DATA MINING
TECHNICAL REPORT

DRUG RECOMMENDER SYSTEM

-

Rully Meidyta (2206130795)

PROGRAM STUDI MAGISTER MATEMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS INDONESIA

DEPOK

2023

TABLE OF CONTENTS

1. Introduction

The field of healthcare has witnessed significant advancements in recent years, particularly in the area of drug recommendation systems. With the ever-increasing number of available drugs and the complexity of medical conditions, there is a growing need for personalized and effective drug recommendations to optimize patient care. Drug recommendation systems leverage the power of data analysis and machine learning techniques to provide tailored suggestions for appropriate medications based on individual patient characteristics and medical histories.

The objective of this report is to present a comprehensive analysis and evaluation of drug recommendation systems, focusing on their importance, benefits, and challenges in the healthcare domain. By exploring the existing literature and reviewing state-of-the-art approaches, we aim to contribute to the understanding of drug recommendation systems and provide insights into their potential applications and limitations.

The significance of drug recommendation systems lies in their ability to improve the efficiency and safety of medical treatments. By considering various factors such as patient demographics, medical history, symptoms, and drug interactions, these systems can generate personalized recommendations that enhance treatment outcomes and reduce the risk of adverse effects. Moreover, drug recommendation systems have the potential to streamline the decision-making process for healthcare professionals, enabling them to make informed prescribing choices based on evidence-based suggestions.

However, the development and implementation of effective drug recommendation systems are not without challenges. The complexity of medical data, privacy concerns, and the need for accurate and up-to-date information pose significant hurdles. Additionally, ensuring the transparency and interpretability of the recommendations generated by these systems is crucial to gain trust from healthcare providers and patients.

In conclusion, drug recommendation systems have the potential to revolutionize the field of healthcare by providing personalized and evidence-based suggestions for drug prescriptions. By addressing the challenges and leveraging the advancements in data analytics and machine learning, these systems can play a pivotal role in enhancing patient care and optimizing medical treatments. The subsequent sections of this report will delve into a detailed analysis and evaluation of drug recommendation systems, aiming to contribute to the understanding and advancement of this important area in healthcare.

2. Background

Drug recommendation systems play a vital role in healthcare by providing personalized suggestions for appropriate medications based on individual patient characteristics. These systems leverage advanced algorithms and machine learning techniques to analyze patient data and generate tailored recommendations. In this background section, we will provide an overview of drug recommendation systems and discuss the use of logistic regression, Stochastic Gradient Descent (SGD), and Multinomial Naive Bayes as popular algorithms in this domain.

2.1 Logistic Regression:

Logistic regression is a widely used algorithm in drug recommendation systems. It is a statistical model that is particularly suitable for binary classification tasks, such as predicting whether a specific drug is suitable for a patient or not. Logistic regression models the relationship between input features (e.g., patient demographics, medical history) and the probability of a particular drug being recommended. By estimating the coefficients of the model, logistic regression can predict the likelihood of a positive drug recommendation for a given patient.

2.2 Stochastic Gradient Descent (SGD):

SGD is an optimization algorithm commonly used in machine learning, including drug recommendation systems. It is particularly effective when dealing with large datasets and high-dimensional feature spaces, which are often encountered in healthcare applications. SGD iteratively updates the model parameters based on a subset of the training data, making it computationally efficient. In drug recommendation systems, SGD can be applied to learn the optimal weights for the features and make predictions on drug recommendations based on these learned weights.

2.3 Multinomial Naive Bayes:

Multinomial Naive Bayes is another popular algorithm for drug recommendation systems, especially when dealing with categorical or discrete features. It is based on Bayes' theorem and assumes independence between the features. Multinomial Naive Bayes calculates the probability of a drug recommendation given the patient's features and selects the drug with the highest probability. This algorithm is particularly useful when dealing with textual data or discrete variables related to patient characteristics.

These algorithms, logistic regression, SGD, and multinomial Naive Bayes, provide different approaches to drug recommendation within a recommendation system. They have been widely studied and applied in the field, showcasing their effectiveness in generating accurate and personalized drug recommendations based on patient data.

In the following sections, we will delve deeper into the methodology and implementation of these algorithms in drug recommendation systems, exploring their strengths, limitations, and potential applications. Additionally, we will discuss evaluation metrics and considerations for the performance assessment of these recommendation systems, aiming to provide a comprehensive understanding of their usage and effectiveness in the domain of drug recommendation.

3. Data Collection and Preprocessing

```
import numpy as np
import pandas as pd
import os
import spacy
import en_core_web_sm
from spacy.lang.en import English
from spacy.lang.en.stop_words import STOP_WORDS
import string
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.feature_extraction import DictVectorizer
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer,HashingVectorizer
from sklearn.base import TransformerMixin
from sklearn.pipeline import Pipeline
!pip install vaderSentiment
import vaderSentiment
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

Import several libraries and prepares the environment for sentiment analysis using logistic regression model.

```
!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')
```

The above code is used to set up and authenticate access to Google Drive within the Google Colab environment.

```
# preprocessing the data file
#  read the data
df1 = pd.read_csv("/content/gdrive/MyDrive/Colab Notebook/drugsComTrain_raw.tsv",sep='\t')
df2 = pd.read_csv("/content/gdrive/MyDrive/Colab Notebook/drugsComTest_raw.tsv",sep='\t')
# combine two file
df = pd.concat([df1, df2])
df
# rename the cols
df.columns = ['ID','drug name','condition','review','rating','date','useful count']
```

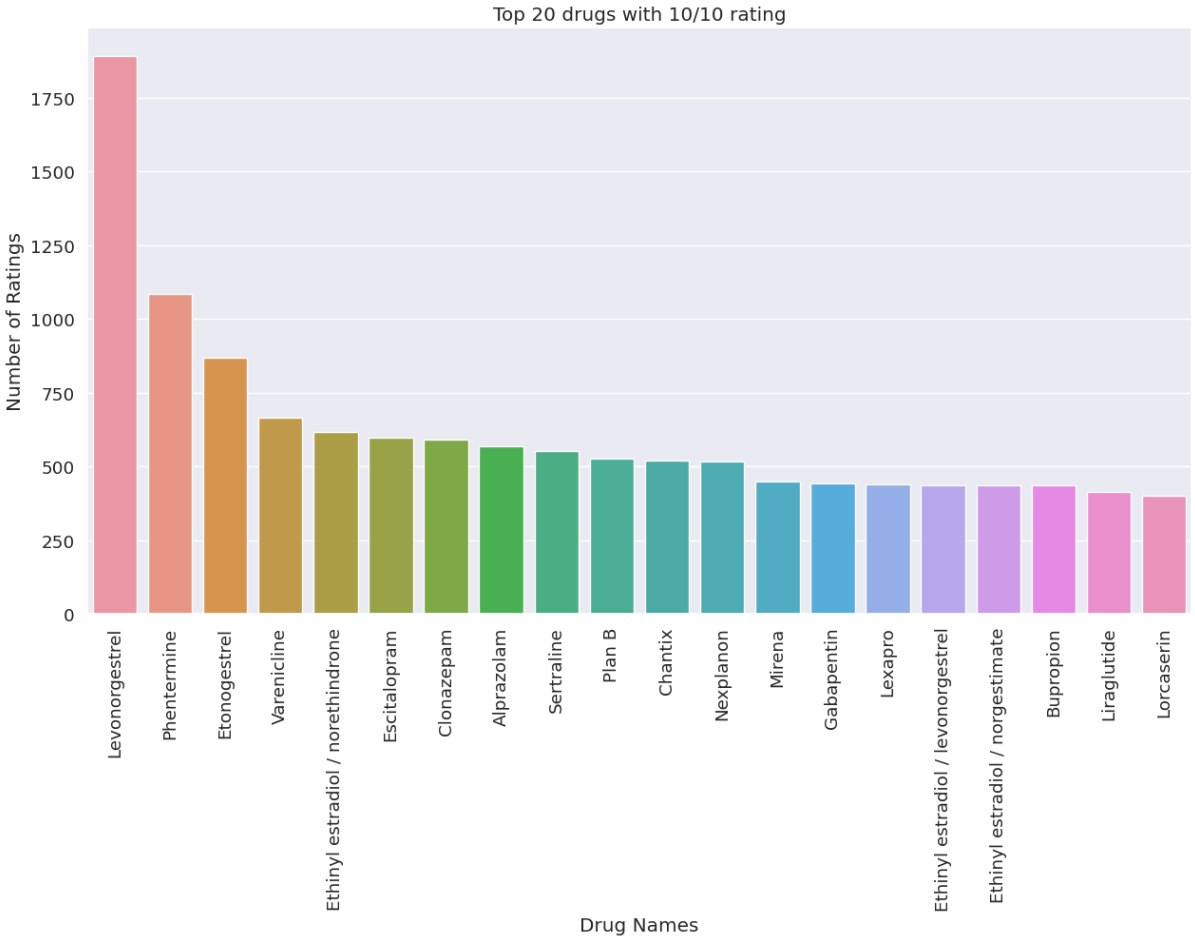
Import the dataset into the project using the read module in Pandas. The above code reads two datasets that have been divided into train and test sets. Then, both datasets are combined using Pandas concat function, and the column names of the merged dataset are changed.

```
print(df1.shape)
print(df2.shape)
print(df.shape)
```

```
(161297, 7)
(53766, 7)
(215063, 7)
```

```
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);
```

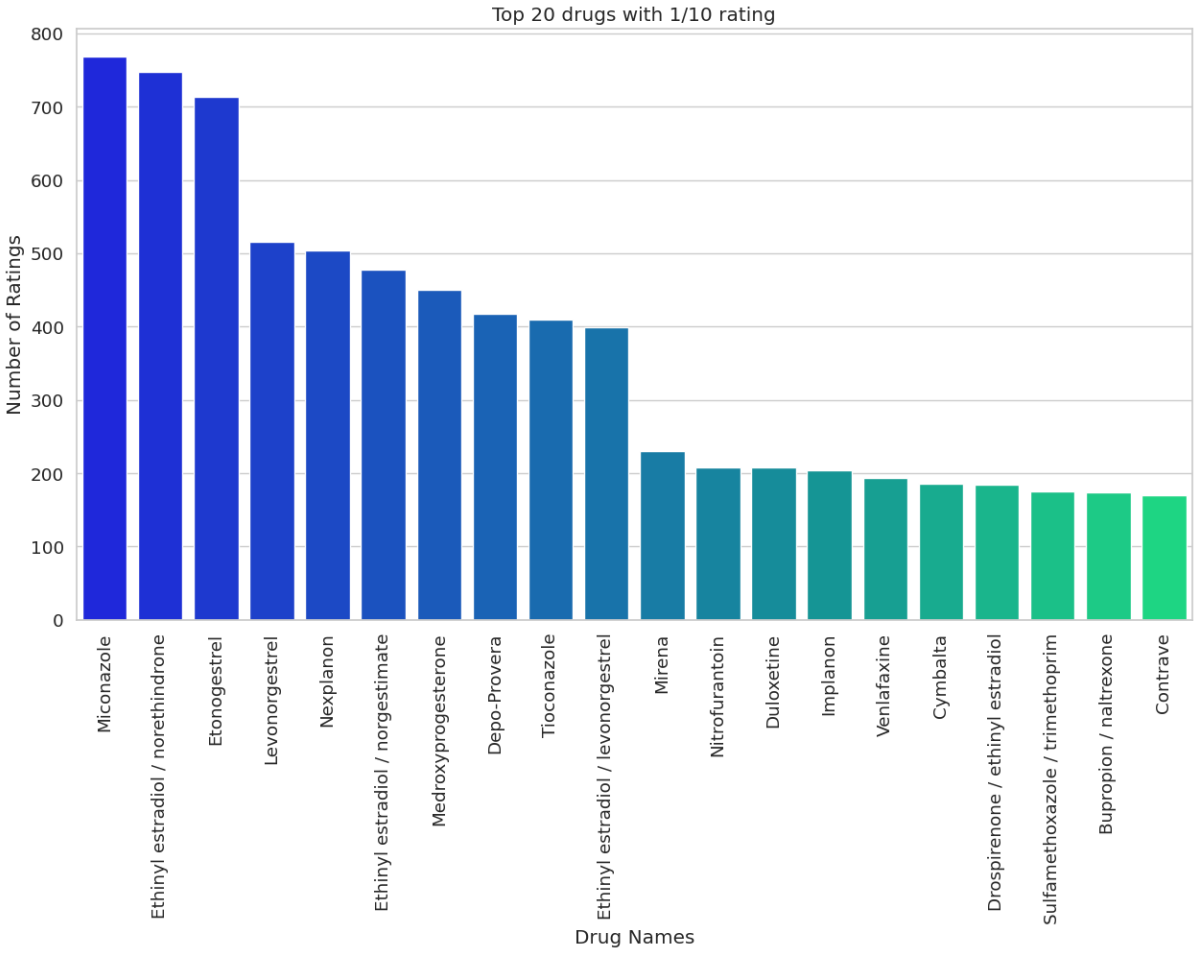
Import the seaborn and matplotlib libraries to visualize the data. Set the parameters first, then count how many ratings have a value of 10. Next, create a bar plot with the x-axis as drugname and the y-axis as drug_rating.



This is the previously created bar plot. Here, it can be observed that there are the top 20 drugs with the highest total rating of 10. Among them, the drug Levonorgestrel has the highest total rating value of over 1750, followed by the drug Phentermine and others.

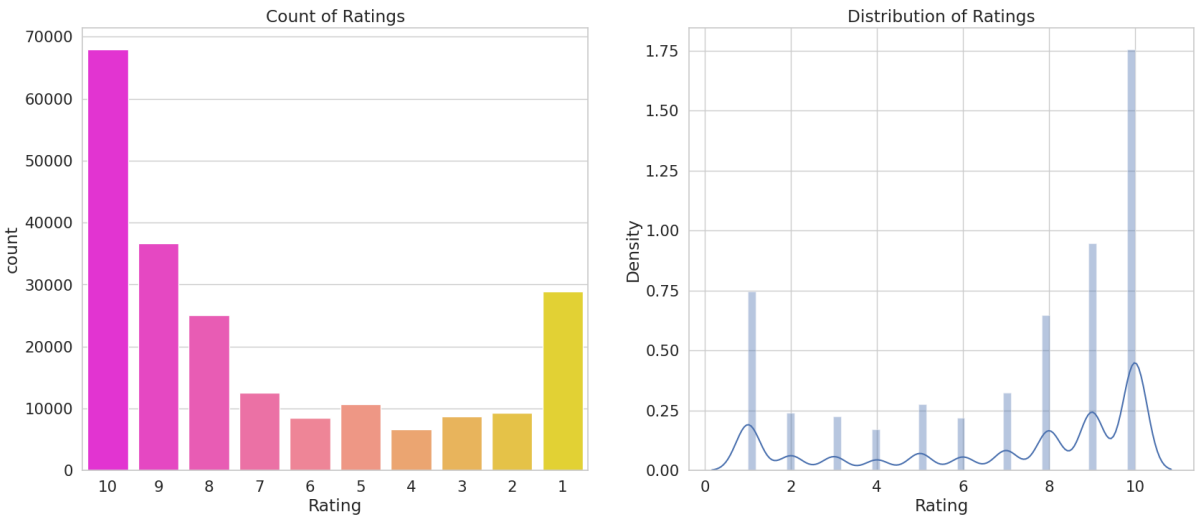
```
# Setting the Parameter
sns.set(font_scale = 1.2, style = 'whitegrid')
plt.rcParams['figure.figsize'] = [15, 8]

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);
```

Import the seaborn and matplotlib libraries to visualize the data. Set the parameters first, then count how many ratings have a value of 1. Next, create a bar plot with the x-axis as drugname and the y-axis as drug_rating.



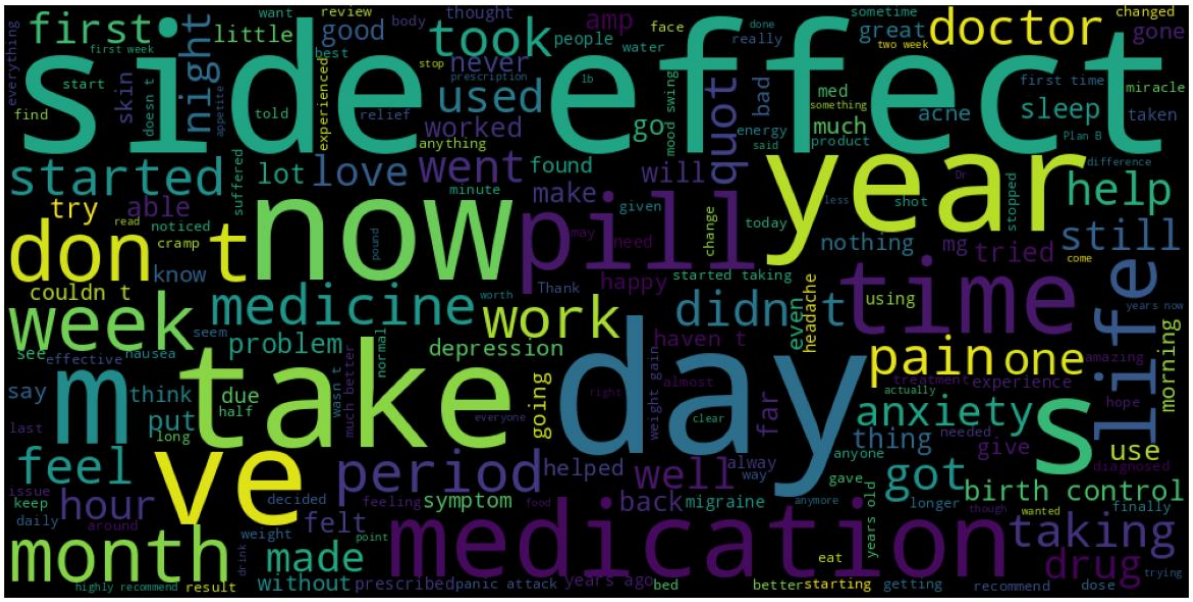
This is the previously created bar plot. Here, it can be observed that there are the top 20 drugs with the highest total rating of 1. Among them, the drug Miconacole has the highest total rating value of over 750, followed by the drug Ethinyl estradiol and others.

```
# 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(x='rating', data=df, 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")
```



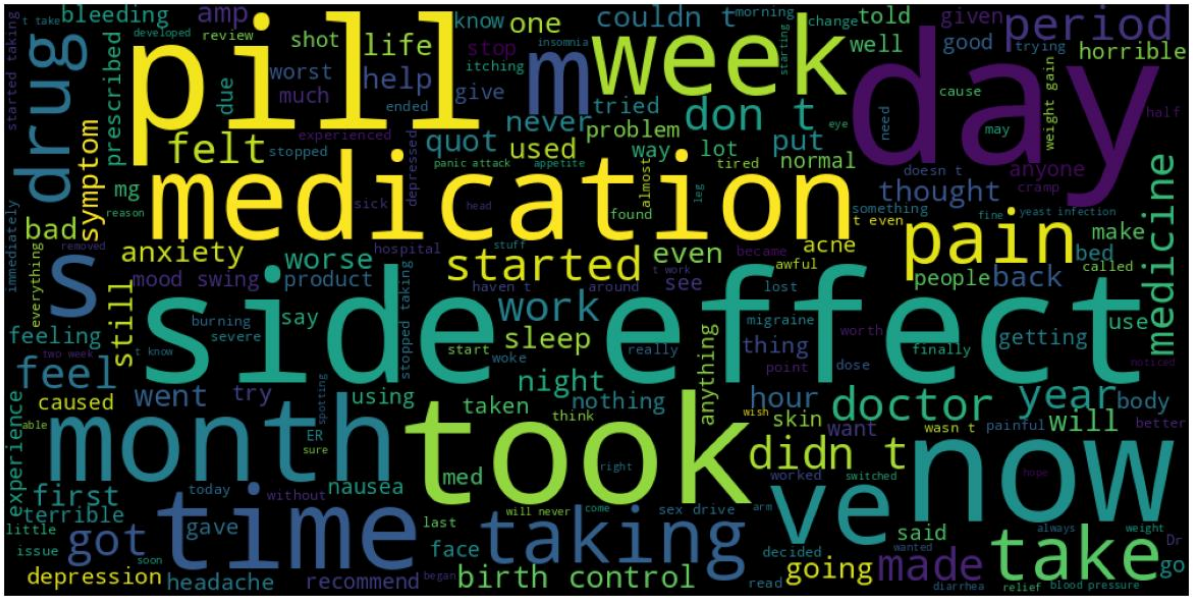
Display the total rating and rating distribution using a countplot.

```
#!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');
```



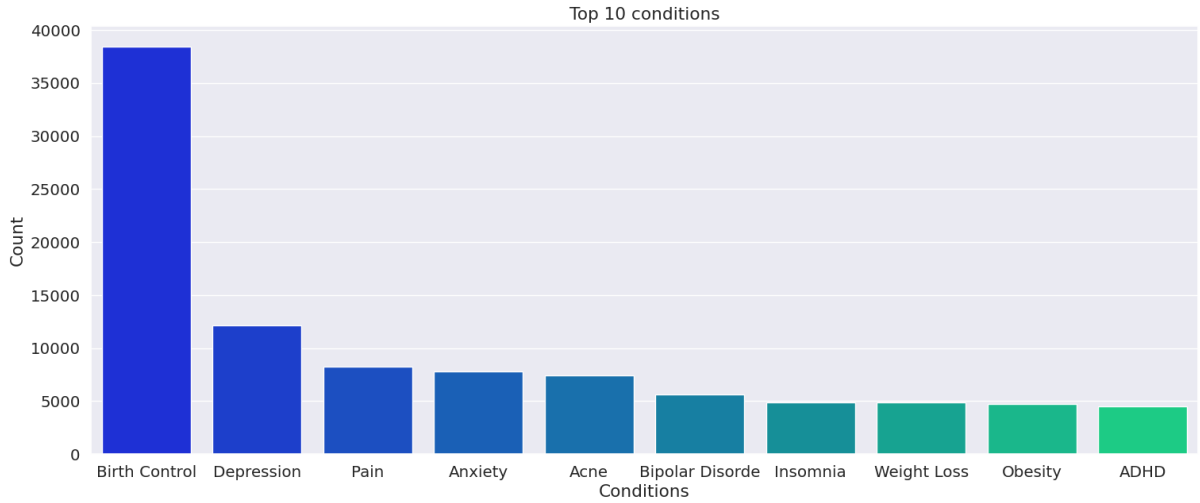
Display a word cloud for reviews that have a rating of 1. Here, several large words indicate the frequency of their appearance, such as "side", "effect", "now", "year", "take", and "day".

```
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');
```



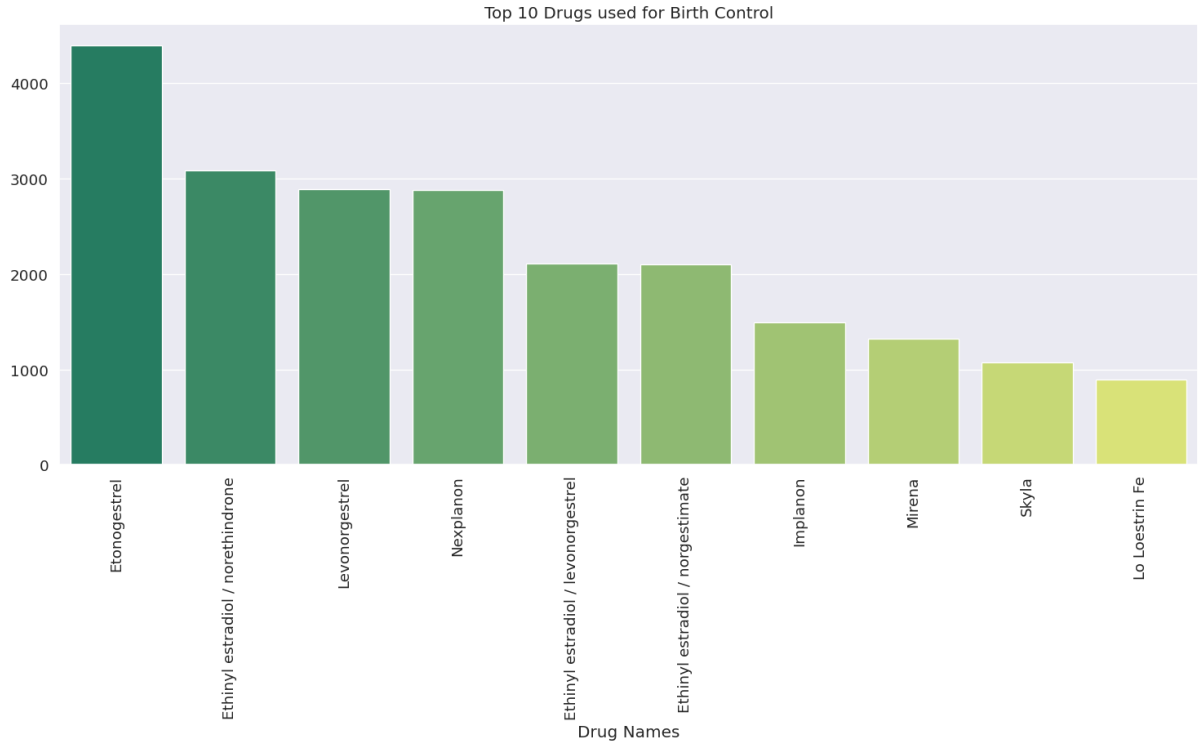
Display a word cloud for reviews that have a rating of 1. Here, several large words indicate the frequency of their appearance, such as "pill", "side", "effect", "took", "now", and "month".

```
# 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");
```



This bar plot displays the top 10 conditions with the highest total number of affected individuals. The highest value is for the condition "birth control".

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

sns_ = sns.barplot(x = df_cond_birth.index, y = df_cond_birth.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);
```



This bar plot displays the top 10 drugs that are used for birth control.

```
[232] df2 = df[df['useful count'] > 10]
```

```
df_condition = df2.groupby(['condition'])['drug name'].nunique().sort_values(ascending=False)
df_condition = pd.DataFrame(df_condition).reset_index()
df_condition.tail(20)
```

	condition	drug name
706	64 users found this comment helpful.	1
707	92 users found this comment helpful.	1
708	Gastritis/Duodenitis	1
709	Esophageal Variceal Hemorrhage Prophylaxis	1
710	98 users found this comment helpful.	1
711	Severe Mood Dysregulation	1
712	Short Stature	1
713	Short Stature for Age	1
714	Meningitis	1
715	Skin Disinfection, Preoperative	1
716	Melanoma	1
717	Gallbladder Disease	1
718	Spondylolisthesis	1
719	Small Fiber Neuropathy	1
720	95 users found this comment helpful.	1
721	94 users found this comment helpful.	1
722	Somat	1
723	Performance Anxiety	1
724	Malaria	1
725	100 users found this comment helpful.	1

Calculate the total "useful count" that has a value greater than 10 based on the conditions and drugs used.

```
df_condition_1 = df_condition[df_condition['drug name'] == 1].reset_index()

all_list = set(df.index)

# deleting them
condition_list = []
for i,j in enumerate(df['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 = df.iloc[list(new_idx)].reset_index()
del df['index']
```

Remove the "condition" values that have only one drug associated with them, and then reset the index to ensure that the index is sorted after removing some rows of data.

```
[236] # removing the conditions with <span> in it.

all_list = set(df.index)
span_list = []
for i,j in enumerate(df['condition']):
    if "</span>" in str(j):
        span_list.append(i)
new_idx = all_list.difference(set(span_list))
df = df.iloc[list(new_idx)].reset_index()
del df['index']
```

```
import re
from bs4 import BeautifulSoup
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from nltk.stem.porter import PorterStemmer
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
```

Remove the "condition" values that contain "", and then reset the index to ensure that the index is sorted after removing some rows of data.

Import the nltk library, which includes the stopwords dictionary used for text-based data processing.

```
[238] # removing some stopwords from the list of stopwords as they are important for drug recommendation

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

      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)
```

Removing some stopwords from the list of stopwords that are important for drug recommendations.

```
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))
```

```
# create a list of stopwords
nlp = spacy.load('en_core_web_sm')
stop_words = spacy.lang.en.stop_words.STOP_WORDS
parser = English()
punctuations = string.punctuation
# Creating our tokenizer function
def spacy_tokenizer(sentence):
    # Creating our token object, which is used to create documents with linguistic annotations.
    mytokens = parser(sentence)

    # Lemmatizing each token and converting each token into lowercase
    mytokens = [ word.lemma_.lower().strip() if word.lemma_ != "-PRON-" else word.lower_ for word in mytokens ]

    # Removing stop words
    mytokens = [ word for word in mytokens if word not in stop_words and word not in punctuations ]

    # return preprocessed list of tokens
    return mytokens
```

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

	ID	drug name	condition	review	rating	date	useful count	review_clean
0	206461	Valsartan	Left Ventricular Dysfunction	"It has no side effect, I take it in combinati...	9.0	May 20, 2012	27	no side effect take combin bystol mg fish oil
1	95260	Guanfacine	ADHD	"My son is halfway through his fourth week of ...	8.0	April 27, 2010	192	son halfway fourth week intuniv becam concern ...
2	92703	Lybrel	Birth Control	"I used to take another oral contraceptive, wh...	5.0	December 14, 2009	17	use take anoth oral contracept pill cycl happi...
3	138000	Ortho Evra	Birth Control	"This is my first time using any form of birth...	8.0	November 3, 2015	10	first time use form birth control glad went pa...
4	35696	Buprenorphine / naloxone	Opiate Dependence	"Suboxone has completely turned my life around...	9.0	November 27, 2016	37	suboxon complet turn life around feel healthie...

Display the dataframe df with the added column review_clean.

```
[241] 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"
```

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

      0.9935
```

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

      -0.9973
```

Display the maximum and minimum value of the column vaderReviewScore.

```
[245] #Normalizing "vaderReviewScore"
      criteria = [df['vaderReviewScore'].between(-0.997, -0.799), df['vaderReviewScore'].between(-0.798, -0.601), df['vaderReviewScore'].between(-0.600, 0.
      values = [1, 2, 3,4,5,6,7,8,9,10]

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

```
[247] #Final Normalized Score combining Rating and normalVaderScore
      df['meanNormalizedScore'] = (df['rating'] + df['normalVaderScore'])/2
      df.head()
```

	ID	drug name	condition	review	rating	date	useful count	review_clean	vaderReviewScore	vaderSentiment	
0	206461	Valsartan	Left Ventricular Dysfunction	"It has no side effect, I take it in combinati...	9.0	May 20, 2012	27	no side effect take combin bystol mg fish oil	-0.2960	1	negative
1	95260	Guanfacine	ADHD	"My son is halfway through his fourth week of ...	8.0	April 27, 2010	192	son halfway fourth week intuniv becam concern ...	0.6929	2	positive
2	92703	Lybrel	Birth Control	"I used to take another oral contraceptive, wh...	5.0	December 14, 2009	17	use take anoth oral contracept pill cycl happi...	0.2732	2	positive
3	138000	Ortho Evra	Birth Control	"This is my first time using any form of birth...	8.0	November 3, 2015	10	first time use form birth control glad went pa...	0.4199	2	positive
4	35696	Buprenorphine / naloxone	Opiate Dependence	"Suboxone has completely turned my life around...	9.0	November 27, 2016	37	suboxon complet turn life around feel healthie...	0.8934	2	positive

Display the dataframe df with the added column meanNormalizedScore that has been normalized.

```
#Sorting Data based on Condition and then by drug
grouped = df.groupby(['condition','drug name']).agg({'meanNormalizedScore' : ['mean']})
grouped.to_csv('Grouped_Drug_Recommendation_Normalized')
grouped.head(10)
```

		mean
condition	drug name	
ADHD	Adderall	6.990625
	Adderall XR	7.172897
	Adzenys XR-ODT	6.805556
	Amantadine	5.666667
	Amphetamine	6.522727
	Amphetamine / dextroamphetamine	6.941926
	Aptensio XR	5.833333
	Armodafinil	6.937500
	Atomoxetine	5.182266
	Bupropion	6.879310

Display sort data based on Condition and then by drugs.

```
[249] #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)
1.0    120085
0.0     39723
Name: Review_Sentiment, dtype: int64
```

```
import six
import sys
sys.modules['sklearn.externals.six'] = six
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_class='auto')
#clf3 = SVC(kernel="linear", C=5)
clf7 = SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3, random_state=42, max_iter=10, tol=None)
```

4. Models

4.1 Logistic Regression

```
#LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression

# Create a CountVectorizer for text feature extraction
vectorizer = CountVectorizer()

# Fit the vectorizer on the training data and transform the data
X_train_vec = vectorizer.fit_transform(X_train)

# Transform the test data using the same vectorizer
X_test_vec = vectorizer.transform(X_test)

# Create a Logistic Regression classifier
logreg = LogisticRegression()

# Fit the classifier on the vectorized training data and corresponding labels
logreg.fit(X_train_vec, y_train)
```

```
# Make predictions on the test data
y_pred_logreg = logreg.predict(X_test_vec)

# Calculate the accuracy score
logreg_accuracy = accuracy_score(y_test, y_pred_logreg)
print("Log Reg: ", logreg_accuracy)
```

```
Log Reg:  0.8410925474000376
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
[253] from sklearn.metrics import roc_curve, auc
      import matplotlib.pyplot as plt

      # Obtain the predicted probabilities for the positive class
      y_score = logreg.predict_proba(X_test_vec)[:, 1]

      # Calculate the false positive rate, true positive rate, and thresholds
      fpr, tpr, thresholds = roc_curve(y_test, y_score)

      # Calculate the AUC score
      roc_auc = auc(fpr, tpr)
```

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

```
              precision    recall  f1-score   support

     class 1       0.72      0.58      0.64      7920
     class 2       0.87      0.93      0.90     24042

    accuracy                           0.84     31962
   macro avg       0.80      0.75      0.77     31962
weighted avg       0.83      0.84      0.83     31962
```

```
[255] #Confusion Matrix
      cm2 = confusion_matrix(y_pred=y_pred_logreg, y_true=y_test)
      cm2
```

```
array([[ 4580,  3340],
       [ 1739, 22303]])
```

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

```
AUC: 0.8816562177805003
```

The classification using logistic regression model yielded an accuracy of 84,1%.

4.2 SGD

```
#SGD

# Fit the vectorizer on the training data and transform the data
X_train_vec = vectorizer.fit_transform(X_train)

# Transform the test data using the same vectorizer
X_test_vec = vectorizer.transform(X_test)

# Create a SGD classifier
SGDC = SGDClassifier(loss='log')

# Fit the classifier on the vectorized training data and corresponding labels
SGDC.fit(X_train_vec, y_train)

# Make predictions on the test data
y_pred_SGD = SGDC.predict(X_test_vec)

# Calculate the accuracy score
SGDC_accuracy = accuracy_score(y_test, y_pred_SGD)
print("SGD Classification: ", SGDC_accuracy)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:163: FutureWarning: The loss 'log' was deprecated in v1.1 and wil
  warnings.warn(
SGD Classification:  0.8377761091295914
```

```
[258] # Obtain the predicted probabilities for the positive class
      y_score_SGD = SGDC.predict_proba(X_test_vec)[:, 1]

      # Calculate the false positive rate, true positive rate, and thresholds
      fpr, tpr, thresholds = roc_curve(y_test, y_score)

      # Calculate the AUC score
      roc_auc = auc(fpr, tpr)
```

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

```
                precision    recall  f1-score   support

       class 1       0.73      0.54      0.62      7920
       class 2       0.86      0.93      0.90     24042

      accuracy                           0.84     31962
     macro avg       0.80      0.74      0.76     31962
  weighted avg       0.83      0.84      0.83     31962
```

```
[260] #Confusion Matrix
      cm3 = confusion_matrix(y_pred=y_pred_SGD, y_true=y_test)
      cm3
```

```
array([[ 4303,  3617],
       [ 1568, 22474]])
```

The classification using the SGD classifier model yielded an accuracy of 83.7%.

4.3 Multinomial Naïve Bayes

```
# Fit the vectorizer on the training data and transform the data
X_train_vec = vectorizer.fit_transform(X_train)

# Transform the test data using the same vectorizer
X_test_vec = vectorizer.transform(X_test)

# Create a Multinomial Naive Bayes classifier
MultiNB = MultinomialNB()

# Fit the classifier on the vectorized training data and corresponding
MultiNB.fit(X_train_vec, y_train)

# Make predictions on the test data
y_pred_MultiNB = MultiNB.predict(X_test_vec)

# Calculate the accuracy score
MultiNB_accuracy = accuracy_score(y_test, y_pred_MultiNB)
print("MultinomialNB: ", MultiNB_accuracy)
```

```
MultinomialNB:  0.812370940491834
```

```
# Obtain the predicted probabilities for the positive class
y_score_MultiNB = MultiNB.predict_proba(X_test_vec)[:, 1]

# Calculate the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_score)

# Calculate the AUC score
roc_auc = auc(fpr, tpr)
```

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

```
              precision    recall  f1-score   support

     class 1       0.62      0.64      0.63      7920
     class 2       0.88      0.87      0.87     24042

    accuracy                           0.81     31962
   macro avg       0.75      0.75      0.75     31962
weighted avg       0.81      0.81      0.81     31962
```

```
[264] #Confusion Matrix
      cm4 = confusion_matrix(y_pred=y_pred_MultiNB, y_true=y_test)
      cm4

      array([[ 5078,  2842],
             [ 3155, 20887]])
```

The classification using the Multinomial Naive Bayes classifier model yielded an accuracy of 81.2%.

5 Conclusion

By comparing the three classification models used for drug recommendation, logistic regression, SGD, and multinomial naive Bayes, based on their accuracy values, logistic regression obtained the highest accuracy of 84.1%.