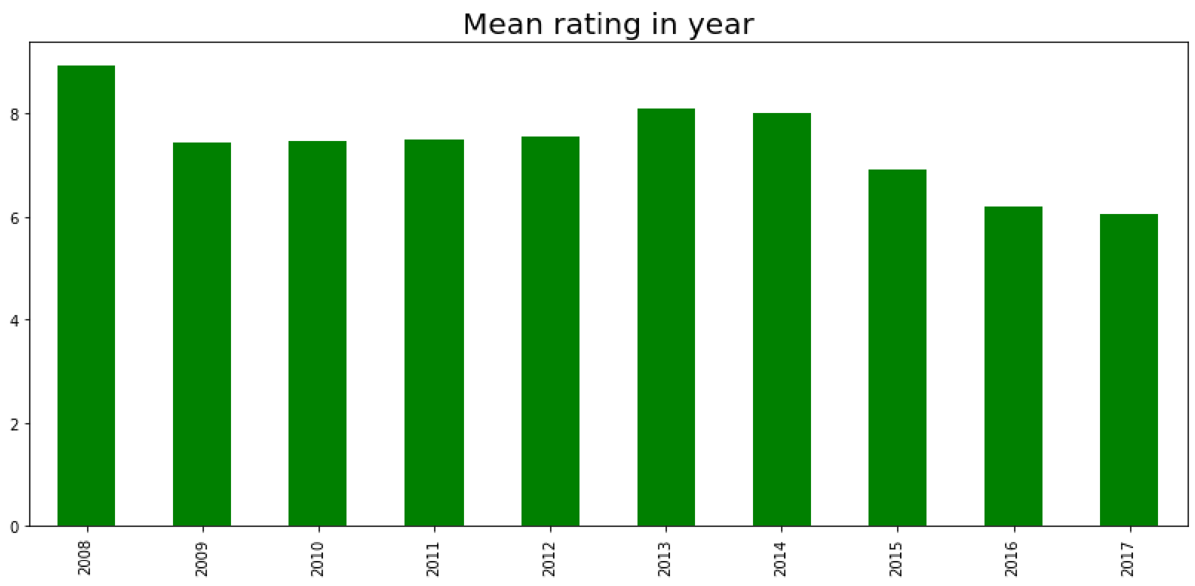


Most people choose four values; 10, 9, 1, 8, and the number of 10 is more than twice as many as the others. With this, we can see that the percentage of positives is higher than negative, and people's reactions are extreme. Next, we will check the number of reviews and percentage of ratings according to weather.

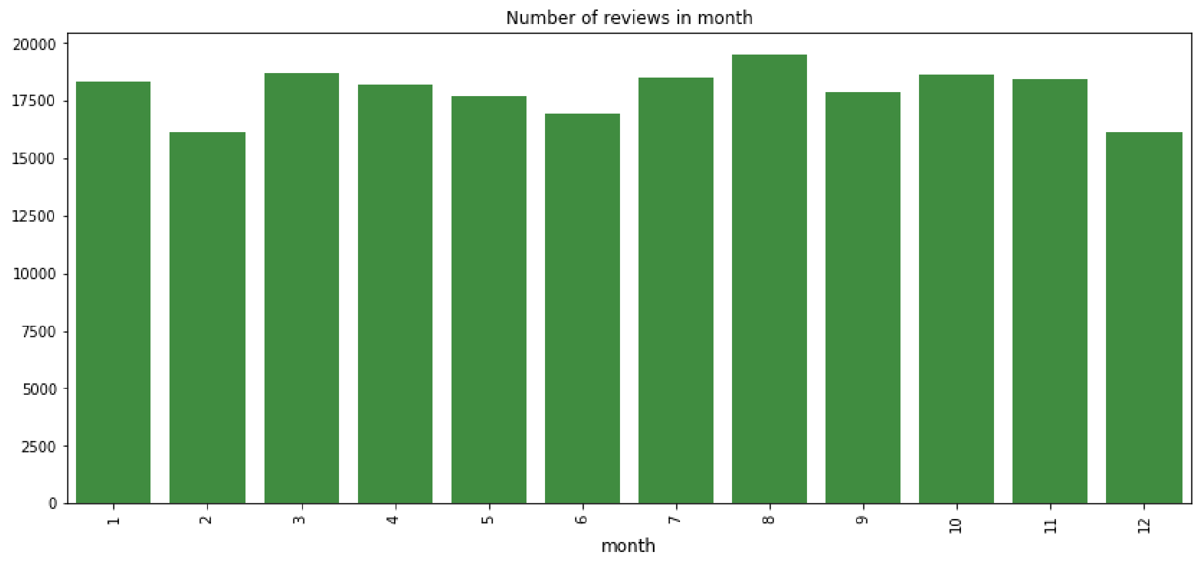
```
# Code in https://www.kaggle.com/sudalairajkumar/simple-exploration
-notebook-elo
# SRK - Simple Exploration Notebook

cnt_srs = df_all['date'].dt.year.value_counts()
cnt_srs = cnt_srs.sort_index()
plt.figure(figsize=(14,6))
sns.barplot(cnt_srs.index, cnt_srs.values, alpha=0.8, color='green
')
plt.xticks(rotation='vertical')
plt.xlabel('year', fontsize=12)
plt.ylabel('', fontsize=12)
plt.title("Number of reviews in year")
plt.show()
```



```
df_all['year'] = df_all['date'].dt.year
rating = df_all.groupby('year')['rating'].mean()
rating.plot(kind="bar", figsize = (14,6), fontsize = 10,color="green")
plt.xlabel("", fontsize = 20)
plt.ylabel("", fontsize = 20)
plt.title("Mean rating in year", fontsize = 20)
```

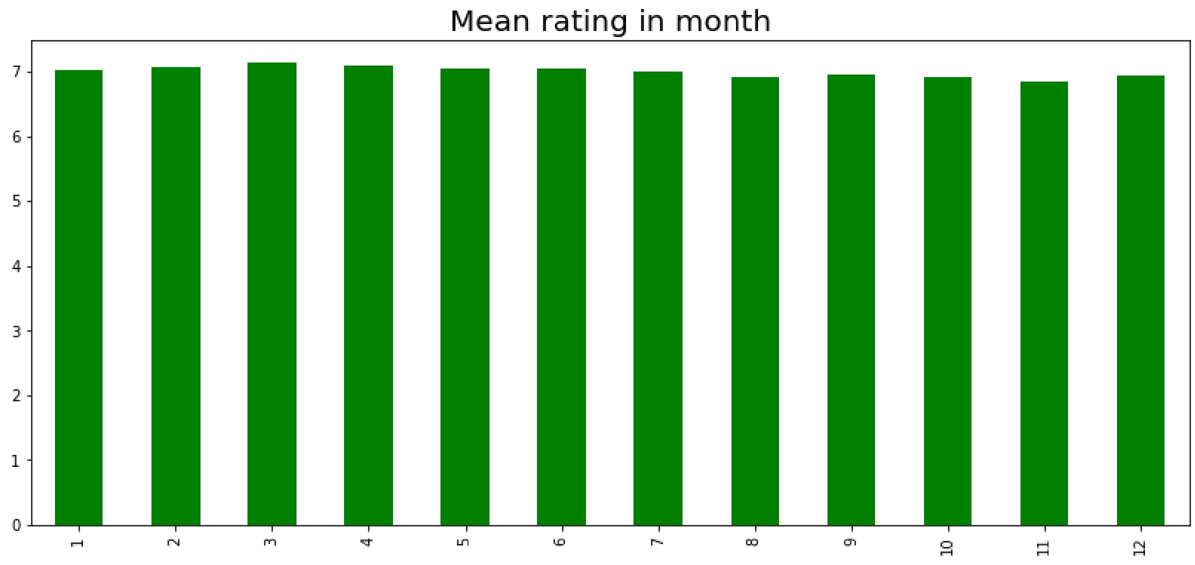


```
# Code in https://www.kaggle.com/sudalairajkumar/simple-exploration-noteb
ook-elo
# SRK - Simple Exploration Notebook

cnt_srs = df_all['date'].dt.month.value_counts()
cnt_srs = cnt_srs.sort_index()
plt.figure(figsize=(14,6))
sns.barplot(cnt_srs.index, cnt_srs.values, alpha=0.8, color='green')
plt.xticks(rotation='vertical')
plt.xlabel('month', fontsize=12)
plt.ylabel('', fontsize=12)
plt.title("Number of reviews in month")
plt.show()
```



```
df_all['month'] = df_all['date'].dt.month
rating = df_all.groupby('month')['rating'].mean()
rating.plot(kind="bar", figsize = (14,6), fontsize = 10,color="green")
plt.xlabel("", fontsize = 20)
plt.ylabel("", fontsize = 20)
plt.title("Mean rating in month", fontsize = 20)
```

```
Out[23]:
```

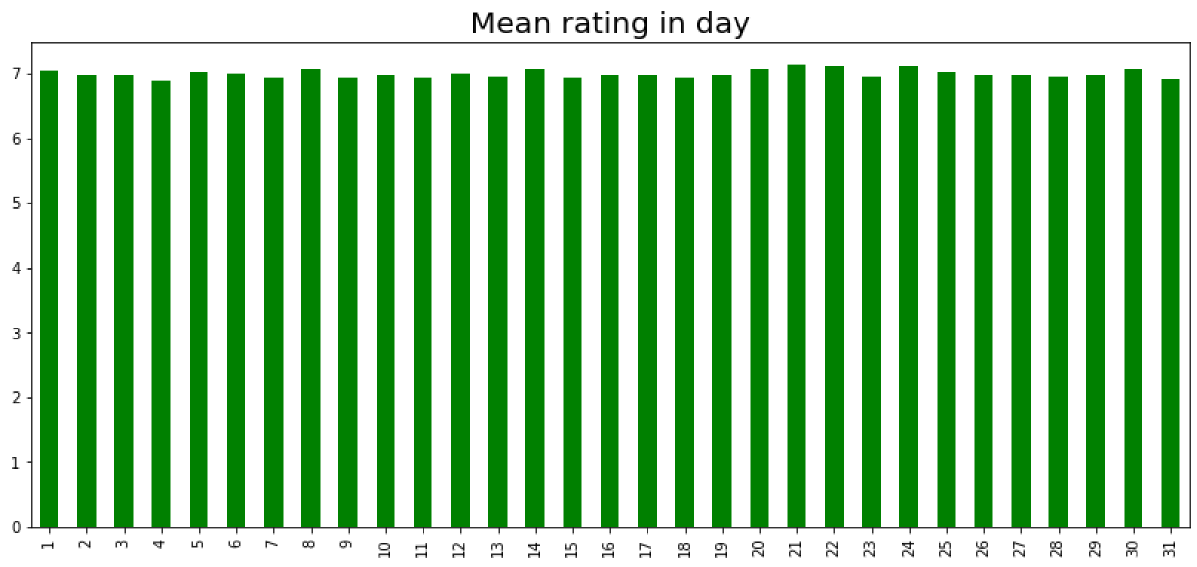


Interestingly, you can see that the average rating differs by year, but it is similar by month.

```
df_all['day'] = df_all['date'].dt.day
rating = df_all.groupby('day')['rating'].mean()
rating.plot(kind="bar", figsize = (14,6), fontsize = 10,color="green")
```

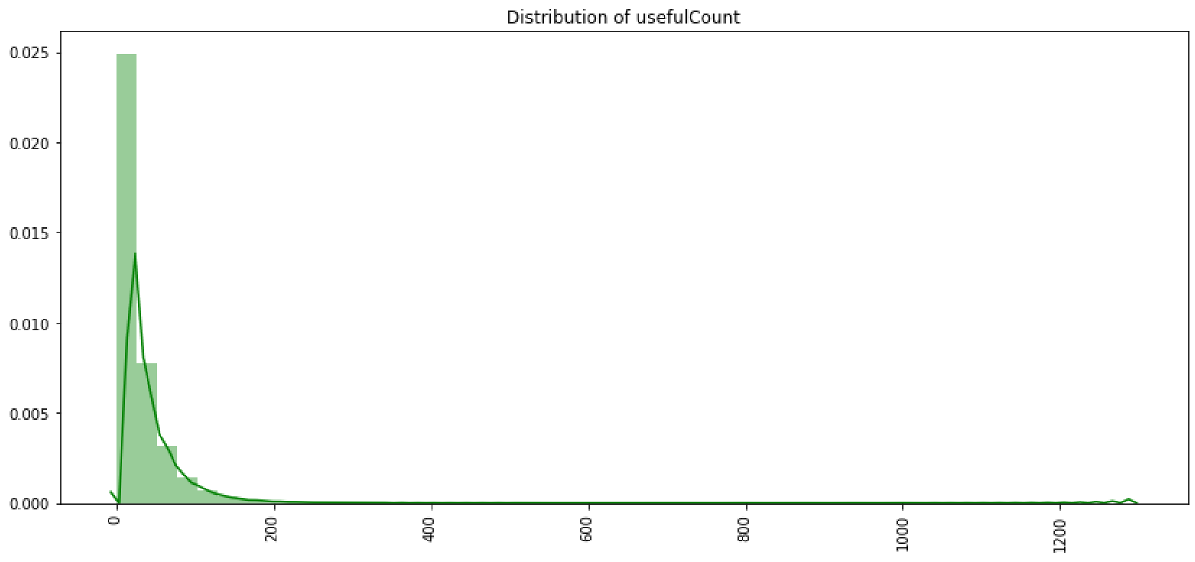
```
plt.xlabel("", fontsize = 20)
plt.ylabel("", fontsize = 20)
plt.title("Mean rating in day", fontsize = 20)
```

Out[24]:



We checked whether the day of the week affects the rating like salary day, but it does not make a big difference.

```
plt.figure(figsize=(14,6))
sns.distplot(df_all["usefulCount"].dropna(),color="green")
plt.xticks(rotation='vertical')
plt.xlabel('', fontsize=12)
plt.ylabel('', fontsize=12)
plt.title("Distribution of usefulCount")
plt.show()
```



```
df_all["usefulCount"].describe()
```

Out[26]:

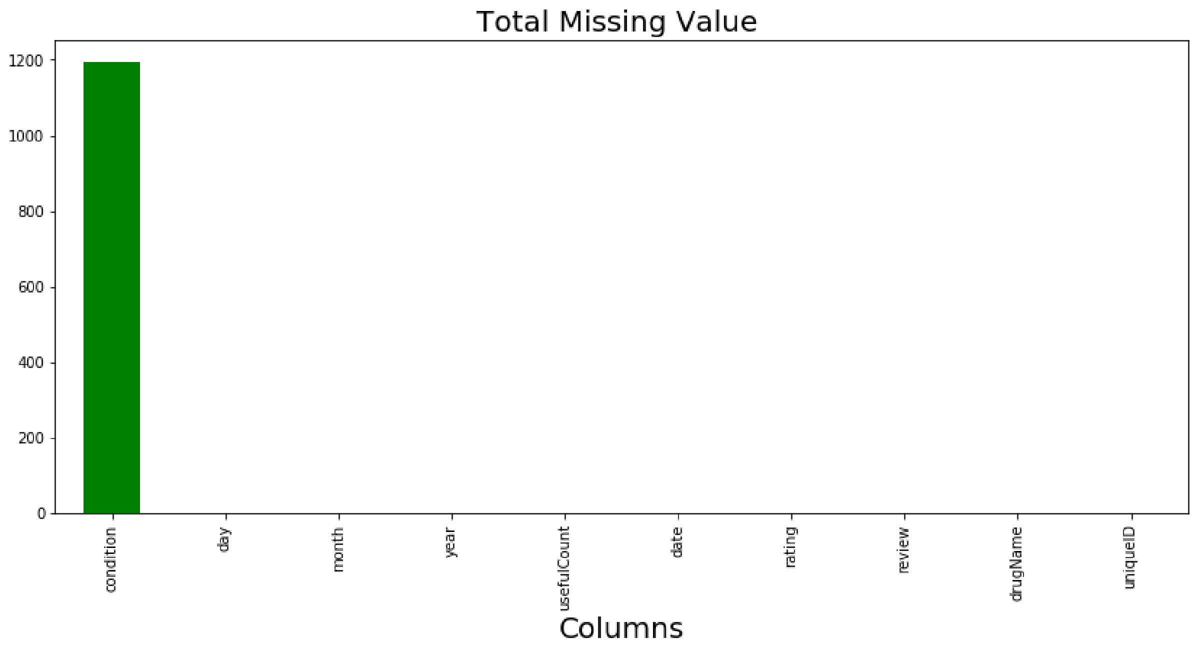
```
count     215063.000000
```

```
mean          28.001004
std           36.346069
min            0.000000
25%            6.000000
50%           16.000000
75%           36.000000
max         1291.000000
Name: usefulCount, dtype: float64
```

If you look at the distribution of usefulCount, you can see that the difference between minimum and maximum is 1291, which is high. In addition, the deviation is huge, which is 36. The reason for this is that the more drugs people look for, the more people read the review no matter their contents are good or bad, which makes the usefulcount very high. So when we create the model, we will normalize it by conditions, considering people's accessibility.

```
percent = (df_all.isnull().sum()).sort_values(ascending=False)
percent.plot(kind="bar", figsize = (14,6), fontsize = 10, color='green')
plt.xlabel("Columns", fontsize = 20)
plt.ylabel("", fontsize = 20)
plt.title("Total Missing Value ", fontsize = 20)
```

```
Out[27]:
```



```
print("Missing value (%):", 1200/df_all.shape[0] *100)
Missing value (%): 0.5579760349292998
```

We will delete because the percentage is lower than 1%.

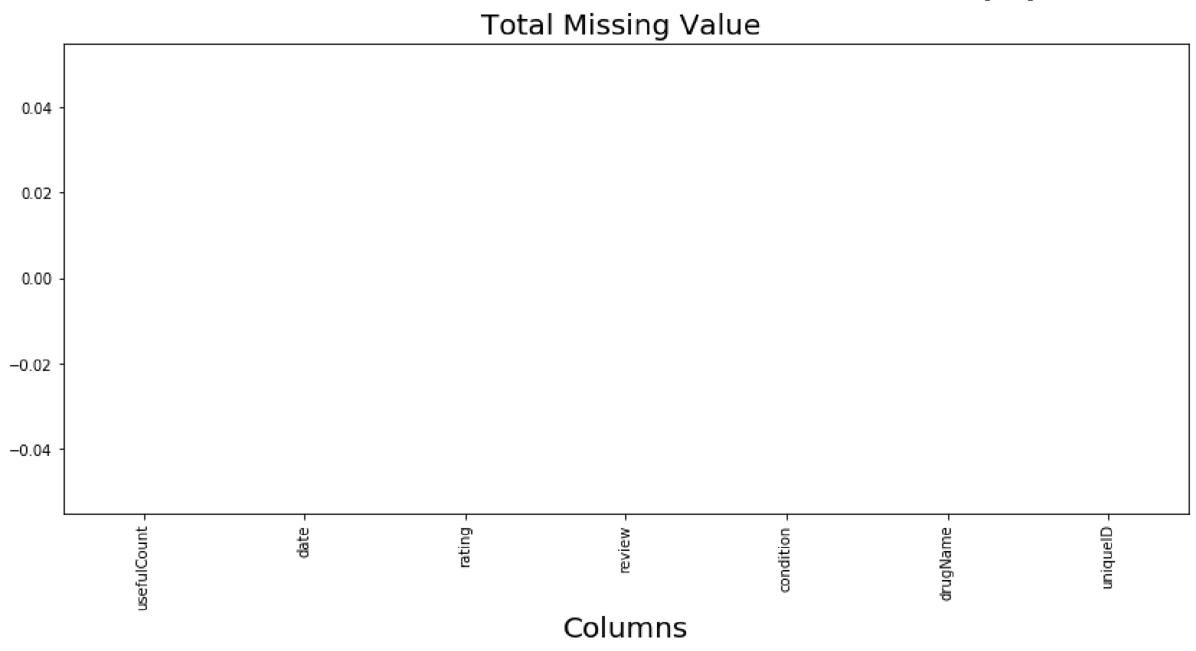
### 3.3 Date Preprocessing

```
df_train = df_train.dropna(axis=0)
df_test = df_test.dropna(axis=0)
```

```
df_all = pd.concat([df_train,df_test]).reset_index()
```

```
del df_all['index']
percent = (df_all.isnull().sum()).sort_values(ascending=False)
percent.plot(kind="bar", figsize = (14,6), fontsize = 10, color='green')
plt.xlabel("Columns", fontsize = 20)
plt.ylabel("", fontsize = 20)
plt.title("Total Missing Value ", fontsize = 20)
```

Out[30]:



We will delete the sentences with the form above.

In [31]:

```
all_list = set(df_all.index)
span_list = []
for i,j in enumerate(df_all['condition']):
    if '</span>' in j:
        span_list.append(i)
```

In [32]:

```
new_idx = all_list.difference(set(span_list))
df_all = df_all.iloc[list(new_idx)].reset_index()
del df_all['index']
```

Next, we will delete conditions with only one drug.

```
df_condition = df_all.groupby(['condition'])['drugName'].nunique().sort_val
ues(ascending=False)
df_condition = pd.DataFrame(df_condition).reset_index()
df_condition.tail(20)
```

Out[33]:

|  | condition | drugName |
|---|---|---|
| 816 | Hemangioma | 1 |
| 817 | Q Feve | 1 |
| 818 | Urinary Retention | 1 |
| 819 | Diagnostic Bronchograms | 1 |
| 820 | Steroid Responsive Inflammatory Conditions | 1 |
| 821 | Cluster-Tic Syndrome | 1 |
| 822 | Nausea (phosphorated carbohydrate solution) | 1 |
| 823 | Ramsay Hunt Syndrome | 1 |
| 824 | Rat-bite Feve | 1 |
| 825 | Hemorrhoids (pramoxine / zinc oxide) | 1 |
| 826 | Myotonia Congenita | 1 |
| 827 | Sepsis | 1 |
| 828 | ailure to Thrive | 1 |
| 829 | Hepatitis B Prevention | 1 |
| 830 | Transverse Myelitis | 1 |
| 831 | Tympanostomy Tube Placement Surgery | 1 |
| 832 | Muscle Twitching | 1 |
| 833 | Somatoform Pain Disorde | 1 |
| 834 | acial Lipoatrophy | 1 |
| 835 | Rabies Prophylaxis | 1 |

```
df_condition_1 = df_condition[df_condition['drugName']==1].reset_index()
df_condition_1['condition'][0:10]
```

Out[34]:

```
0                      Uveitis, Posteri
1                Pseudogout, Prophylaxis
2                    Infectious Diarrhea
3                Thyroid Suppression Test
4                             Angioedema
5                            Scleroderma
6                                    mis
7           Anti NMDA Receptor Encephalitis
8                                 mist (
9     Pruritus of Partial Biliary Obstruction
Name: condition, dtype: object
```

In [35]:

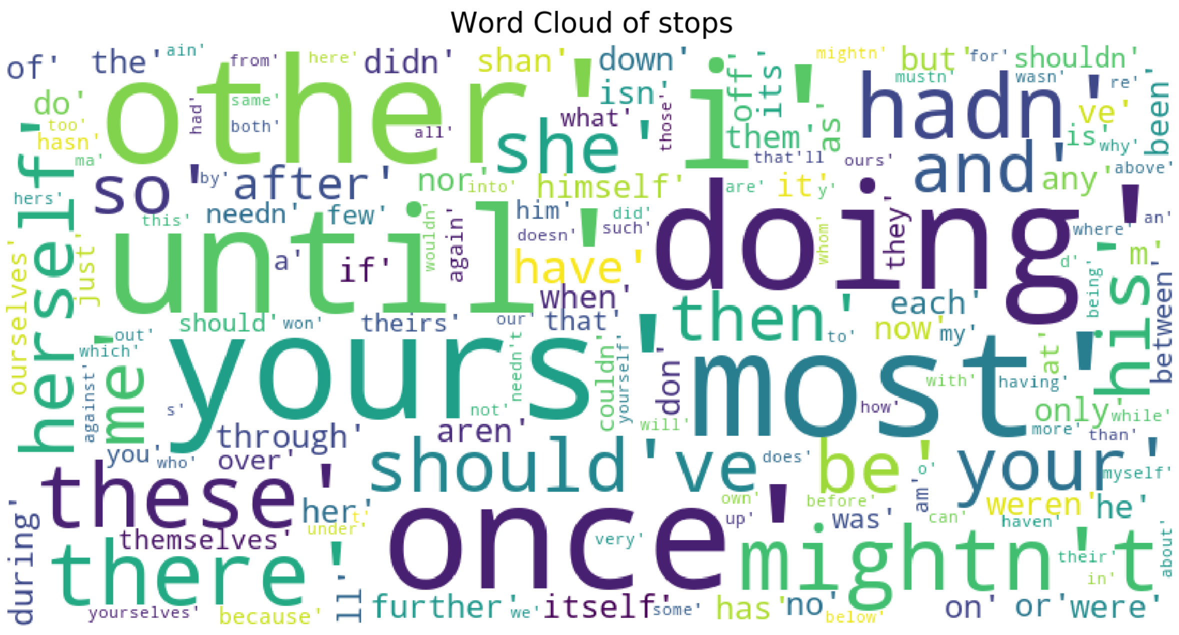
```
all_list = set(df_all.index)
condition_list = []
for i,j in enumerate(df_all['condition']):
    for c in list(df_condition_1['condition']):
        if j == c:
            condition_list.append(i)

new_idx = all_list.difference(set(condition_list))
df_all = df_all.iloc[list(new_idx)].reset_index()
del df_all['index']

from bs4 import BeautifulSoup
import nltk
```

```
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
```

- \r\n : we need to convert html grammer
- ... , ' : deal with not alphabet

In [37]:

```
stops = set(stopwords.words('english'))
#stops

#https://www.kaggle.com/sudalairajkumar/simple-exploration-notebook-qiqc ker
nel
from wordcloud import WordCloud, STOPWORDS

# Thanks : https://www.kaggle.com/aashita/word-clouds-of-various-shapes ##
def plot_wordcloud(text, mask=None, max_words=200, max_font_size=100, figur
e_size=(24.0,16.0),
                   title = None, title_size=40, image_color=False):
    stopwords = set(STOPWORDS)
    more_stopwords = {'one', 'br', 'Po', 'th', 'sayi', 'fo', 'Unknown'}
    stopwords = stopwords.union(more_stopwords)

    wordcloud = WordCloud(background_color='white',
                    stopwords = stopwords,
                    max_words = max_words,
                    max_font_size = max_font_size,
                    random_state = 42,
                    width=800,
                    height=400,
                    mask = mask)
    wordcloud.generate(str(text))

    plt.figure(figsize=figure_size)
    if image_color:
        image_colors = ImageColorGenerator(mask);
        plt.imshow(wordcloud.recolor(color_func=image_colors), interpolatio
n="bilinear");
        plt.title(title, fontdict={'size': title_size,
                                  'verticalalignment': 'bottom'})
    else:
        plt.imshow(wordcloud);
        plt.title(title, fontdict={'size': title_size, 'color': 'black',
                                  'verticalalignment': 'bottom'})
    plt.axis('off');
    plt.tight_layout()

plot_wordcloud(stops, title="Word Cloud of stops")
```



First, let's see what words are used as stopwords. There are many words that include not, like needn't. These words are key parts of emotional analysis, so we will remove them from stopwords.

```
                                                                        In [39]:
not_stop = ["aren't","couldn't","didn't","doesn't","don't","hadn't","hasn't
","haven't","isn't","mightn't","mustn't","needn't","no","nor","not","shan't
","shouldn't","wasn't","weren't","wouldn't"]
for i in not_stop:
    stops.remove(i)

from sklearn import model_selection, preprocessing, metrics, ensemble, naiv
e_bayes, linear_model
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorize
r
from sklearn.decomposition import TruncatedSVD
import lightgbm as lgb

pd.options.mode.chained_assignment = None
pd.options.display.max_columns = 999
from bs4 import BeautifulSoup
import re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline

from sklearn.model_selection import train_test_split
from sklearn import metrics

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation
, CuDNNGRU, Conv1D
from keras.layers import Bidirectional, GlobalMaxPool1D
from keras.models import Model
from keras import initializers, regularizers, constraints, optimizers, laye
rs

stemmer = SnowballStemmer('english')

def review_to_words(raw_review):
    # 1. Delete HTML
    review_text = BeautifulSoup(raw_review, 'html.parser').get_text()
    # 2. Make a space
    letters_only = re.sub('[^a-zA-Z]', ' ', review_text)
    # 3. lower letters
    words = letters_only.lower().split()
    # 5. Stopwords
    meaningful_words = [w for w in words if not w in stops]
    # 6. Stemming
    stemming_words = [stemmer.stem(w) for w in meaningful_words]
    # 7. space join words
    return( ' '.join(stemming_words))
```

In [42]:

```
%time df_all['review_clean'] = df_all['review'].apply(review_to_words)
```

```
CPU times: user 2min 44s, sys: 448 ms, total: 2min 44s
Wall time: 2min 44s
```

## 4. Models

### 4.1 Collaborative Filtering

```
bow_vector = CountVectorizer(tokenizer = spacy_tokenizer,
ngram_range=(1,2))
#  tf-idf vector
tfidf_vector = TfidfVectorizer(tokenizer = spacy_tokenizer)
```

```
# part 1---vader sentiment analyzer for c_review
analyzer = SentimentIntensityAnalyzer()
# create new col vaderReviewScore based on C-review
df['vaderReviewScore'] = df['review_clean'].apply(lambda x:
analyzer.polarity_scores(x)['compound'])

# define the positive, neutral and negative
positive_num = len(df[df['vaderReviewScore'] >=0.05])
neutral_num = len(df[(df['vaderReviewScore'] >-0.05) &
(df['vaderReviewScore']<0.05)])
negative_num = len(df[df['vaderReviewScore']<=-0.05])

# create new col vaderSentiment based on vaderReviewScore
df['vaderSentiment'] = df['vaderReviewScore'].map(lambda x:int(2) if
x>=0.05 else int(1) if x<=-0.05 else int(0) )
df['vaderSentiment'].value_counts() # 2-pos: 99519; 1-neg: 104434; 0-neu:
11110

# label pos/neg/neu based on vaderSentiment result
df.loc[df['vaderReviewScore'] >=0.05,"vaderSentimentLabel"] ="positive"
df.loc[(df['vaderReviewScore'] >-0.05) &
(df['vaderReviewScore']<0.05),"vaderSentimentLabel"]= "neutral"
df.loc[df['vaderReviewScore']<=-0.05,"vaderSentimentLabel"] = "negative"
```

```
df['vaderReviewScore'].max()
```

```
0.992
```

```
df['vaderReviewScore'].min()
```

```
-0.9976
```

```
criteria = [df['vaderReviewScore'].between(-0.997, -0.799),
df['vaderReviewScore'].between(-0.798, -0.601),
df['vaderReviewScore'].between(-0.600, 0.403),
df['vaderReviewScore'].between(-0.402, -0.205),
df['vaderReviewScore'].between(-0.204, -0.007),
df['vaderReviewScore'].between(-0.006,0.191),
df['vaderReviewScore'].between(0.192, 0.389),
df['vaderReviewScore'].between(0.390, 0.587),
df['vaderReviewScore'].between(0.588, 0.785),
df['vaderReviewScore'].between(0.786, 1)]
values = [1, 2, 3,4,5,6,7,8,9,10]
```

```
df['normalVaderScore'] = np.select(criteria, values, 0)
```

```
df['meanNormalizedScore'] = (df['rating'] + df['normalVaderScore'])/2
```

```
grouped = df.groupby(['condition','drug name',
'ID']).agg({'meanNormalizedScore' : ['mean']})
grouped.to_csv('Medicare_Normalized_results')
grouped1 = grouped.reset_index()
grouped1.head(100)
```

```
grouped1.set_index('ID')
```

```
user_ratings = grouped1.pivot_table(index = ['condition'], columns =
['ID'], values = 'meanNormalizedScore')
user_ratings1 = grouped1.pivot_table(index = ['condition'], columns =
['ID'], values = 'meanNormalizedScore')
user_ratings.head(20)
user_ratings.to_csv('Itemtoitem_recom.csv')
```

```
user_ratings.iloc[0,:].sum(axis=0)
```

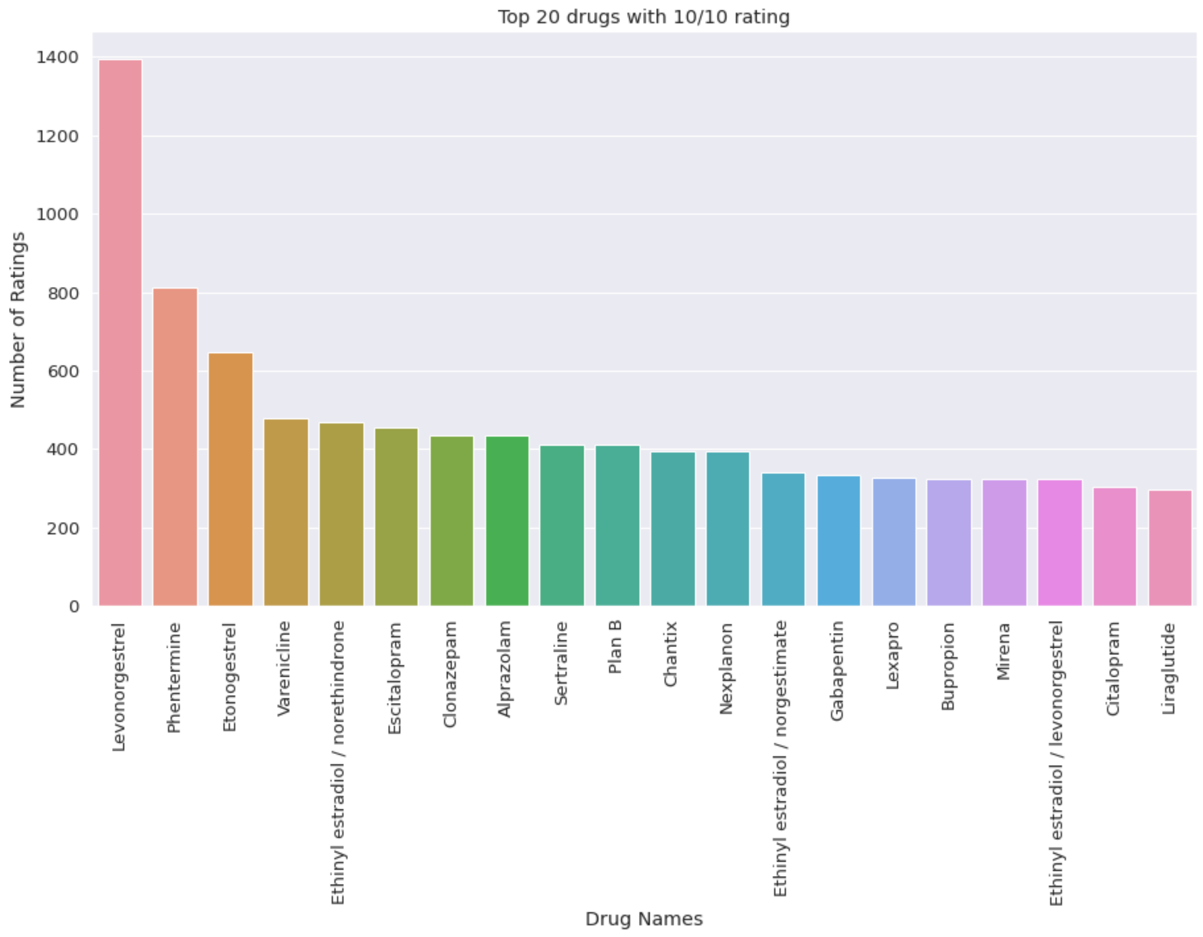
```
4.889422200449189e-13
```

```
#Let's predict the rating for the first drug  for the 1st condition.
#Therefore, we need to find the similarity between the them
import math as np
def takesec(num):
    return num[1]
```

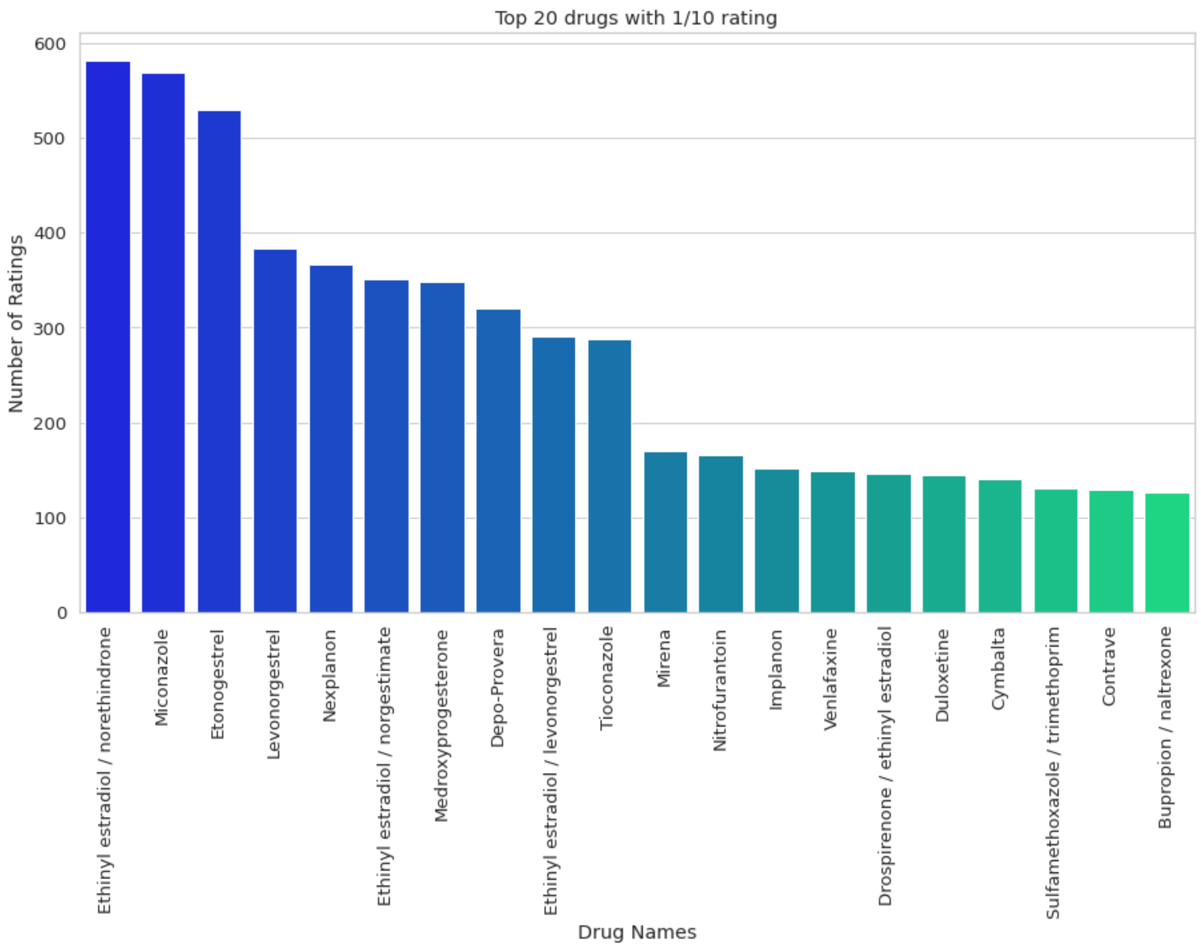
```
cosine_vector = []
num = 0
l1 = 0
```

```
l2 = 0
for i in range(1,user_ratings.shape[0]):
    num = 0
    l1 = 0
    l2 = 0
    for j in range(user_ratings.shape[1]):
        if not np.isnan(user_ratings.iloc[i,j]) and not
np.isnan(user_ratings.iloc[0,j]):
            num = num + (user_ratings.iloc[i,j] * user_ratings.iloc[0,j])
        if not np.isnan(user_ratings.iloc[i,j]):
            l1 = l1 + (user_ratings.iloc[i,j] * user_ratings.iloc[i,j])
        if not np.isnan(user_ratings.iloc[0,j]):
            l2 = l2 + (user_ratings.iloc[0,j] * user_ratings.iloc[0,j])
    eventual_prod = np.sqrt(l1) * np.sqrt(l2)
    if eventual_prod != 0 :
        eventual_div = num/eventual_prod
        cosine_vector.append([i,eventual_div])
        cosine_vector.sort(key=takesec, reverse=True)

cosine_vector[:100000]
      user_ratings1.iloc[4,1]
5.5

import csv
# opening the csv file in 'w+' mode
file = open('cosine_vec.csv', 'w+', newline ='')

# writing the data into the file
with file:
    write = csv.writer(file)
        write.writerows(cosine_vector)

#Let's consider the top 50 similar rated drugs and predict the output.

predict_drug = int(input ("Enter a drug ID within range  0 to 1000000 : "))
count = 0
num1 = 0
den1 = 0
for i in cosine_vector:
    if user_ratings1.iloc[i[0],predict_drug] > 0:
            count = count + 1
            num1 = num1 + i[1] * user_ratings1.iloc[i[0],predict_drug]
            #print(num1)
            den1 = den1 + i[1]
            #print(den1)
            #print("{}..{}".format(i[0],i[1]))

    if count == 50:
        print ("Reached 50 :)")
        break

print("Expected Rating for 1st Drug for condition: {} is
:{}".format(predict_drug,num1/den1))

for i in range(user_ratings1.shape[1]):
    if user_ratings1.iloc[0,i] > 0:
        print("Id = {} , Rating = {}".format(i,user_ratings1.iloc[0,i]))

import seaborn as sns
```

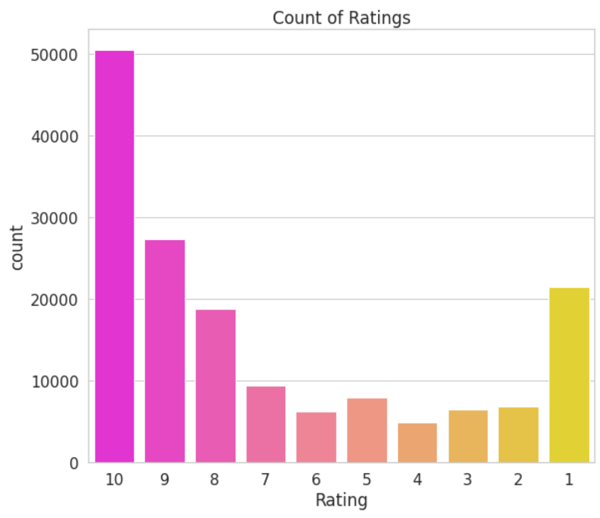
```
#!pip install matplotlib
import matplotlib.pyplot as plt

# Setting the Parameter
sns.set(font_scale = 1.2, style = 'darkgrid')
plt.rcParams['figure.figsize'] = [15, 8]

rating = dict(df.loc[df.rating == 10, "drug name"].value_counts())
drugname = list(rating.keys())
drug_rating = list(rating.values())

sns_rating = sns.barplot(x = drugname[0:20], y = drug_rating[0:20])

sns_rating.set_title('Top 20 drugs with 10/10 rating')
sns_rating.set_ylabel("Number of Ratings")
sns_rating.set_xlabel("Drug Names")
plt.setp(sns_rating.get_xticklabels(), rotation=90);
```

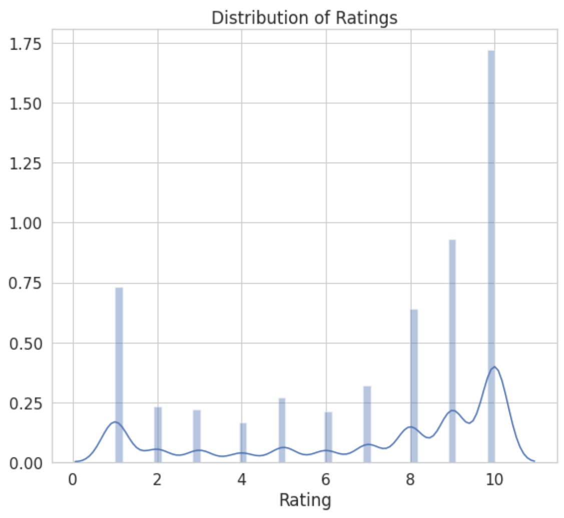


```
# Setting the Parameter
sns.set(font_scale = 1.2, style = 'whitegrid')
plt.rcParams['figure.figsize'] = [15, 8]

rating = dict(df.loc[df.rating == 1, "drug name"].value_counts())
drugname = list(rating.keys())
drug_rating = list(rating.values())

sns_rating = sns.barplot(x = drugname[0:20], y = drug_rating[0:20], palette
= 'winter')

sns_rating.set_title('Top 20 drugs with 1/10 rating')
sns_rating.set_ylabel("Number of Ratings")
sns_rating.set_xlabel("Drug Names")
plt.setp(sns_rating.get_xticklabels(), rotation=90);
```

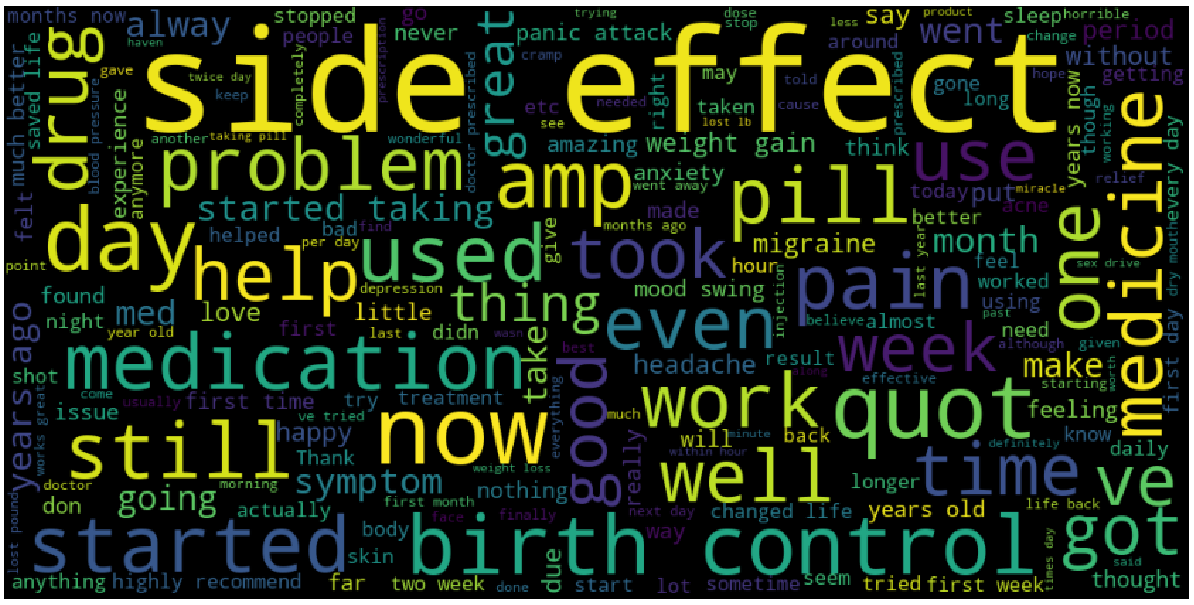


```
# A countplot of the ratings so we can see the distribution of the
ratings
plt.rcParams['figure.figsize'] = [20,8]
sns.set(font_scale = 1.4, style = 'whitegrid')
fig, ax = plt.subplots(1, 2)

sns_1 = sns.countplot(df['rating'], palette = 'spring', order =
list(range(10, 0, -1)), ax = ax[0])
sns_2 = sns.distplot(df['rating'], ax = ax[1])
sns_1.set_title('Count of Ratings')
sns_1.set_xlabel("Rating")

sns_2.set_title('Distribution of Ratings')
sns_2.set_xlabel("Rating")
```

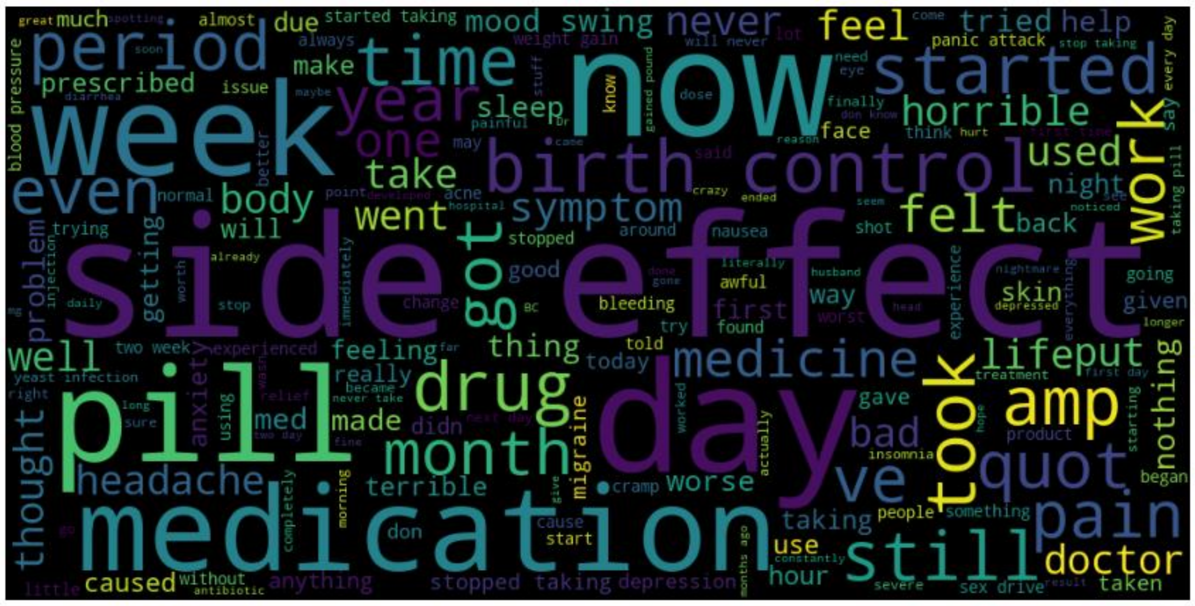




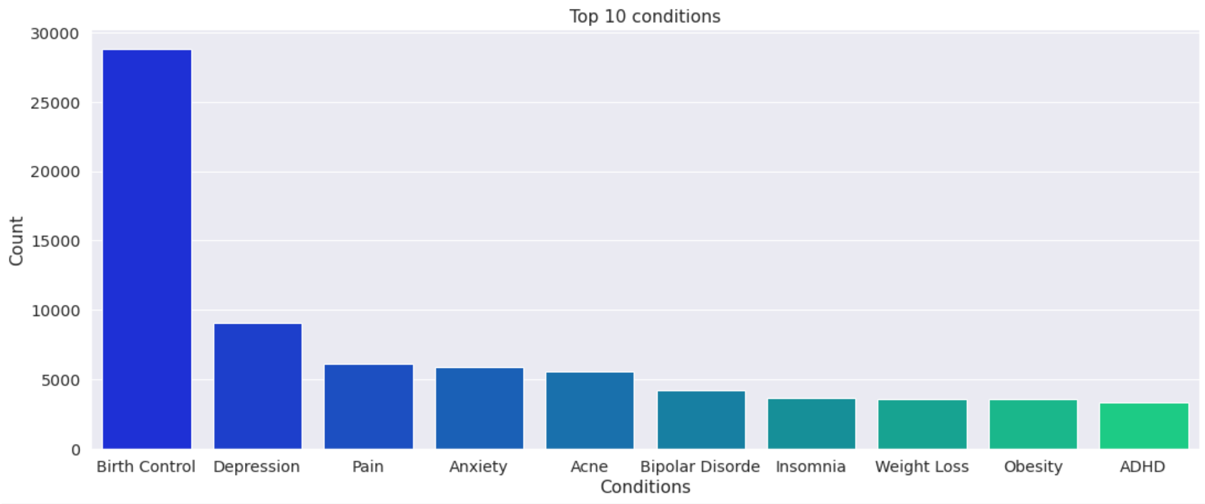
```
#!pip install wordcloud
from wordcloud import WordCloud# Word cloud of the reviews with rating
equal to 10
df_rate_ten = df.loc[df.rating == 10, 'review']
k = (' '.join(df_rate_ten))

wordcloud = WordCloud(width = 1000, height = 500).generate(k)
plt.figure(figsize=(15, 10))
plt.imshow(wordcloud)
plt.axis('off');
```



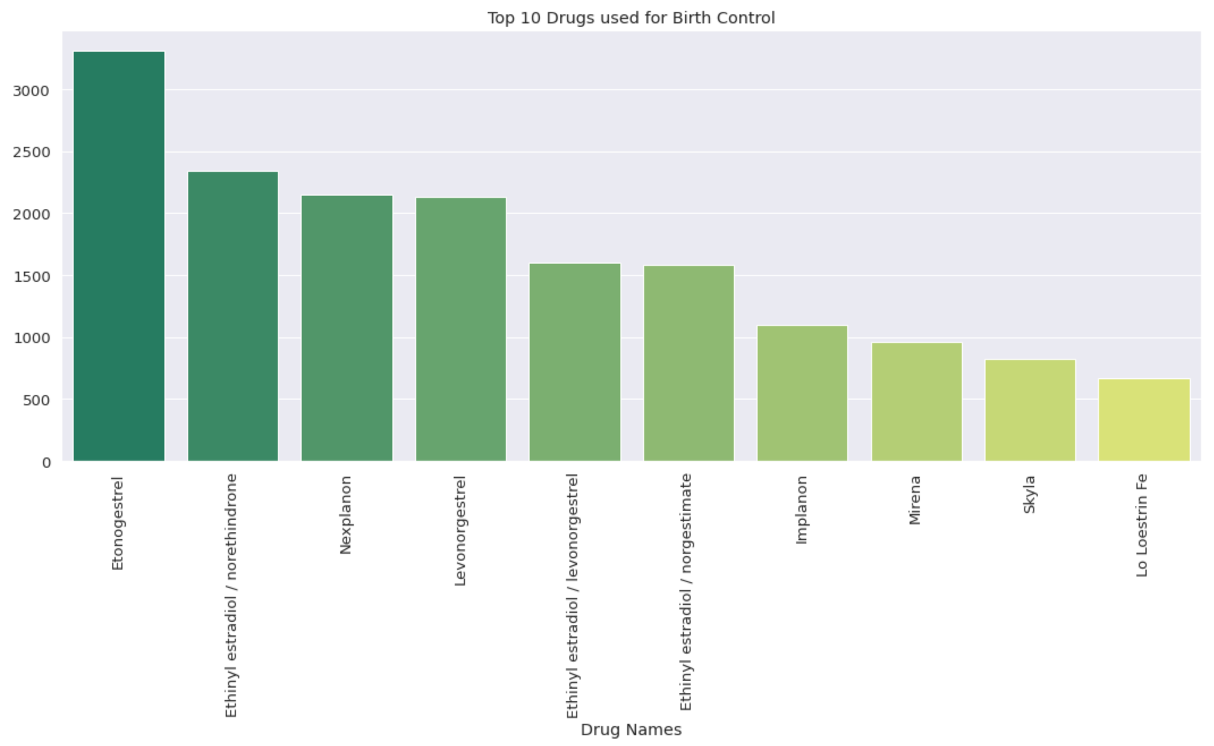
```
from wordcloud import WordCloud# Word cloud of the reviews with rating
equal to 1
df_rate_ten = df.loc[df.rating == 1, 'review']
k = (' '.join(df_rate_ten))

wordcloud = WordCloud(width = 1000, height = 500).generate(k)
plt.figure(figsize=(15, 10))
plt.imshow(wordcloud)
plt.axis('off');
```



```
# This barplot show the top 10 conditions the people are suffering.
cond = dict(df['condition'].value_counts())
top_condition = list(cond.keys())[0:10]
values = list(cond.values())[0:10]
sns.set(style = 'darkgrid', font_scale = 1.3)
plt.rcParams['figure.figsize'] = [18, 7]

sns_ = sns.barplot(x = top_condition, y = values, palette = 'winter')
sns_.set_title("Top 10 conditions")
sns_.set_xlabel("Conditions")
sns_.set_ylabel("Count");
```



```
# Top 10 drugs which are used for the top condition, that is Birth Control
df = df[df['condition'] == 'Birth Control']['drug name'].value_counts()[0:
10]
sns.set(font_scale = 1.2, style = 'darkgrid')

sns_ = sns.barplot(x = df.index, y = df.values, palette = 'summer')
sns_.set_xlabel('Drug Names')
sns_.set_title("Top 10 Drugs used for Birth Control")
plt.setp(sns_.get_xticklabels(), rotation = 90);
```



```
#Verification using Machine Learning Models
from sklearn.model_selection import train_test_split

x_train, x_test = train_test_split(df, test_size = 0.25, random_state = 0)

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline

cv = CountVectorizer(max_features = 20000, ngram_range = (4, 4))
pipeline = Pipeline([('vect',cv)])

x_train_features = pipeline.fit_transform(x_train['review_clean'])
x_test_features = pipeline.fit_transform(x_test['review_clean'])

print("x_train_features :", x_train_features.shape)
print("x_test_features :", x_test_features.shape)
# let's make a new column review sentiment

df.loc[(df['rating'] >= 5), 'Review_Sentiment'] = 1
df.loc[(df['rating'] < 5), 'Review_Sentiment'] = 0

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

```
x_train_features : (119856, 20000)
x_test_features : (39952, 20000)
```

Out[34]:

```
1.0    120085
0.0     39723
Name: Review_Sentiment, dtype: int64
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from mlxtend.classifier import StackingClassifier
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import precision_score, confusion_matrix,
recall_score
from sklearn.metrics import classification_report
# making our dependent variable



features = df['review_clean'] # the features we want to analyze
labels = df['Review_Sentiment'] # the labels, we want to test against
X_train, X_test, y_train, y_test= train_test_split(features, labels,
test_size=0.2, random_state=0)

clf2 =
LogisticRegression(random_state=0,solver='lbfgs',max_iter=2000,multi_cl
ass='auto')
#clf3 = SVC(kernel="linear", C=5)
clf7 = SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3,
random_state=42, max_iter=10, tol=None)


# Logistic Regression

x_train1 = x_train['review_clean']
pipe2 = Pipeline([('vectorizer', bow_vector),
                 ('classifier', clf2)])
pipe2.fit(X_train, y_train)
y_pred2 = pipe2.predict(X_test)

logreg_accuracy = accuracy_score(y_test, y_pred2)
print("Log Reg: ", logreg_accuracy )

Log Reg:  0.9133032976659784

#AUC
from sklearn.metrics import roc_curve, auc
y_score = pipe2.fit(X_train, y_train).decision_function(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_score)

#Classification Report
target_names = ["class 1", "class 2"]
print(classification_report(y_test, y_pred2,target_names=target_names))

              precision    recall  f1-score   support

     class 1       0.86      0.78      0.82      7920
     class 2       0.93      0.96      0.94     24042

    accuracy                           0.91     31962
   macro avg       0.89      0.87      0.88     31962
weighted avg       0.91      0.91      0.91     31962
```

```
#Confusion Matrix
cm2 = confusion_matrix(y_pred=y_pred2, y_true=y_test)
cm2
```

Out[31]:

```
array([[ 6199,  1721],
       [ 1050, 22992]])
```

In [32]:

```
print('AUC: {}'.format(auc(fpr, tpr)))

AUC: 0.9438023599693802
```

In [39]:

```
#SGD
pipe7 = Pipeline([('vectorizer', bow_vector),
                 ('classifier', clf7)])
pipe7.fit(X_train, y_train)
y_pred7 = pipe7.predict(X_test)
sgd_accuracy = accuracy_score(y_test, y_pred7)
print("SGD: ", sgd_accuracy)

SGD:  0.8504161191414805
```

In [40]:

```
#AUC
from sklearn.metrics import roc_curve, auc
y_score2 = pipe7.fit(X_train, y_train).decision_function(X_test)
fpr2, tpr2, thresholds2 = roc_curve(y_test, y_score2)
print('AUC: {}'.format(auc(fpr2, tpr2)))

AUC: 0.903384454939546
```

In [41]:

```
#Classification Report
target_names = ["class 1", "class 2"]
print(classification_report(y_test, y_pred7,target_names=target_names))

              precision    recall  f1-score   support

     class 1       0.81      0.52      0.63      7920
     class 2       0.86      0.96      0.91     24042

    accuracy                           0.85     31962
   macro avg       0.83      0.74      0.77     31962
weighted avg       0.85      0.85      0.84     31962
```

In [42]:

```
#Confusion Matrix
cm2 = confusion_matrix(y_pred=y_pred7, y_true=y_test)
cm2
```

Out[42]:

```
array([[ 4109,  3811],
       [  970, 23072]])
```

In [0]:

```
clf5 = MultinomialNB()
```

In [45]:

```
pipe5 = Pipeline([
                 ('vectorizer', bow_vector),
                 ('classifier', clf5)])
pipe5.fit(X_train, y_train)
y_pred5 = pipe5.predict(X_test)
print("MultinomialNB: ", accuracy_score(y_test, y_pred5))
```

```
MultinomialNB:  0.877886239909893
```

In [48]:

```
#AUC
from sklearn.metrics import roc_curve, auc
y_score3 = pipe5.fit(X_train, y_train).predict_proba(X_test)
fpr3, tpr3, thresholds3 = roc_curve(y_test, y_score3[:,1])
print('AUC: {}'.format(auc(fpr3, tpr3)))
```

```
AUC: 0.9084264731585047
```

In [49]:

```
#Classification Report
target_names = ["class 1", "class 2"]
print(classification_report(y_test, y_pred5,target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 1       0.87      0.60      0.71      7920
     class 2       0.88      0.97      0.92     24042

    accuracy                           0.88     31962
   macro avg       0.87      0.78      0.82     31962
weighted avg       0.88      0.88      0.87     31962
```

In [50]:

```
#Confusion Matrix
cm3 = confusion_matrix(y_pred=y_pred5, y_true=y_test)
cm3
```

Out[50]:

```
array([[ 4755,  3165],
       [  738, 23304]])
```

### 4.2 Deep Learning Model Using N-gram

```
# Make a rating
df_all['sentiment'] = df_all["rating"].apply(lambda x: 1 if x > 5 else 0)
```

In [44]:

```
df_train, df_test = train_test_split(df_all, test_size=0.33, random_state=4
2)
```

In [45]:

```
# https://github.com/corazzon/KaggleStruggle/blob/master/word2vec-nlp-tutori
al/tutorial-part-1.ipynb
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline

vectorizer = CountVectorizer(analyzer = 'word',
                             tokenizer = None,
                             preprocessor = None,
                             stop_words = None,
                             min_df = 2, # 토큰이 나타날 최소 문서 개수
                             ngram_range=(4, 4),
                             max_features = 20000
                            )
vectorizer
```

Out[45]:

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=20000, min_df=2,
        ngram_range=(4, 4), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

In [46]:

```
#https://stackoverflow.com/questions/28160335/plot-a-document-tfidf-2d-graph
pipeline = Pipeline([
    ('vect', vectorizer),
])
```

In [47]:

```
%time train_data_features = pipeline.fit_transform(df_train['review_clean']
)
%time test_data_features = pipeline.fit_transform(df_test['review_clean'])
CPU times: user 28.4 s, sys: 868 ms, total: 29.3 s
Wall time: 29.2 s
CPU times: user 15.5 s, sys: 396 ms, total: 15.9 s
Wall time: 15.9 s
```

In [48]:

```
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Bidirectional, LSTM, Batc
hNormalization, Dropout
from tensorflow.python.keras.preprocessing.sequence import pad_sequences
```

In [49]:

```
#Source code in keras 김태영'blog
# 0. Package
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense
import random

# 1. Dataset
y_train = df_train['sentiment']
```

```
y_test = df_test['sentiment']
solution = y_test.copy()

# 2. Model Structure
model = keras.models.Sequential()

model.add(keras.layers.Dense(200, input_shape=(20000,)))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.Dropout(0.5))

model.add(keras.layers.Dense(300))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.Dropout(0.5))

model.add(keras.layers.Dense(100, activation='relu'))
model.add(keras.layers.Dense(1, activation='sigmoid'))

# 3. Model compile
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accur
acy'])
```

In [50]:

```
model.summary()
```

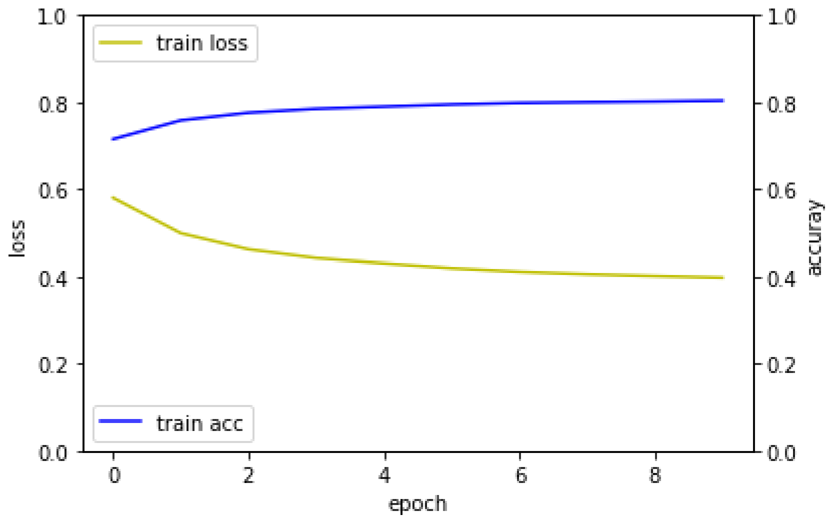
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 200)               4000200   
_________________________________________________________________
batch_normalization_1 (Batch (None, 200)               800       
_________________________________________________________________
activation_1 (Activation)    (None, 200)               0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 200)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 300)               60300     
_________________________________________________________________
batch_normalization_2 (Batch (None, 300)               1200      
_________________________________________________________________
activation_2 (Activation)    (None, 300)               0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 300)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 100)               30100     
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 101       
=================================================================
Total params: 4,092,701
Trainable params: 4,091,701
Non-trainable params: 1,000
_________________________________________________________________
```

In [51]:

```
# 4. Train model
hist = model.fit(train_data_features, y_train, epochs=10, batch_size=64)

# 5. Traing process
%matplotlib inline
import matplotlib.pyplot as plt

fig, loss_ax = plt.subplots()

acc_ax = loss_ax.twinx()

loss_ax.set_ylim([0.0, 1.0])
acc_ax.set_ylim([0.0, 1.0])

loss_ax.plot(hist.history['loss'], 'y', label='train loss')
acc_ax.plot(hist.history['acc'], 'b', label='train acc')

loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuray')

loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')

plt.show()

# 6. Evaluation
loss_and_metrics = model.evaluate(test_data_features, y_test, batch_size=32
)
print('loss_and_metrics : ' + str(loss_and_metrics))
```

```
Epoch 1/10
142075/142075 [==============================] - 39s 272us/step - loss:
0.5802 - acc: 0.7152
Epoch 2/10
142075/142075 [==============================] - 43s 302us/step - loss:
0.4990 - acc: 0.7582
Epoch 3/10
142075/142075 [==============================] - 35s 246us/step - loss:
0.4621 - acc: 0.7758
Epoch 4/10
142075/142075 [==============================] - 42s 299us/step - loss:
0.4422 - acc: 0.7848
Epoch 5/10
142075/142075 [==============================] - 43s 301us/step - loss:
0.4300 - acc: 0.7899
Epoch 6/10
142075/142075 [==============================] - 43s 300us/step - loss:
0.4189 - acc: 0.7948
Epoch 7/10
```

```
142075/142075 [==============================] - 43s 301us/step - loss:
0.4109 - acc: 0.7983
Epoch 8/10
142075/142075 [==============================] - 43s 301us/step - loss:
0.4053 - acc: 0.8000
Epoch 9/10
142075/142075 [==============================] - 43s 305us/step - loss:
0.4013 - acc: 0.8015
Epoch 10/10
142075/142075 [==============================] - 43s 302us/step - loss:
0.3976 - acc: 0.8036
```



```
69978/69978 [==============================] - 12s 165us/step
loss_and_metrics : [1.0738699619418919, 0.6469318928772125]
```

In [52]:

```
linkcode
sub_preds_deep = model.predict(test_data_features,batch_size=32)
```

### 4.3 Lightgbm

The To improve the low accuracy, we will use machine learning. First of all, this is the sentiment analysis model using only usefulCount.

In [53]:

```
from sklearn.metrics import roc_auc_score, precision_recall_curve, roc_curv
e, average_precision_score
from sklearn.model_selection import KFold
from lightgbm import LGBMClassifier
from sklearn.metrics import confusion_matrix

#folds = KFold(n_splits=5, shuffle=True, random_state=546789)
target = df_train['sentiment']
feats = ['usefulCount']

sub_preds = np.zeros(df_test.shape[0])
```

```
trn_x, val_x, trn_y, val_y = train_test_split(df_train[feats], target, test
_size=0.2, random_state=42)
feature_importance_df = pd.DataFrame()

clf = LGBMClassifier(
        n_estimators=2000,
        learning_rate=0.05,
        num_leaves=30,
        #colsample_bytree=.9,
        subsample=.9,
        max_depth=7,
        reg_alpha=.1,
        reg_lambda=.1,
        min_split_gain=.01,
        min_child_weight=2,
        silent=-1,
        verbose=-1,
        )

clf.fit(trn_x, trn_y,
        eval_set= [(trn_x, trn_y), (val_x, val_y)],
        verbose=100, early_stopping_rounds=100  #30
    )

sub_preds = clf.predict(df_test[feats])

fold_importance_df = pd.DataFrame()
fold_importance_df["feature"] = feats
fold_importance_df["importance"] = clf.feature_importances_
feature_importance_df = pd.concat([feature_importance_df, fold_importance_d
f], axis=0)
```

```
Training until validation scores don't improve for 100 rounds.
[100]  training's binary_logloss: 0.570406  valid_1's binary_logloss: 0.
572234
[200]  training's binary_logloss: 0.570206  valid_1's binary_logloss: 0.
572221
Early stopping, best iteration is:
[168]  training's binary_logloss: 0.570241  valid_1's binary_logloss: 0.
572209
```

In [54]:

```
solution = df_test['sentiment']
confusion_matrix(y_pred=sub_preds, y_true=solution)
```

Out[54]:

```
array([[    0, 21009],
       [    0, 48969]])
```

We will add variables for higher accuracy.

In [55]:

```
len_train = df_train.shape[0]
df_all = pd.concat([df_train,df_test])
del df_train, df_test;
gc.collect()
```

Out[55]:

```
3437
```

In [56]:

```
df_all['date'] = pd.to_datetime(df_all['date'])
df_all['day'] = df_all['date'].dt.day
df_all['year'] = df_all['date'].dt.year
df_all['month'] = df_all['date'].dt.month
```

In [57]:

```
from textblob import TextBlob
from tqdm import tqdm
reviews = df_all['review_clean']

Predict_Sentiment = []
for review in tqdm(reviews):
    blob = TextBlob(review)
    Predict_Sentiment += [blob.sentiment.polarity]
df_all["Predict_Sentiment"] = Predict_Sentiment
df_all.head()
```

```
100%|██████████| 212053/212053 [02:29<00:00, 1417.16it/s]
```

Out[57]:

| | uniqueID | drugName | condition | review | rating | date | usefulCount | review_clean | sentiment | day | year | month | Predict_Sentiment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 130185 | 66913 | Seroquel | Generalized Anxiety Disorde | "After trying nearly every SSRI on the market ... | 9 | 2010-11-26 | 102 | tri near everi ssri market place pristiq impro... | 1 | 26 | 2010 | 11 | 0.023958 |
| 155501 | 222222 | Fluconazole | Onychomycosis, Toenail | "This takes 6+ months, but did clear up a deca... | 9 | 2008-05-03 | 39 | take month clear decad long infect | 1 | 3 | 2008 | 5 | 0.025000 |

|  | uniqueID | drugName | condition | review | rating | date | usefulCount | review_clean | sentiment | day | year | month | Predict_Sentiment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 75325 | 75807 | Temazepam | Insomnia | "Worked for awhile pretty well but then went b... | 2 | 2016-04-21 | 15 | work awhil pretti well went back mayb get hour... | 0 | 21 | 2016 | 4 | -0.125000 |
| 19174 | 229747 | Ketamine | Pain | "I was given this after surgery for pain as I ... | 1 | 2016-04-16 | 19 | given surgeri pain morphin tri drug sort omg w... | 0 | 16 | 2016 | 4 | 0.166667 |
| 209735 | 102495 | Aripiprazole | Major Depressive Disorde | "Abilify served me well over a two month perio... | 9 | 2016-12-16 | 5 | abilifi serv well two month period antidepress... | 1 | 16 | 2016 | 12 | 0.038889 |

In [58]:

```
np.corrcoef(df_all["Predict_Sentiment"], df_all["rating"])
```

Out[58]:

```
array([[1.        , 0.25709864],
       [0.25709864, 1.        ]])
```

In [59]:

```
np.corrcoef(df_all["Predict_Sentiment"], df_all["sentiment"])
```

Out[59]:

```
array([[1.        , 0.23518272],
       [0.23518272, 1.        ]])
```

In [60]:

```
reviews = df_all['review']

Predict_Sentiment = []
for review in tqdm(reviews):
    blob = TextBlob(review)
    Predict_Sentiment += [blob.sentiment.polarity]
df_all["Predict_Sentiment2"] = Predict_Sentiment
100%|██████████| 212053/212053 [03:49<00:00, 924.00it/s]
```

In [61]:

```
np.corrcoef(df_all["Predict_Sentiment2"], df_all["rating"])
```

Out[61]:

```
array([[1.        , 0.34831213],
       [0.34831213, 1.        ]])
```

In [62]:

```
np.corrcoef(df_all["Predict_Sentiment2"], df_all["sentiment"])
```

Out[62]:

```
array([[1.        , 0.31714515],
       [0.31714515, 1.        ]])
```

In [63]:

```
#문장길이 (줄바꿈표시가 몇번나왔는지 셈)
df_all['count_sent']=df_all["review"].apply(lambda x: len(re.findall("\n",s
tr(x)))+1)

#Word count in each comment:(단어갯수)
df_all['count_word']=df_all["review_clean"].apply(lambda x: len(str(x).spli
t()))

#Unique word count(unique한 단어 갯수)
df_all['count_unique_word']=df_all["review_clean"].apply(lambda x: len(set(
str(x).split())))

#Letter count(리뷰길이)
df_all['count_letters']=df_all["review_clean"].apply(lambda x: len(str(x)))

#punctuation count(특수문자)
df_all["count_punctuations"] = df_all["review"].apply(lambda x: len([c for
c in str(x) if c in string.punctuation]))

#upper case words count(전부다 대문자인 단어 갯수)
df_all["count_words_upper"] = df_all["review"].apply(lambda x: len([w for w
in str(x).split() if w.isupper()]))

#title case words count(첫글자가 대문자인 단어 갯수)
df_all["count_words_title"] = df_all["review"].apply(lambda x: len([w for w
in str(x).split() if w.istitle()]))

#Number of stopwords(불용어 갯수)
```

```
df_all["count_stopwords"] = df_all["review"].apply(lambda x: len([w for w i
n str(x).lower().split() if w in stops]))

#Average length of the words(평균단어길이)
df_all["mean_word_len"] = df_all["review_clean"].apply(lambda x: np.mean([l
en(w) for w in str(x).split()]))
```
We added a season variable.

```
                                                                    In [64]:
df_all['season'] = df_all["month"].apply(lambda x: 1 if ((x>2) & (x<6)) els
e(2 if (x>5) & (x<9) else (3 if (x>8) & (x<12) else 4)))
```
We normalized useful count.

```
df_train = df_all[:len_train]
df_test = df_all[len_train:]

from sklearn.metrics import roc_auc_score, precision_recall_curve, roc_curv
e, average_precision_score
from sklearn.model_selection import KFold
from lightgbm import LGBMClassifier

#folds = KFold(n_splits=5, shuffle=True, random_state=546789)
target = df_train['sentiment']
feats = ['usefulCount','day','year','month','Predict_Sentiment','Predict_Se
ntiment2', 'count_sent',
 'count_word', 'count_unique_word', 'count_letters', 'count_punctuations',
 'count_words_upper', 'count_words_title', 'count_stopwords', 'mean_word_le
n', 'season']

sub_preds = np.zeros(df_test.shape[0])

trn_x, val_x, trn_y, val_y = train_test_split(df_train[feats], target, test
_size=0.2, random_state=42)
feature_importance_df = pd.DataFrame()

clf = LGBMClassifier(
        n_estimators=10000,
        learning_rate=0.10,
        num_leaves=30,
        #colsample_bytree=.9,
        subsample=.9,
        max_depth=7,
        reg_alpha=.1,
        reg_lambda=.1,
        min_split_gain=.01,
        min_child_weight=2,
        silent=-1,
        verbose=-1,
        )

clf.fit(trn_x, trn_y,
        eval_set= [(trn_x, trn_y), (val_x, val_y)],
        verbose=100, early_stopping_rounds=100  #30
```

```
    )

sub_preds = clf.predict(df_test[feats])

fold_importance_df = pd.DataFrame()
fold_importance_df["feature"] = feats
fold_importance_df["importance"] = clf.feature_importances_
feature_importance_df = pd.concat([feature_importance_df, fold_importance_d
f], axis=0)
```

```
Training until validation scores don't improve for 100 rounds.
[100]	training's binary_logloss: 0.481279	valid_1's binary_logloss: 0.
494978
[200]	training's binary_logloss: 0.464017	valid_1's binary_logloss: 0.
489836
[300]	training's binary_logloss: 0.448687	valid_1's binary_logloss: 0.
485795
[400]	training's binary_logloss: 0.434477	valid_1's binary_logloss: 0.
481708
[500]	training's binary_logloss: 0.422172	valid_1's binary_logloss: 0.
478113
[600]	training's binary_logloss: 0.410153	valid_1's binary_logloss: 0.
4747
[700]	training's binary_logloss: 0.399379	valid_1's binary_logloss: 0.
471887
[800]	training's binary_logloss: 0.388203	valid_1's binary_logloss: 0.
468779
[900]	training's binary_logloss: 0.377657	valid_1's binary_logloss: 0.
465732
[1000]	training's binary_logloss: 0.367101	valid_1's binary_logloss: 0.
462981
[1100]	training's binary_logloss: 0.357608	valid_1's binary_logloss: 0.
460332
[1200]	training's binary_logloss: 0.348188	valid_1's binary_logloss: 0.
457833
[1300]	training's binary_logloss: 0.339333	valid_1's binary_logloss: 0.
455505
[1400]	training's binary_logloss: 0.3308	valid_1's binary_logloss: 0.
452876
[1500]	training's binary_logloss: 0.322571	valid_1's binary_logloss: 0.
450271
[1600]	training's binary_logloss: 0.314523	valid_1's binary_logloss: 0.
447521
[1700]	training's binary_logloss: 0.30683	valid_1's binary_logloss: 0.
445272
[1800]	training's binary_logloss: 0.299902	valid_1's binary_logloss: 0.
442985
[1900]	training's binary_logloss: 0.292878	valid_1's binary_logloss: 0.
440813
```

```
[2000]  training's binary_logloss: 0.286391  valid_1's binary_logloss: 0.
439022
[2100]  training's binary_logloss: 0.279798  valid_1's binary_logloss: 0.
437461
[2200]  training's binary_logloss: 0.273216  valid_1's binary_logloss: 0.
43553
[2300]  training's binary_logloss: 0.267114  valid_1's binary_logloss: 0.
433628
[2400]  training's binary_logloss: 0.260879  valid_1's binary_logloss: 0.
432172
[2500]  training's binary_logloss: 0.255223  valid_1's binary_logloss: 0.
430448
[2600]  training's binary_logloss: 0.249595  valid_1's binary_logloss: 0.
428833
[2700]  training's binary_logloss: 0.244094  valid_1's binary_logloss: 0.
427271
[2800]  training's binary_logloss: 0.238678  valid_1's binary_logloss: 0.
425821
[2900]  training's binary_logloss: 0.233193  valid_1's binary_logloss: 0.
424003
[3000]  training's binary_logloss: 0.22774   valid_1's binary_logloss: 0.
422476
[3100]  training's binary_logloss: 0.222546  valid_1's binary_logloss: 0.
421128
[3200]  training's binary_logloss: 0.217182  valid_1's binary_logloss: 0.
419837
[3300]  training's binary_logloss: 0.212582  valid_1's binary_logloss: 0.
418584
[3400]  training's binary_logloss: 0.207872  valid_1's binary_logloss: 0.
417177
[3500]  training's binary_logloss: 0.20318   valid_1's binary_logloss: 0.
415465
[3600]  training's binary_logloss: 0.19881   valid_1's binary_logloss: 0.
414296
[3700]  training's binary_logloss: 0.194574  valid_1's binary_logloss: 0.
413357
[3800]  training's binary_logloss: 0.190084  valid_1's binary_logloss: 0.
412226
[3900]  training's binary_logloss: 0.185752  valid_1's binary_logloss: 0.
41124
[4000]  training's binary_logloss: 0.181641  valid_1's binary_logloss: 0.
410157
[4100]  training's binary_logloss: 0.177666  valid_1's binary_logloss: 0.
409236
[4200]  training's binary_logloss: 0.173646  valid_1's binary_logloss: 0.
408272
[4300]  training's binary_logloss: 0.169929  valid_1's binary_logloss: 0.
407405
[4400]  training's binary_logloss: 0.165956  valid_1's binary_logloss: 0.
406168
```

```
[4500]	training's binary_logloss: 0.16243	valid_1's binary_logloss: 0.
405496
[4600]	training's binary_logloss: 0.158864	valid_1's binary_logloss: 0.
404793
[4700]	training's binary_logloss: 0.155165	valid_1's binary_logloss: 0.
403937
[4800]	training's binary_logloss: 0.151448	valid_1's binary_logloss: 0.
403068
[4900]	training's binary_logloss: 0.148384	valid_1's binary_logloss: 0.
402691
[5000]	training's binary_logloss: 0.145498	valid_1's binary_logloss: 0.
4019
[5100]	training's binary_logloss: 0.142466	valid_1's binary_logloss: 0.
401213
[5200]	training's binary_logloss: 0.139651	valid_1's binary_logloss: 0.
400643
[5300]	training's binary_logloss: 0.136603	valid_1's binary_logloss: 0.
400254
[5400]	training's binary_logloss: 0.133609	valid_1's binary_logloss: 0.
399837
[5500]	training's binary_logloss: 0.13088	valid_1's binary_logloss: 0.
399404
[5600]	training's binary_logloss: 0.127999	valid_1's binary_logloss: 0.
398777
[5700]	training's binary_logloss: 0.125435	valid_1's binary_logloss: 0.
398375
[5800]	training's binary_logloss: 0.122862	valid_1's binary_logloss: 0.
397779
[5900]	training's binary_logloss: 0.120402	valid_1's binary_logloss: 0.
397536
[6000]	training's binary_logloss: 0.117785	valid_1's binary_logloss: 0.
397047
[6100]	training's binary_logloss: 0.115336	valid_1's binary_logloss: 0.
396875
[6200]	training's binary_logloss: 0.11312	valid_1's binary_logloss: 0.
396779
[6300]	training's binary_logloss: 0.110779	valid_1's binary_logloss: 0.
396631
[6400]	training's binary_logloss: 0.108439	valid_1's binary_logloss: 0.
396546
[6500]	training's binary_logloss: 0.106091	valid_1's binary_logloss: 0.
39677
Early stopping, best iteration is:
[6409]	training's binary_logloss: 0.108243	valid_1's binary_logloss: 0.
396526
```

In [67]:

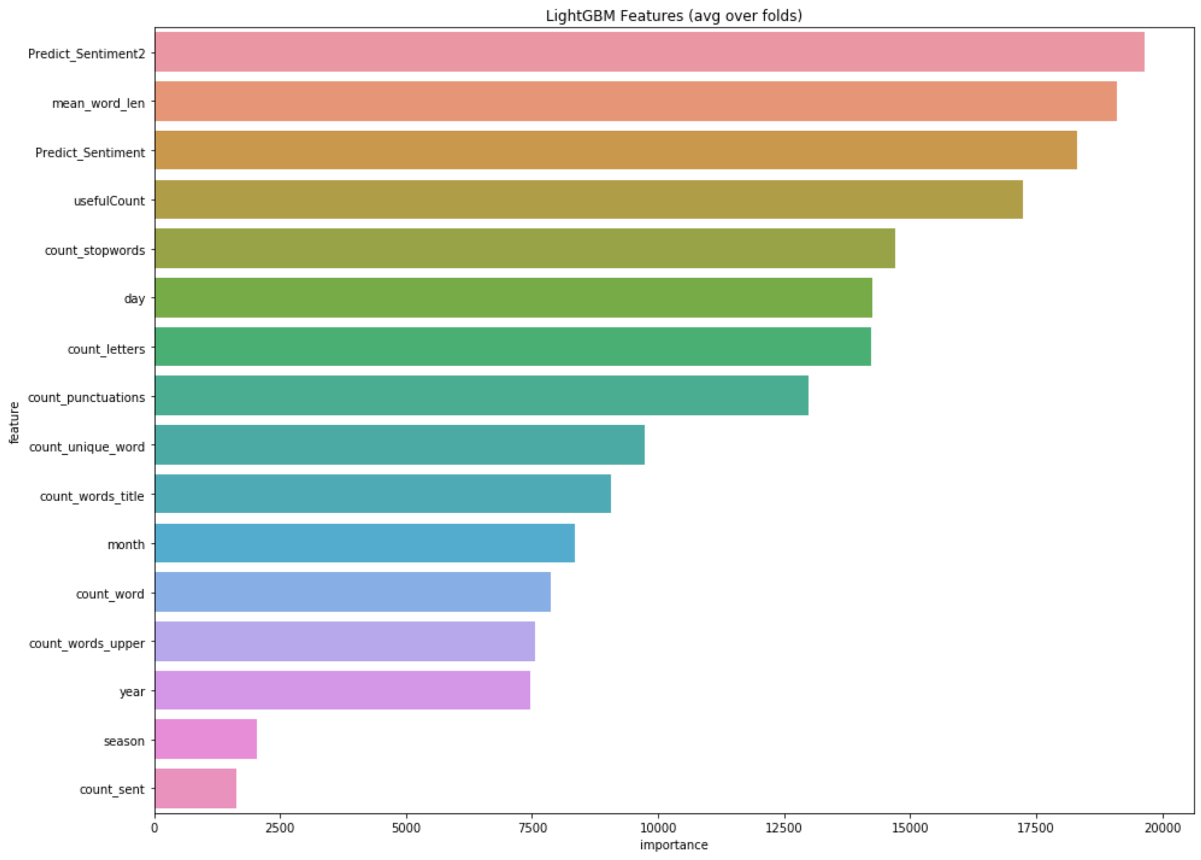
```
confusion_matrix(y_pred=sub_preds, y_true=solution)
```

Out[67]:

```
array([[13698,  7311],
       [ 3664, 45305]])
```

In [68]:

```
cols = feature_importance_df[["feature", "importance"]].groupby("feature").
mean().sort_values(
    by="importance", ascending=False)[:50].index

best_features = feature_importance_df.loc[feature_importance_df.feature.isi
n(cols)]

plt.figure(figsize=(14,10))
sns.barplot(x="importance", y="feature", data=best_features.sort_values(by=
"importance", ascending=False))
plt.title('LightGBM Features (avg over folds)')
plt.tight_layout()
plt.savefig('lgbm_importances.png')
```



## 5. Conclusion

Our team set the topic as recommending the right medicine for the patient's condition with reviews and proceeded the project according to the topic with the data exploration, data preprocessing and modeling. In the data exploration section, we looked at the forms of data using visualization techniques and statistical techniques. We also looked for n-grams that can best represent emotions, and the relationship with date and rating. The next step was to preprocess the data according to the topic we set, such as removing the condition that has only one drug for recommendation. In the process of modeling, we used deep learning model with n-gram, and additionally used a machine learning model called Lightgbm to overcome the limitation of natural language processing. In addition, we conducted emotional analysis using emotional word dictionary to overcome limitations of package formed with movie data. In addition, we nomalized usefulcount by condition for better reliability. These steps allowed us to calculate the final predicted value and recommend the appropriate drug for each condition according to the order of the value.

In conclusion, these are the limitations we had during the project.

Sentiment analysis using sentiment word dictionary has low reliability when the number of positive and negative words is small. For example, if there are 0 positive words and 1 negative word, it is classified as negative. Therefore, if the number of sentiment words is 5 or less, we could exclude the observations.

To ensure the reliability of the predicted values, we normalized usefulCount and multiplied it to the predicted values. However, usefulCount may tend to be higher for older reviews as the number of cumulated site visitors increases. Therefore, we should have also considered time when normalizing usefulCount.

If the emotion is positive, the reliability should be increased to the positive side, and if it is negative, the reliability should be increased toward the negative side. However, we simply multiplied the usefulCount for reliability and did not consider this part. So we should have multiplied considering the sign of usefulCount according to different kinds of emotion.