CIS 9665 Team 4 NLP Project

Chris Angelidis, Jose Cao-Alvira, Nicholas Exel Moreira de Andrade, Melis Ocal, Kristina Sarkissyan, Mohammad Shafique & Samantha Wang

Imports

In [1]:
```
import nltk
import re
import pandas as pd
import numpy as np
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, Voting
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer

import textstat

import seaborn as sns; sns.set()
import matplotlib.pyplot as plt

from dmba import plotDecisionTree, classificationSummary, regressionSummary
from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score
from nltk.tokenize import sent_tokenize, word_tokenize
```

Sources:

https://towardsdatascience.com/%EF%B8%8F-sentiment-analysis-aspect-based-opinion-mining-72a75e8c8a6d

https://yanlinc.medium.com/how-to-build-a-lda-topic-model-using-from-text-601cdcbfd3a6

https://machinelearninggeek.com/latent-dirichlet-allocation-using-scikit-learn/

Functions for Scoring and Metrics

In [2]:
```
def ca_score_model(train_act_y, train_pred_y, test_act_y, test_pred_y):
    print("Training Set Metrics:")
```

```python
    print("Accuracy on the train is:",accuracy_score(train_act_y,train_pred_y))
    classificationSummary(train_act_y,train_pred_y)
    print('The Precision on the train is:', precision_score(train_act_y,train_pred_y))
    print('The Recall on the train is:',recall_score(train_act_y,train_pred_y))
    print('The F-Measure on the train is:',f1_score(train_act_y,train_pred_y))

    print("\nTesting Set Metrics:")
    print("Accuracy on the test is:",accuracy_score(test_act_y, test_pred_y))
    classificationSummary(test_act_y, test_pred_y)
    print('The Precision on the test is:', precision_score(test_act_y, test_pred_y))
    print('The Recall on the test is:',recall_score(test_act_y, test_pred_y))
    print('The F-Measure on the test is:',f1_score(test_act_y, test_pred_y))

    test_acc = accuracy_score(test_act_y, test_pred_y)
    test_prec = precision_score(test_act_y, test_pred_y)
    test_recall = recall_score(test_act_y, test_pred_y)
    test_f = f1_score(test_act_y, test_pred_y)

    list_result = [test_acc, test_prec, test_recall, test_f]

    return(list_result)

def model_comp(base,optim,case1='Base Case:',case2='Optimized:'):
    print('\n{:<28}{:<14}{:<14}{:<14}{:<14}'.format('Model Comparison','Accuracy','Prec
    print('{:<28}{:<14.4f}{:<14.4f}{:<14.4f}{:<14.4f}'.format(case1,base[0],base[1],bas
    print('{:<28}{:<14.4f}{:<14.4f}{:<14.4f}{:<14.4f}'.format(case2,optim[0],optim[1],o
```

In [3]:

```python
def get_metrics(y_test, y_predicted):
    # Precision = TP / (TP + FP)
    precision = precision_score(y_test, y_predicted, pos_label=None,
                                average='weighted')
    # Recall = TP / (TP + FN)
    recall = recall_score(y_test, y_predicted, pos_label=None,
                          average='weighted')

    # Harmonic mean of precision and recall
    f1 = f1_score(y_test, y_predicted, pos_label=None, average='weighted')

    # Accuracy = TP + TN / Total
    accuracy = accuracy_score(y_test, y_predicted)
    return accuracy, precision, recall, f1
```

In [4]:

```python
def get_NBmetrics(test_y_est,test_y):

    # Scores for nltk Naive Bayes

    # Identify the predicted value of the classifier

    # test_y_est = [classifier.classify(xxx) for xxx in test_set_X]

    # Identify the true value of the variable

    # test_y = list(test_set_X)

    # Calculate the components of the Confusion Matrix

    TP = 0.00000000001
```

```
    TN = 0.00000000001
    FP = 0.00000000001
    FN = 0.00000000001

    for i in range(len(test_y)):
        if str(test_y_est[i])=="True" and test_y_est[i]==test_y[i]:
            TP = TP+1
        elif str(test_y_est[i])=="False" and test_y_est[i]==test_y[i]:
            TN = TN+1
        elif str(test_y_est[i])=="True" and test_y_est[i]!=test_y[i]:
            FP = FP+1
        else:
            FN = FN+1


    # Calculate the Metrics

    # a. Accuracy Rate
    ACC = (TP+TN)/(TP+TN+FP+FN)
    #print("The accuracy rate is", round(ACC,4))

    # b. Precision
    PRE = TP/(TP+FP)
    #print("The precision is", round(PRE,4))

    # c. Recall
    REC = TP/(TP+FN)
    #print("The recall is", round(REC,4))

    # d. F-measure
    Fscore = 2*PRE*REC/(PRE+REC)
    #print("The F-measure is", round(Fscore,4))

    return ACC, PRE, REC, Fscore
```

Dataset Preparation

In [5]:

```
# Read in training and test data
# Then recombine so we can later split according to a seed and create 3 sets: train, de
df1=pd.read_csv('drugsComTest_raw.csv')
df2=pd.read_csv('drugsComTrain_raw.csv')
source_df = pd.concat([df1,df2])
```

In [6]:

```
# Overall statistics about the dataset
source_df.describe(include='all')
```

Out[6]:

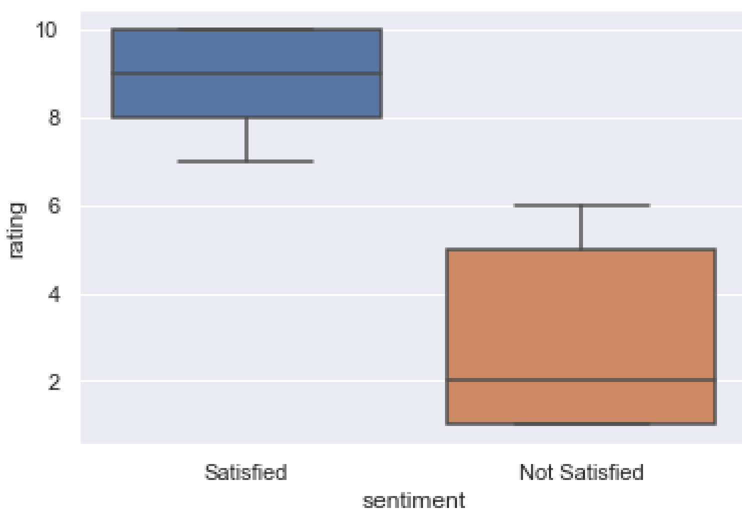
	uniqueID	drugName	condition	review	rating	date	usefulCount
count	215063.000000	215063	213869	215063	215063.000000	215063	215063.000000
unique	NaN	3671	916	128478	NaN	3579	NaN
top	NaN	Levonorgestrel	Birth Control	"Good"	NaN	1-Mar-16	NaN
freq	NaN	4930	38436	39	NaN	185	NaN

	uniqueID	drugName	condition	review	rating	date	usefulCount
mean	116039.364814	NaN	NaN	NaN	6.990008	NaN	28.001004
std	67007.913366	NaN	NaN	NaN	3.275554	NaN	36.346069
min	0.000000	NaN	NaN	NaN	1.000000	NaN	0.000000
25%	58115.500000	NaN	NaN	NaN	5.000000	NaN	6.000000
50%	115867.000000	NaN	NaN	NaN	8.000000	NaN	16.000000
75%	173963.500000	NaN	NaN	NaN	10.000000	NaN	36.000000
max	232291.000000	NaN	NaN	NaN	10.000000	NaN	1291.000000

In [7]:

```
# Create sentiment column
source_df['sentiment']= np.where(source_df['rating']>=7,'Satisfied','Not Satisfied')

# Show the coverage for each sentiment by its rating
sns.boxplot(x='sentiment', y="rating", data=source_df)
plt.show()
```



In [8]:

```
# Code that temporarily limits df to first 10k to improve performance while designing t
#source_df=source_df.head(10000)
```

Add External Data on Drug Pricing

In [9]:

```
price=pd.read_csv('prices.csv')
price=price.rename(columns = {'Spending ':'Spending'})
price['Brand Name'].replace(to_replace=r'\*$',value="", inplace=True, regex=True)
price['Brand Name'].replace(to_replace=r'-',value=" / ", inplace=True, regex=True)
price = price.groupby(['Brand Name'])['Spending'].mean().reset_index()

price.columns = [s.strip().replace(' ', '') for s in price.columns]

# Perform cleaning, normalization, and tokenization of data

# Normalize the drug names
```

```
price['drugName_list2']=price['BrandName'].str.lower()

# Tokenize the drug names
price['drugName_list2']=price['drugName_list2'].apply(lambda BrandName: nltk.word_token

# Create list of drug names and remove 20,000 most common words as per Google
drugs = price['drugName_list2'].tolist()
words_common = [line.strip() for line in open('20k.txt', 'r')]
drugs = [token for token in drugs if token not in words_common]
drugs_names2=[name for sublist in drugs for name in sublist]

# Perform further cleaning on the drug names
price['drugName_list2'].replace(to_replace=r"&#039;",value="'", inplace=True, regex=Tru
price['drugName_list2'].replace(to_replace=r'^\"',value="", inplace=True, regex=True)
price['drugName_list2'].replace(to_replace=r'\"$',value="", inplace=True, regex=True)
price['drugName_list2'].replace(to_replace=r'http\S+',value="", inplace=True, regex=Tru
price['drugName_list2'].replace(to_replace=r'http',value="", inplace=True, regex=True)
price['drugName_list2'].replace(to_replace=r'(\d)',value="", inplace=True, regex=True)
price['drugName_list2'].replace(to_replace=r'@\S+',value="", inplace=True, regex=True)
price['drugName_list2'].replace(to_replace=r'[^A-Za-z0-9(),!?@\'\`\"\_\n]',value="", in
price['drugName_list2'].replace(to_replace=r'@',value="at", inplace=True, regex=True)

price["drugName_list3"] = [" ".join(w) for w in price["drugName_list2"]]
price
```

Out[9]:

	BrandName	Spending	drugName_list2	drugName_list3
0	1st Tier Unifine Pentips	0.200	[1st, tier, unifine, pentips]	1st tier unifine pentips
1	1st Tier Unifine Pentips Plus	0.210	[1st, tier, unifine, pentips, plus]	1st tier unifine pentips plus
2	Abacavir	4.800	[abacavir]	abacavir
3	Abacavir / Lamivudine	26.980	[abacavir, /, lamivudine]	abacavir / lamivudine
4	Abacavir / Lamivudine / Zidovudine	21.530	[abacavir, /, lamivudine, /, zidovudine]	abacavir / lamivudine / zidovudine
...	...	...	...	...
2984	Zyprexa	22.670	[zyprexa]	zyprexa
2985	Zyprexa Relprevv	1002.290	[zyprexa, relprevv]	zyprexa relprevv
2986	Zyprexa Zydis	26.580	[zyprexa, zydis]	zyprexa zydis
2987	Zytiga	73.690	[zytiga]	zytiga
2988	Zyvox	76.765	[zyvox]	zyvox

2989 rows × 4 columns

In [10]:

```
print(source_df.shape)
```

```
(215063, 8)
```

In [11]:

```
source_df.head()
```

Out[11]:

uniqueID	drugName	condition	review	rating	date	usefulCount	sentiment

	uniqueID	drugName	condition	review	rating	date	usefulCount	sentiment
0	163740	Mirtazapine	Depression	"I've tried a few antidepressants over th...	10	28-Feb-12	22	Satisfied
1	206473	Mesalamine	Crohn's Disease, Maintenance	"My son has Crohn's disease and has done ...	8	17-May-09	17	Satisfied
2	159672	Bactrim	Urinary Tract Infection	"Quick reduction of symptoms"	9	29-Sep-17	3	Satisfied
3	39293	Contrave	Weight Loss	"Contrave combines drugs that were used for al...	9	5-Mar-17	35	Satisfied
4	97768	Cyclafem 1 / 35	Birth Control	"I have been on this birth control for one cyc...	9	22-Oct-15	4	Satisfied

In [12]:

```
# Create a new column called "cleaned_review"
# Normalize the reviews
source_df['cleaned_review']=source_df['review'].str.lower()
```

In [13]:

```
# Normalize the conditions
source_df['condition']=source_df['condition'].str.lower()
```

In [14]:

```
# Remove the quotes surrounding reviews and fix apostrophes
source_df['cleaned_review'].replace(to_replace=r"&#039;",value="'", inplace=True, regex
source_df['cleaned_review'].replace(to_replace=r'^\"',value="", inplace=True, regex=Tru

source_df['cleaned_review'].replace(to_replace=r'\"$',value="", inplace=True, regex=Tru
```

In addition to removing the English stopwords we once again remove the 20,000 most common words as per Google.

In [15]:

```
# Normalize the drug names
source_df['drugName_list']=source_df['drugName'].str.lower()

# Tokenize drug names
source_df['drugName_list']=source_df['drugName_list'].apply(lambda drugName: nltk.word_

# Create list of drug names and remove 20,000 most common words as per Google
drugs = source_df["drugName_list"].tolist()
words_common = [line.strip() for line in open('20k.txt', 'r')]
drugs = [token for token in drugs if token not in words_common]
drugs_names=[name for sublist in drugs for name in sublist]
```

In [16]:

```
# Perform further cleaning on the drug names
source_df['drugName_list'].replace(to_replace=r"&#039;",value="'", inplace=True, regex=
```

```
source_df['drugName_list'].replace(to_replace=r'^\"',value="", inplace=True, regex=True
source_df['drugName_list'].replace(to_replace=r'\"$',value="", inplace=True, regex=True
source_df['drugName_list'].replace(to_replace=r'http\S+',value="", inplace=True, regex=
source_df['drugName_list'].replace(to_replace=r'http',value="", inplace=True, regex=Tru
source_df['drugName_list'].replace(to_replace=r'(\d)',value="", inplace=True, regex=Tru
source_df['drugName_list'].replace(to_replace=r'@\S+',value="", inplace=True, regex=Tru
source_df['drugName_list'].replace(to_replace=r'[^A-Za-z0-9(),!?@\'\`\"\_\n]',value="",
source_df['drugName_list'].replace(to_replace=r'@',value="at", inplace=True, regex=True
```

In [17]:

```
source_df["drugName_list3"] = [" ".join(w) for w in source_df["drugName_list"]]
source_df.head()
```

Out[17]:

	uniqueID	drugName	condition	review	rating	date	usefulCount	sentiment	cleaned_re
0	163740	Mirtazapine	depression	"I've tried a few antidepressants over th...	10	28-Feb-12	22	Satisfied	i've tried antidepres over the
1	206473	Mesalamine	crohn's disease, maintenance	"My son has Crohn's disease and has done ...	8	17-May-09	17	Satisfied	my so crohn's di and has ve
2	159672	Bactrim	urinary tract infection	"Quick reduction of symptoms"	9	29-Sep-17	3	Satisfied	quick redu of symp
3	39293	Contrave	weight loss	"Contrave combines drugs that were used for al...	9	5-Mar-17	35	Satisfied	cor combines that were fo
4	97768	Cyclafem 1 / 35	birth control	"I have been on this birth control for one cyc...	9	22-Oct-15	4	Satisfied	i have be this control fo

In [18]:

```
# Change drugNames to title case so that it betters matches with names in price
source_df = source_df.join(price.set_index('drugName_list3'), on='drugName_list3')
```

In [19]:

```
# Determine the number of null values
for col in source_df.columns:
        na_df = source_df[col].isna()
        print('Number of Nulls in',col,':',na_df[na_df==True].count())

#source_df.dropna(inplace=True)
print(source_df.shape)

# The original data set contains 215,063 observations
# Dropping the null values from price leaves us with 215063-120557 = 94256 observations
```

```
Number of Nulls in uniqueID : 0
Number of Nulls in drugName : 0
Number of Nulls in condition : 1194
```

```
Number of Nulls in review : 0
Number of Nulls in rating : 0
Number of Nulls in date : 0
Number of Nulls in usefulCount : 0
Number of Nulls in sentiment : 0
Number of Nulls in cleaned_review : 0
Number of Nulls in drugName_list : 0
Number of Nulls in drugName_list3 : 0
Number of Nulls in BrandName : 120557
Number of Nulls in Spending : 120557
Number of Nulls in drugName_list2 : 120557
(215063, 14)
```

In [20]:

```
# Save a raw version of the cleaned reviews to be used in the VADER classifier
source_df['cleaned_review_raw_vader'] = source_df['cleaned_review']
```

In [21]:

```
# Tokenize each the reviews
source_df['cleaned_review'] = source_df['cleaned_review'].apply(lambda review: nltk.wor
```

In addition to removing English stopwords and the 20,000 as per Google we also eliminate some corpus specific stopwords which we inferred from the 500 most common words in the reviews.

In [22]:

```
# Remove nltk stopwords as well as inferred corpus specific stopwords
stop_words = nltk.corpus.stopwords.words('english')

words_commonK = pd.read_csv('Identifying Corpus Specific Stop Words.csv', index_col=Fal

stop_words = stop_words + list(words_commonK["Stop_Words"])

source_df['cleaned_review'] = source_df['cleaned_review'].apply(lambda review: [word fo
```

In [23]:

```
# Remove numbers and punctions from each review
source_df['cleaned_review'] = source_df['cleaned_review'].apply(lambda review: [word fo
```

In [24]:

```
# Lemmatize each review
wnl = nltk.WordNetLemmatizer()
source_df['cleaned_review'] = source_df['cleaned_review'].apply(lambda review: [wnl.lem
```

In [25]:

```
# Save a raw version of cleaned reviews for use in the count vectorizer
source_df['cleaned_review_raw_cv'] = source_df['cleaned_review'].apply(lambda review: '
```

In [26]:

```
# Inspect a subset of the cleaned reviews to confirm that data pre-processing performed
source_df.head(5)
```

Out[26]:

uniqueID	drugName	condition	review	rating	date	usefulCount	sentiment	cleaned_re

	uniqueID	drugName	condition	review	rating	date	usefulCount	sentiment	cleaned_re
0	163740	Mirtazapine	depression	"I've tried a few antidepressants over th...	10	28-Feb-12	22	Satisfied	[antidepre: citalo
1	206473	Mesalamine	crohn's disease, maintenance	"My son has Crohn's disease and has done ...	8	17-May-09	17	Satisfied	[son, c disease, well, a cc
2	159672	Bactrim	urinary tract infection	"Quick reduction of symptoms"	9	29-Sep-17	3	Satisfied	[c redu symp
3	39293	Contrave	weight loss	"Contrave combines drugs that were used for al...	9	5-Mar-17	35	Satisfied	[con combine, used, alc sn
4	97768	Cyclafem 1 / 35	birth control	"I have been on this birth control for one cyc...	9	22-Oct-15	4	Satisfied	[birth, co cycle, rea review, t

Flesch-Kincaid Grade

In [27]:

```
review_col = source_df["review"]

Grade = [textstat.flesch_kincaid_grade(i) for i in review_col]
```

In [28]:

```
# Inspect a subset of the cleaned reviews to confirm that data pre-processing performed

source_df["Grade"] = Grade

source_df.head(5)
```

Out[28]:

	uniqueID	drugName	condition	review	rating	date	usefulCount	sentiment	cleaned_re
0	163740	Mirtazapine	depression	"I've tried a few antidepressants over th...	10	28-Feb-12	22	Satisfied	[antidepre: citalo
1	206473	Mesalamine	crohn's disease, maintenance	"My son has Crohn's disease and has done ...	8	17-May-09	17	Satisfied	[son, c disease, well, a cc
2	159672	Bactrim	urinary tract infection	"Quick reduction of symptoms"	9	29-Sep-17	3	Satisfied	[c redu symp

	uniqueID	drugName	condition	review	rating	date	usefulCount	sentiment	cleaned_re
3	39293	Contrave	weight loss	"Contrave combines drugs that were used for al...	9	5-Mar-17	35	Satisfied	[con combine, used, alc sn
4	97768	Cyclafem 1 / 35	birth control	"I have been on this birth control for one cyc...	9	22-Oct-15	4	Satisfied	[birth, co cycle, rea review, t

Data Exploration

In [29]:

```
review_words = source_df['cleaned_review'].values.tolist()
review_words = [word for review in review_words for word in review]
review_words_fdist = nltk.FreqDist(review_words)
```

In [30]:

```
# Here we see that there are many corpus specific stopwords
# We will try to exclude some from the count vectorizer to improve performance
review_words_fdist.most_common(100)
```

Out[30]:

```
[('effect', 73524),
 ('side', 70548),
 ('taking', 68268),
 ('pain', 63622),
 ('take', 62525),
 ('year', 62110),
 ('get', 58207),
 ('month', 56803),
 ('started', 56480),
 ('like', 55776),
 ('pill', 55356),
 ('day', 53653),
 ('period', 53546),
 ('work', 51380),
 ('feel', 50560),
 ('would', 48434),
 ('medication', 42475),
 ('doctor', 42119),
 ('week', 39554),
 ('took', 37435),
 ('weight', 36723),
 ('got', 36341),
 ('medicine', 34036),
 ('since', 33776),
 ('life', 33738),
 ('bad', 32602),
 ('still', 32534),
 ('really', 31721),
 ('much', 31178),
 ('could', 30843),
 ('anxiety', 30767),
 ('never', 29951),
 ('better', 29224),
 ('went', 29148),
```

```
('control', 28198),
('go', 28181),
('help', 27715),
('felt', 27593),
('every', 27136),
('well', 27028),
('good', 26441),
('great', 25675),
('drug', 24852),
('sleep', 24498),
('tried', 23096),
('acne', 22342),
('birth', 22279),
('made', 21790),
('little', 21679),
('hour', 21425),
('going', 21319),
('depression', 20941),
('make', 20825),
('dose', 20804),
('prescribed', 20586),
('worked', 20532),
('used', 20366),
('headache', 19976),
('problem', 19942),
('feeling', 19915),
('put', 19462),
('getting', 19442),
('severe', 19362),
('lot', 19172),
('mood', 18581),
('use', 17407),
('experience', 17190),
('far', 17167),
('cramp', 17116),
('nothing', 17112),
('ever', 17043),
('skin', 16955),
('symptom', 16749),
('away', 16330),
('know', 16221),
('nausea', 16209),
('try', 16161),
('say', 16154),
('thought', 16152),
('sex', 16099),
('think', 16083),
('using', 16073),
('time', 15995),
('bleeding', 15696),
('lost', 15423),
('helped', 15416),
('normal', 15168),
('gain', 15095),
('recommend', 14804),
('stop', 14748),
('without', 14696),
('morning', 14569),
('long', 14415),
('body', 14332),
('worse', 14168),
('however', 14138),
('old', 14126),
('within', 13558),
```

```
 ('taken', 13522),
 ('see', 13512)]
```

In [31]:

```
# Here we see that there are many nuanced words that only appear a few times
# We will try to exclude some from the count vectorizer to improve performance
review_words_fdist.most_common()[-20000:]
```

Out[31]:

```
[('azothromycin', 2),
 ('tricyclone', 2),
 ('prevfiem', 2),
 ('grinder', 2),
 ('wavelength', 2),
 ('robaxasil', 2),
 ('useles', 2),
 ('stck', 2),
 ('skinlite', 2),
 ('insta', 2),
 ('laminectemy', 2),
 ('pallative', 2),
 ('veteneriàn', 2),
 ('bydurean', 2),
 ('mobth', 2),
 ('boggling', 2),
 ('perimenapause', 2),
 ('decayed', 2),
 ('uterin', 2),
 ('overestimate', 2),
 ('visanne', 2),
 ('nuycnta', 2),
 ('crazyyyy', 2),
 ('vineager', 2),
 ('quesadilla', 2),
 ('suplement', 2),
 ('quater', 2),
 ('ropinerol', 2),
 ('cyclean', 2),
 ('bobby', 2),
 ('unclogging', 2),
 ('granulation', 2),
 ('precedure', 2),
 ('quicklg', 2),
 ('relistore', 2),
 ('servu', 2),
 ('zolfren', 2),
 ('antiepileptic', 2),
 ('acholholic', 2),
 ('expedience', 2),
 ('kfc', 2),
 ('pioglitzones', 2),
 ('gliclizide', 2),
 ('methalyn', 2),
 ('expensively', 2),
 ('osmose', 2),
 ('cresent', 2),
 ('anc', 2),
 ('eduring', 2),
 ('santura', 2),
 ('sunked', 2),
 ('refuted', 2),
 ('mindboggling', 2),
 ('forumla', 2),
 ('simphone', 2),
 ('carving', 2),
```

```
('finessing', 2),
('scc', 2),
('intruding', 2),
('azasan', 2),
('lent', 2),
('downrated', 2),
('lookup', 2),
('promo', 2),
('meshed', 2),
('anuryzm', 2),
('pseudotumour', 2),
('expunged', 2),
('pathogen', 2),
('citaloplam', 2),
('hypochondriacal', 2),
('confidentiality', 2),
('medicatios', 2),
('balanitis', 2),
('ppne', 2),
('testoterone', 2),
('fanatically', 2),
('violin', 2),
('emotionalness', 2),
('drowienes', 2),
('triptin', 2),
('holisticly', 2),
('koliopin', 2),
('risperadal', 2),
('fumiderm', 2),
('hpts', 2),
('psd', 2),
('folllow', 2),
('xiitra', 2),
('quota', 2),
('profasi', 2),
('diebeties', 2),
('theray', 2),
('tropicamide', 2),
('hysteoscopy', 2),
('tranaxamic', 2),
('dewey', 2),
('freguent', 2),
('releaving', 2),
('lunge', 2),
('neuorontin', 2),
('symmetrically', 2),
('unexpectedness', 2),
('protested', 2),
('fha', 2),
('deepshit', 2),
('retaliation', 2),
('noones', 2),
('subtex', 2),
('absoletley', 2),
('weepies', 2),
('mitazipine', 2),
('ekkkk', 2),
('rocefin', 2),
('corlanor', 2),
('eben', 2),
('darvacets', 2),
('gnatting', 2),
('shiort', 2),
('vaiety', 2),
('protofoam', 2),
```

```
('cirrosis', 2),
('harv', 2),
('learne', 2),
('atty', 2),
('cardiocthoracic', 2),
('espesically', 2),
('rathered', 2),
('beclomethasone', 2),
('salmeterol', 2),
('sporonox', 2),
('gelatinous', 2),
('appeased', 2),
('jason', 2),
('voorhees', 2),
('complection', 2),
('unattractiveness', 2),
('cheeked', 2),
('firbromyalgia', 2),
('lipton', 2),
('ellipitical', 2),
('posso', 2),
('pegged', 2),
('affix', 2),
('unspeakable', 2),
('menopas', 2),
('neverland', 2),
('impct', 2),
('ppg', 2),
('fentanil', 2),
('convertible', 2),
('physique', 2),
('scalopine', 2),
('vancadymicin', 2),
('penguin', 2),
('nightmarishly', 2),
('doppiness', 2),
('paxtine', 2),
('tamixifen', 2),
('demadex', 2),
('urogenic', 2),
('demertorolgist', 2),
('doiing', 2),
('unabating', 2),
('alergies', 2),
('juniper', 2),
('metropolitan', 2),
('maryland', 2),
('zomby', 2),
('everlasting', 2),
('subarachnoid', 2),
('hurtles', 2),
('sloughes', 2),
('cycts', 2),
('overies', 2),
('pobagai', 2),
('beslomra', 2),
('antipyschotics', 2),
('restated', 2),
('colititis', 2),
('caecum', 2),
('antecedent', 2),
('hemis', 2),
('cirvix', 2),
('ablot', 2),
('goining', 2),
```

```
('zanatane', 2),
('lourdes', 2),
('symvastin', 2),
('supratz', 2),
('viibryrd', 2),
('copazone', 2),
('shoelace', 2),
('vancenase', 2),
('unendurable', 2),
('reportedsuch', 2),
('erges', 2),
('evenually', 2),
('nonesense', 2),
('representation', 2),
('dehumidifier', 2),
('luproid', 2),
('enzy', 2),
('downy', 2),
('vegetate', 2),
('methocarb', 2),
('inablity', 2),
('ripen', 2),
('concossion', 2),
('certaindri', 2),
('duraflu', 2),
('limish', 2),
('nagged', 2),
('agarol', 2),
('nulax', 2),
('myinsurance', 2),
('switchng', 2),
('manufactor', 2),
('nightshift', 2),
('salami', 2),
('epid', 2),
('yook', 2),
('spotter', 2),
('anyting', 2),
('amiodipine', 2),
('seisure', 2),
('aregano', 2),
('derralin', 2),
('inderral', 2),
('pimozide', 2),
('yey', 2),
('zealous', 2),
('butnever', 2),
('overconfidence', 2),
('gallemore', 2),
('preemptive', 2),
('entitlement', 2),
('perviously', 2),
('atheist', 2),
('atn', 2),
('cardioligst', 2),
('numbingness', 2),
('vijay', 2),
('thermoplasty', 2),
('motorcycling', 2),
('costo', 2),
('condritis', 2),
('leopard', 2),
('happeir', 2),
('gadolinium', 2),
('brazillian', 2),
```

```
('jiu', 2),
('jitsu', 2),
('adenomyosus', 2),
('zuboxone', 2),
('abornmal', 2),
('fistful', 2),
('assistiance', 2),
('duragesics', 2),
('forsure', 2),
('conning', 2),
('venogram', 2),
('stenting', 2),
('tinged', 2),
('mantain', 2),
('neorupathic', 2),
('nugrape', 2),
('feei', 2),
('slidder', 2),
('lethesis', 2),
('flipfloping', 2),
('nux', 2),
('vomica', 2),
('normall', 2),
('litletta', 2),
('stabalizes', 2),
('wook', 2),
('bennifit', 2),
('saratonin', 2),
('keyworker', 2),
('hypocalcaemic', 2),
('hypoparathyroid', 2),
('symp', 2),
('tepid', 2),
('trilifon', 2),
('hydrocone', 2),
('raynaund', 2),
('imozop', 2),
('gastroplasty', 2),
('nomenclature', 2),
('synthesized', 2),
('cocci', 2),
('dxa', 2),
('evenwork', 2),
('lighty', 2),
('priceline', 2),
('culf', 2),
('anzemet', 2),
('anesthesized', 2),
('masseuse', 2),
('han', 2),
('mexiletine', 2),
('toggled', 2),
('crimson', 2),
('ipledge', 2),
('fushion', 2),
('excersion', 2),
('skined', 2),
('ambiens', 2),
('withoug', 2),
('reopen', 2),
('loracets', 2),
('unearthly', 2),
('temperpedic', 2),
('trendelenburg', 2),
('diaphoretic', 2),
```

```
('torturoed', 2),
('statterra', 2),
('condescending', 2),
('stella', 2),
('chios', 2),
('koz', 2),
('lesbos', 2),
('plied', 2),
('confiscated', 2),
('vedolizumab', 2),
('ustekinumab', 2),
('laborious', 2),
('benedrly', 2),
('routing', 2),
('obnoxiously', 2),
('certaintly', 2),
('risnia', 2),
('hesitatant', 2),
('sulfazalazine', 2),
('riddle', 2),
('contently', 2),
('boop', 2),
('blooding', 2),
('breat', 2),
('coy', 2),
('achene', 2),
('brint', 2),
('avonez', 2),
('methyphenidate', 2),
('postoperatively', 2),
('postprandial', 2),
('woud', 2),
('eme', 2),
('progessing', 2),
('crampage', 2),
('tenacity', 2),
('bulking', 2),
('vegging', 2),
('enegry', 2),
('chainsmoking', 2),
('mathematician', 2),
('altboutel', 2),
('gildes', 2),
('familt', 2),
('allese', 2),
('petition', 2),
('intuitiv', 2),
('internationally', 2),
('scronyx', 2),
('advocating', 2),
('frequecy', 2),
('uncontronablle', 2),
('jim', 2),
('konopin', 2),
('recomemnd', 2),
('ialso', 2),
('underachiever', 2),
('symbolizes', 2),
('brightest', 2),
('eniemic', 2),
('turtleneck', 2),
('metonia', 2),
('fidelity', 2),
('overprotective', 2),
('effectives', 2),
```

```
('considera', 2),
('checkered', 2),
('stfu', 2),
('fostex', 2),
('againz', 2),
('ziprazidone', 2),
('neorephrine', 2),
('depository', 2),
('radiace', 2),
('medicaiton', 2),
('sdri', 2),
('floxied', 2),
('insurace', 2),
('peridontal', 2),
('simulating', 2),
('ccf', 2),
('muscleaches', 2),
('enamored', 2),
('moodiest', 2),
('blockbuster', 2),
('scizophrenia', 2),
('aimlessness', 2),
('creaky', 2),
('hhe', 2),
('sweeney', 2),
('vaporrub', 2),
('hadno', 2),
('simulated', 2),
('studyingl', 2),
('benedeyl', 2),
('inand', 2),
('cleverly', 2),
('reinforced', 2),
('horney', 2),
('buperenorphine', 2),
('nutting', 2),
('comittment', 2),
('melanosis', 2),
('thaythat', 2),
('osteoarthrititis', 2),
('celebrix', 2),
('platlet', 2),
('vesicular', 2),
('monanessa', 2),
('irratble', 2),
('monogomous', 2),
('spectracef', 2),
('kypoplasty', 2),
('malaysia', 2),
('topi', 2),
('aerogel', 2),
('pseudotumo', 2),
('subsy', 2),
('academy', 2),
('dle', 2),
('psin', 2),
('sideline', 2),
('jaiil', 2),
('detriot', 2),
('comparably', 2),
('hypventilating', 2),
('infecwtion', 2),
('hows', 2),
('accessory', 2),
('merging', 2),
```

```
('lamogotrine', 2),
('recommit', 2),
('catalepxy', 2),
('tallness', 2),
('anakinra', 2),
('kilometre', 2),
('subdided', 2),
('swisper', 2),
('toiled', 2),
('baddddd', 2),
('microgrestrin', 2),
('pleasureable', 2),
('naturethyroid', 2),
('moleculer', 2),
('supercedes', 2),
('magnifies', 2),
('rxing', 2),
('craptacular', 2),
('urianary', 2),
('anyyy', 2),
('humilog', 2),
('diarhoe', 2),
('wieghed', 2),
('differwnt', 2),
('chaning', 2),
('cambell', 2),
('bolted', 2),
('predensone', 2),
('shrek', 2),
('effaced', 2),
('homocysteine', 2),
('subtleness', 2),
('neighbour', 2),
('mitraxepam', 2),
('generes', 2),
('coexist', 2),
('entertain', 2),
('speedo', 2),
('populate', 2),
('clawed', 2),
('nore', 2),
('braken', 2),
('probelm', 2),
('nexplanons', 2),
('beeotch', 2),
('loxitane', 2),
('losings', 2),
('plummetted', 2),
('smearing', 2),
('mayonaise', 2),
('kn', 2),
('alwaysbhas', 2),
('gabapentim', 2),
('orgamism', 2),
('beck', 2),
('setpoint', 2),
('cyctitis', 2),
('easays', 2),
('sacrio', 2),
('ashlyan', 2),
('statred', 2),
('diarrhe', 2),
('harmones', 2),
('intuned', 2),
('emema', 2),
```

```
('belsomr', 2),
('kepprage', 2),
('excperiencing', 2),
('subdermal', 2),
('whaawhoo', 2),
('uppermost', 2),
('chantel', 2),
('paragon', 2),
('kim', 2),
('seperates', 2),
('bigpharma', 2),
('humulog', 2),
('overstress', 2),
('dampening', 2),
('happing', 2),
('bladed', 2),
('sterol', 2),
('vardenifil', 2),
('paperback', 2),
('kindle', 2),
('cholestral', 2),
('accumalated', 2),
('gasey', 2),
('ange', 2),
('tranced', 2),
('scribe', 2),
('alteting', 2),
('qbtest', 2),
('crampers', 2),
('tumbler', 2),
('discrepancy', 2),
('lu', 2),
('pron', 2),
('uuuuuuuuggggg', 2),
('naseauss', 2),
('neverending', 2),
('ablilify', 2),
('defficult', 2),
('bulgaria', 2),
('promacta', 2),
('nexlpanon', 2),
('colonectomy', 2),
('hallucenenic', 2),
('clinc', 2),
('inbrel', 2),
('cutler', 2),
('proximal', 2),
('asthama', 2),
('awasting', 2),
('farce', 2),
('unobtainable', 2),
('nrti', 2),
('extinguished', 2),
('aggrevation', 2),
('blod', 2),
('sahm', 2),
('unsteadyness', 2),
('nasacourt', 2),
('hysterotomy', 2),
('ucerin', 2),
('lemoncrystal', 2),
('jawlines', 2),
('rutgers', 2),
('cadc', 2),
('seroonin', 2),
```

```
('delsum', 2),
('prozacfor', 2),
('cesearean', 2),
('stern', 2),
('corkscrew', 2),
('andrgel', 2),
('morphone', 2),
('tuft', 2),
('macrobin', 2),
('deplayed', 2),
('parted', 2),
('phernagan', 2),
('electrocardioversions', 2),
('boehringer', 2),
('ingelheim', 2),
('dailiness', 2),
('minair', 2),
('psoas', 2),
('polyneuralgia', 2),
('glimeride', 2),
('nopal', 2),
('stieva', 2),
('endometrieosis', 2),
('gloominess', 2),
('oxyface', 2),
('rchop', 2),
('bcnu', 2),
('lengthening', 2),
('polysystic', 2),
('coild', 2),
('ning', 2),
('truvuda', 2),
('propery', 2),
('hokey', 2),
('dorey', 2),
('opp', 2),
('backyard', 2),
('countined', 2),
('crampong', 2),
('bybthe', 2),
('prednosolone', 2),
('allertec', 2),
('flack', 2),
('urinination', 2),
('recommendedation', 2),
('neurolgist', 2),
('cheecks', 2),
('howling', 2),
('ingrediants', 2),
('rotisserie', 2),
('stainless', 2),
('gawds', 2),
('perfumy', 2),
('pleomorfic', 2),
('xantro', 2),
('astro', 2),
('sytoma', 2),
('obseen', 2),
('racingthoughts', 2),
('mif', 2),
('advocacy', 2),
('astonishes', 2),
('reearch', 2),
('prodded', 2),
('znd', 2),
```

```
('receeding', 2),
('latina', 2),
('stunting', 2),
('litacane', 2),
('cordarone', 2),
('mininal', 2),
('propafanone', 2),
('lisipril', 2),
('noghts', 2),
('depravation', 2),
('cholera', 2),
('cimpay', 2),
('arthitis', 2),
('norcs', 2),
('radiological', 2),
('uams', 2),
('fybogel', 2),
('albsolutely', 2),
('reyes', 2),
('hullucinate', 2),
('onthis', 2),
('stayin', 2),
('narvasc', 2),
('underemployed', 2),
('pcf', 2),
('constanly', 2),
('loesrin', 2),
('cyclaflem', 2),
('hounding', 2),
('scrunch', 2),
('linty', 2),
('tango', 2),
('dysrhythmias', 2),
('isvery', 2),
('planta', 2),
('ryme', 2),
('stye', 2),
('triaminic', 2),
('rogue', 2),
('eyestrain', 2),
('armitiza', 2),
('aner', 2),
('vibro', 2),
('anxioty', 2),
('cph', 2),
('ultraviolet', 2),
('advill', 2),
('bupropian', 2),
('aderol', 2),
('awed', 2),
('nicoretter', 2),
('kodak', 2),
('traodone', 2),
('tamilfu', 2),
('yeastaway', 2),
('scrubber', 2),
('dispersing', 2),
('tallied', 2),
('sycle', 2),
('prayerful', 2),
('naprogesic', 2),
('tunica', 2),
('orthovics', 2),
('escrutiating', 2),
('outragous', 2),
```

```
('zyprexia', 2),
('glta', 2),
('palperidone', 2),
('mottled', 2),
('unsureness', 2),
('stamping', 2),
('gemfizibrol', 2),
('clobestasol', 2),
('musenex', 2),
('augmentum', 2),
('ocycontin', 2),
('pita', 2),
('sweatier', 2),
('wouldent', 2),
('excise', 2),
('nephrotic', 2),
('gteat', 2),
('chirpping', 2),
('transtec', 2),
('methyln', 2),
('fluoroqinolone', 2),
('methlylin', 2),
('estee', 2),
('lauder', 2),
('waitlist', 2),
('celtic', 2),
('bloodline', 2),
('suitably', 2),
('superlong', 2),
('contreceptive', 2),
('mirtrazipine', 2),
('molested', 2),
('neorolgist', 2),
('atavin', 2),
('elaquis', 2),
('mam', 2),
('disography', 2),
('medicati', 2),
('clonazapem', 2),
('beatle', 2),
('asthmanex', 2),
('lorecet', 2),
('covereing', 2),
('dissolveable', 2),
('arythmias', 2),
('ibprophin', 2),
('brampton', 2),
('southbound', 2),
('amyril', 2),
('apparenty', 2),
('bronchopneumonia', 2),
('basted', 2),
('michocondria', 2),
('marciano', 2),
('gravadarum', 2),
('cerve', 2),
('lunestas', 2),
('nauseau', 2),
('unisome', 2),
('benzalin', 2),
('bezaclin', 2),
('esphogus', 2),
('dococyline', 2),
('dextrastat', 2),
('pessimist', 2),
```

```
('unimpacted', 2),
('clobetesol', 2),
('humaira', 2),
('colin', 2),
('wellbutrinsr', 2),
('rexult', 2),
('caltrate', 2),
('flit', 2),
('understated', 2),
('deodorizer', 2),
('immediatelly', 2),
('dughter', 2),
('effortful', 2),
('unwary', 2),
('balsamic', 2),
('geeked', 2),
('immunoassay', 2),
('amok', 2),
('natalia', 2),
('regullarly', 2),
('hygenine', 2),
('unabtainable', 2),
('begenning', 2),
('restock', 2),
('overian', 2),
('spasums', 2),
('suppise', 2),
('agai', 2),
('unbothered', 2),
('damm', 2),
('aromatose', 2),
('poopin', 2),
('platinum', 2),
('onit', 2),
('intiniv', 2),
('indentical', 2),
('intellence', 2),
('suckas', 2),
('bugginess', 2),
('combust', 2),
('quintessential', 2),
('mesured', 2),
('goofier', 2),
('premade', 2),
('evans', 2),
('maxsalt', 2),
('cafeful', 2),
('paraylisis', 2),
('camilia', 2),
('speacialist', 2),
('neaouroligist', 2),
('autisum', 2),
('bheavior', 2),
('medifasted', 2),
('aspro', 2),
('ergotomine', 2),
('propannalol', 2),
('cnm', 2),
('stastic', 2),
('guardian', 2),
('extacy', 2),
('platlettes', 2),
('duculox', 2),
('dcm', 2),
('daivonex', 2),
```

```
('genious', 2),
('papranoid', 2),
('lamisal', 2),
('laquer', 2),
('cicloporix', 2),
('ultrascans', 2),
('pregunte', 2),
('differente', 2),
('paralysising', 2),
('quercitin', 2),
('dao', 2),
('stupamax', 2),
('lavora', 2),
('escitlaopram', 2),
('whiched', 2),
('decmeber', 2),
('eerily', 2),
('verrry', 2),
('irrita', 2),
('denitely', 2),
('distort', 2),
('overcounter', 2),
('desiccant', 2),
('kosher', 2),
('unapologetically', 2),
('ballast', 2),
('emotitionaly', 2),
('menopur', 2),
('defibrosis', 2),
('hugger', 2),
('arid', 2),
('recentley', 2),
('pictured', 2),
('mulriple', 2),
('hypnotherapist', 2),
('ethier', 2),
('fibrillaiton', 2),
('fodmop', 2),
('wellchol', 2),
('rifaxin', 2),
('guiafenesin', 2),
('reqired', 2),
('rater', 2),
('trippin', 2),
('ptressure', 2),
('preescription', 2),
('stilled', 2),
('mediicine', 2),
('eeeuuu', 2),
('olutbreak', 2),
('zoviraz', 2),
('ufeff', 2),
('experiencies', 2),
('stasis', 2),
('pliva', 2),
('aipex', 2),
('integral', 2),
('knive', 2),
('hyperkalemia', 2),
('nother', 2),
('hadabsolutely', 2),
('detained', 2),
('itelf', 2),
('consistenly', 2),
('handfulls', 2),
```

```
('causede', 2),
('somwehat', 2),
('acidiphilus', 2),
('solgar', 2),
('zenca', 2),
('microtabs', 2),
('prbly', 2),
('bait', 2),
('stilnoct', 2),
('dycyclomine', 2),
('annoyying', 2),
('inlieu', 2),
('trulicy', 2),
('inmmarch', 2),
('benzalate', 2),
('frusrating', 2),
('jillian', 2),
('sonatata', 2),
('venoflaxalin', 2),
('muchhhhh', 2),
('torrent', 2),
('gastroenerologist', 2),
('belvid', 2),
('leptiburn', 2),
('eribulin', 2),
('hoard', 2),
('arranging', 2),
('sleepgels', 2),
('brkout', 2),
('edward', 2),
('waterpolo', 2),
('plastering', 2),
('amylodipine', 2),
('seperating', 2),
('agained', 2),
('insertedd', 2),
('dissocaitive', 2),
('naseauted', 2),
('pmds', 2),
('lerhargic', 2),
('mediines', 2),
('pyridine', 2),
('tyme', 2),
('tps', 2),
('grrrrrrrrr', 2),
('breathsavers', 2),
('carterizing', 2),
('unwisely', 2),
('accupril', 2),
('rhinorrhea', 2),
('nevre', 2),
('y', 2),
('flapper', 2),
('comminuted', 2),
('ridgidity', 2),
('apportion', 2),
('inconvenienced', 2),
('jake', 2),
('copxone', 2),
('plaid', 2),
('pillowy', 2),
('keprra', 2),
('asterpro', 2),
('winfrey', 2),
('magizine', 2),
```

```
 ('infinity', 2),
 ('peanutbutter', 2),
 ('regains', 2),
 ('diaraeh', 2),
 ('midnigtht', 2),
 ('megase', 2),
 ('auric', 2),
 ('carbapenum', 2),
 ('happenstance', 2),
 ('dyskensic', 2),
 ('bridgeport', 2),
 ('rivalled', 2),
 ('catherize', 2),
 ('paraylayzed', 2),
 ('unfixable', 2),
 ('trended', 2),
 ('fakeness', 2),
 ('pertained', 2),
 ('reconstituting', 2),
 ('levest', 2),
 ('moonth', 2),
 ('justs', 2),
 ('cramming', 2),
 ('betweem', 2),
 ('hamangioma', 2),
 ('envelops', 2),
 ('yohimbe', 2),
 ('relacore', 2),
 ('nuvigilshop', 2),
 ('ovca', 2),
 ('undegoing', 2),
 ('docetaxes', 2),
 ('gemcitabine', 2),
 ('nan', 2),
 ...]
```

This next part identifies the 2000 most common words in reviews that are "satisfied" (>=7) and "dissatified" (<=4).

Then, make a Naive Bayes classifier based on the unique words among these two lists of 2000 words each.

In [32]:

```
source_df['not_dissatisfied']=source_df['rating']>=4
source_df['satisfied']=source_df['rating']>=7

satisfied_review_words = source_df[source_df['satisfied']==True]['cleaned_review'].valu

dissatisfied_review_words = source_df[source_df['not_dissatisfied']==False]['cleaned_re

satisfied_review_words = [word for review in satisfied_review_words for word in review]
dissatisfied_review_words = [word for review in dissatisfied_review_words for word in r

satisfied_review_fdist    = nltk.FreqDist(satisfied_review_words)
dissatisfied_review_fdist = nltk.FreqDist(dissatisfied_review_words)

satisfied_review_common_words=[w[0] for w in satisfied_review_fdist.most_common(2000)]

dissatisfied_review_common_words=[w[0] for w in dissatisfied_review_fdist.most_common(2

d = {"Satisfied": satisfied_review_common_words, "Dissatisfied": dissatisfied_review_co
```

```
Review_Common_words_df = pd.DataFrame(data=d)

Review_Common_words_df
```

Out[32]:

	Satisfied	Dissatisfied
0	effect	taking
1	side	pain
2	year	pill
3	take	month
4	taking	like
...	...	...
1995	magnesium	cialis
1996	picked	toenail
1997	flexeril	reduction
1998	proper	explosive
1999	passing	butt

2000 rows × 2 columns

In [33]:

```
Review_Common_words_df.head(25)
```

Out[33]:

	Satisfied	Dissatisfied
0	effect	taking
1	side	pain
2	year	pill
3	take	month
4	taking	like
5	pain	effect
6	get	started
7	work	get
8	started	would
9	like	day
10	month	side
11	day	period
12	feel	took
13	pill	take
14	period	doctor

	Satisfied	Dissatisfied
15	would	never
16	medication	feel
17	life	got
18	doctor	medication
19	week	week
20	medicine	could
21	weight	work
22	since	bad
23	took	still
24	really	year

Next is the unique words among these two lists of 2000 words each.

In [34]:

```
review_common_words = set(satisfied_review_common_words + dissatisfied_review_common_wo
```

In [35]:

```
len(review_common_words)
```

Out[35]: 2314

Create the document feature extractor

In [36]:

```
def document_features(document, word_features):
    document_words = set(document)
    features = {}

    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    return features
```

Prepare the feature set

In [37]:

```
total_review_list = source_df['review'].values.tolist()
satisfied_status_list = source_df['satisfied'].values.tolist()
```

In [38]:

```
total_review_list = [nltk.word_tokenize(review) for review in total_review_list[:5000]]
satisfied_status_list = satisfied_status_list[:5000]
```

In [39]:

```
documents = list(zip(total_review_list, satisfied_status_list))
```

```python
featuresets = [(document_features(d, review_common_words), c) for (d,c) in documents]

featuresets_df = pd.DataFrame(data=featuresets,columns=['Feature', 'Satisfied'])

featuresets_df
```

Out[39]:

	Feature	Satisfied
0	{'contains(sense)': False, 'contains(exhausted...	True
1	{'contains(sense)': False, 'contains(exhausted...	True
2	{'contains(sense)': False, 'contains(exhausted...	True
3	{'contains(sense)': False, 'contains(exhausted...	True
4	{'contains(sense)': False, 'contains(exhausted...	True
...	...	...
4995	{'contains(sense)': False, 'contains(exhausted...	False
4996	{'contains(sense)': False, 'contains(exhausted...	True
4997	{'contains(sense)': False, 'contains(exhausted...	True
4998	{'contains(sense)': False, 'contains(exhausted...	True
4999	{'contains(sense)': False, 'contains(exhausted...	True

5000 rows × 2 columns

In [40]:

```python
len(featuresets_df["Feature"][0])
```

Out[40]: 2314

In [41]:

```python
d = featuresets_df["Feature"][0]
key_list = list(d.keys())

size = int(.9*len(featuresets))
train_set, test_set = featuresets[:size], featuresets[size:]

test_set_X = featuresets_df["Feature"][size:]
test_set_y = featuresets_df["Satisfied"][size:]
```

In [42]:

```python
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

In [43]:

```python
classifier.show_most_informative_features(15)
```

```
Most Informative Features
             contains(lo) = True           False : True   =     12.3 : 1.0
        contains(pleased) = True            True : False  =     10.2 : 1.0
   contains(discontinued) = True           False : True   =      9.7 : 1.0
         contains(poison) = True           False : True   =      9.7 : 1.0
        contains(rubbish) = True           False : True   =      9.7 : 1.0
      contains(tightness) = True           False : True   =      9.7 : 1.0
```

```
       contains(vagina) = True           False : True   =      9.1 : 1.0
    contains(confusion) = True           False : True   =      8.6 : 1.0
         contains(poor) = True           False : True   =      8.1 : 1.0
        contains(chose) = True            True : False  =      7.7 : 1.0
       contains(easier) = True            True : False  =      7.3 : 1.0
         contains(crap) = True           False : True   =      7.1 : 1.0
        contains(labor) = True           False : True   =      7.1 : 1.0
 contains(testosterone) = True           False : True   =      7.1 : 1.0
        contains(flare) = True            True : False  =      7.1 : 1.0
```

Metrics

In [44]:

```python
# Identify the predicted value of the classifier
test_y_est = [classifier.classify(test_val) for test_val in test_set_X]

# Identify the true value of the variable
test_y = list(test_set_y)

accuracy_NB, precision_NB, recall_NB, f1_NB = get_NBmetrics(test_y_est,test_y)
print("accuracy = %.10f, precision = %.10f, recall = %.10f, f1 = %.10f" % (accuracy_NB,
```

```
accuracy = 0.730, precision = 0.809, recall = 0.797, f1 = 0.803
```

TFIDF Bag of Words

In [45]:

```python
# Apply the CountVectorizer to the "cleaned_reviews_raw_cv" column
list_features = source_df['cleaned_review_raw_cv'].values.tolist()
vectorizer = CountVectorizer(analyzer='word', ngram_range=(1, 2))
vectorized_list=vectorizer.fit_transform(list_features)

# Split into testing and training sets
source_df['class_label']= np.where(source_df['rating']>=7,1,0)
list_labels = source_df["class_label"].values
```

In [46]:

```python
# Applying the vectorizer to the "cleaned_reviews_raw_cv" column
vectorizer_TF = TfidfVectorizer(analyzer='word', ngram_range=(1, 2))
vectorized_list_TF=vectorizer_TF.fit_transform(list_features)
```

In [47]:

```python
# Splitting the model
X_train_TF, X_test_TF, y_train_TF, y_test_TF = train_test_split(vectorized_list_TF, lis
```

In [48]:

```python
clf_TF = LogisticRegression(C=30, class_weight='balanced', solver='sag',
                         multi_class='multinomial', n_jobs=-1, random_state=40,
                         verbose=1, max_iter = 1000)
clf_TF.fit(X_train_TF, y_train_TF)

y_predicted_counts_TF = clf_TF.predict(X_test_TF)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
convergence after 81 epochs took 103 seconds
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:  1.7min finished
```

Metrics

In [49]:

```
accuracy_TF, precision_TF, recall_TF, f1_TF = get_metrics(y_test_TF, y_predicted_counts
print("accuracy = %.10f, precision = %.10f, recall = %.10f, f1 = %.10f" % (accuracy_TF,
```

```
accuracy = 0.9472264844, precision = 0.9471968203, recall = 0.9472264844, f1 = 0.9472109
966
```

In [50]:

```
def get_most_important_features(vectorizer, model, n=5):
    index_to_word = {v:k for k,v in vectorizer.vocabulary_.items()}

    # loop for each class
    classes ={}
    for class_index in range(model.coef_.shape[0]):
        word_importances = [(el, index_to_word[i]) for i,el in enumerate(model.coef_[cl
        sorted_coeff = sorted(word_importances, key = lambda x : x[0], reverse=True)
        tops = sorted(sorted_coeff[:n], key = lambda x : x[0])
        bottom = sorted_coeff[-n:]
        classes[class_index] = {
            'tops':tops,
            'bottom':bottom
        }
    return classes

importance = get_most_important_features(vectorizer_TF, clf_TF, 10)
```

In [51]:

```
# Define important words plot function and plot Bag-of-Words' important words

def plot_important_words(top_scores, top_words, bottom_scores, bottom_words, name):
    y_pos = np.arange(len(top_words))
    top_pairs = [(a,b) for a,b in zip(top_words, top_scores)]
    top_pairs = sorted(top_pairs, key=lambda x: x[1])

    bottom_pairs = [(a,b) for a,b in zip(bottom_words, bottom_scores)]
    bottom_pairs = sorted(bottom_pairs, key=lambda x: x[1], reverse=True)

    top_words = [a[0] for a in top_pairs]
    top_scores = [a[1] for a in top_pairs]

    bottom_words = [a[0] for a in bottom_pairs]
    bottom_scores = [a[1] for a in bottom_pairs]

    fig = plt.figure(figsize=(10, 10))

    plt.subplot(121)
    plt.barh(y_pos,bottom_scores, align='center', alpha=0.5)
    plt.title('Not Satisfied Reviews', fontsize=20)
    plt.yticks(y_pos, bottom_words, fontsize=14)
    plt.suptitle('Key words', fontsize=16)
    plt.xlabel('Importance', fontsize=20)

    plt.subplot(122)
    plt.barh(y_pos,top_scores, align='center', alpha=0.5)
    plt.title('Satisfied Reviews', fontsize=20)
    plt.yticks(y_pos, top_words, fontsize=14)
    plt.suptitle(name, fontsize=16)
    plt.xlabel('Importance', fontsize=20)
```

```python
    plt.subplots_adjust(wspace=0.8)
    plt.show()

top_scores = [a[0] for a in importance[0]['tops']]
top_words = [a[1] for a in importance[0]['tops']]
bottom_scores = [a[0] for a in importance[0]['bottom']]
bottom_words = [a[1] for a in importance[0]['bottom']]

plot_important_words(top_scores, top_words, bottom_scores, bottom_words, "Most importan
```

In [159...

```python
print(sorted(importance[0]['tops']))
print(sorted(importance[0]['bottom']))
```

```
[(8.704972818433665, 'year'), (8.860092707105201, 'lifesaver'), (9.314441336364569, 'won
derful'), (9.781721512045777, 'great'), (10.097359637683658, 'best'), (11.20748934091292
5, 'changed life'), (12.088171232955435, 'saved'), (12.8346391878291, 'amazing'), (13.42
6081762773075, 'miracle'), (16.168100629673418, 'love')]
[(-11.503974325694982, 'worse'), (-10.442572487005595, 'disappointed'), (-8.500149832434
271, 'worst'), (-7.776868978077935, 'would love'), (-7.339891200311777, 'useless'), (-7.
158254139988663, 'waste'), (-7.117206000351102, 'removed'), (-6.629846408346388, 'horrib
le'), (-6.364549056638236, 'never recommend'), (-6.31649110048635, 'depressed')]
```

VADER Sentiment Analysis

In [52]:

```
# Derive the VADER polarity scores (sentiment) for each of the raw reviews using the VA
sa = SentimentIntensityAnalyzer()
source_df['vader_polarity_scores']=source_df['cleaned_review_raw_vader'].apply(lambda r
```

In [53]:

```
# Exract the VADER compound polarity score for each of the reviews
source_df['vader_compound_polarity_score']=source_df['vader_polarity_scores'].apply(lam
```

In [54]:

```
# Inspect a subset of the classified reviews to confirm the output aligns with expectat
source_df.head(5)
```

Out[54]:

	uniqueID	drugName	condition	review	rating	date	usefulCount	sentiment	cleaned_re
0	163740	Mirtazapine	depression	"I've tried a few antidepressants over th...	10	28-Feb-12	22	Satisfied	[antidepre: citalo|
1	206473	Mesalamine	crohn's disease, maintenance	"My son has Crohn's disease and has done ...	8	17-May-09	17	Satisfied	[son, c disease, well, a cc
2	159672	Bactrim	urinary tract infection	"Quick reduction of symptoms"	9	29-Sep-17	3	Satisfied	[(redu symp
3	39293	Contrave	weight loss	"Contrave combines drugs that were used for al...	9	5-Mar-17	35	Satisfied	[con combine, used, alc sn
4	97768	Cyclafem 1 / 35	birth control	"I have been on this birth control for one cyc...	9	22-Oct-15	4	Satisfied	[birth, co cycle, rea review, t

5 rows × 22 columns

Count Vectorizer & LDA Aspect Modeling

In [55]:

```
# Vectorized the cleaned reviews.
# We can consider both unigrams and bigrams by using ngram_range=(1,2)
# We can consider using min_df because there are many medical terms which only appear a
# We can consider using max_df to try to eliminate corpus specific stopwords
# The min_df and max_df are reflected in (min_df = 10, max_df = 25,000)
vectorizer = CountVectorizer(analyzer='word', ngram_range=(1, 1), min_df=10, max_df=250
```

```
cv_review_list = source_df['cleaned_review_raw_cv'].values.tolist()
cv_review_vector = vectorizer.fit_transform(cv_review_list)
```

In [56]:
```
# Prepare the LDA model
lda_model = LatentDirichletAllocation(n_components = 5, random_state = 1, n_jobs = -1)
lda_output = lda_model.fit_transform(cv_review_vector)
```

In [57]:
```
# Determine the primary aspect for each review
topic_names = ["topic_" + str(i) for i in range(lda_model.n_components)]
df_document_topic = pd.DataFrame(np.round(lda_output, 2), columns = topic_names)
dominant_topic = (np.argmax(df_document_topic.values, axis=1))
df_document_topic['dominant_topic'] = dominant_topic
df_document_topic['dominant_topic'] = 'topic_' + df_document_topic['dominant_topic'].as
```

In [58]:
```
# Check that the row count for the source_df and LDA output are the same since we will
print(df_document_topic.shape[0])
print(source_df.shape[0])
```

```
215063
215063
```

In [59]:
```
# Reset the index for the source_df and LDA output to ensure the join happens positiona
df_document_topic=df_document_topic.reset_index(drop=True)
source_df=source_df.reset_index(drop=True)
```

In [60]:
```
# Join results into the original dataframe
source_df=source_df.join(df_document_topic)
```

In [61]:
```
# Inspect subset of source_df to confirm output is as expected
source_df.head(5)
```

Out[61]:

	uniqueID	drugName	condition	review	rating	date	usefulCount	sentiment	cleaned_re
0	163740	Mirtazapine	depression	"I've tried a few antidepressants over th...	10	28-Feb-12	22	Satisfied	[antidepre: citalo|
1	206473	Mesalamine	crohn's disease, maintenance	"My son has Crohn's disease and has done ...	8	17-May-09	17	Satisfied	[son, c disease, well, a co
2	159672	Bactrim	urinary tract infection	"Quick reduction of symptoms"	9	29-Sep-17	3	Satisfied	[(redu symp
3	39293	Contrave	weight loss	"Contrave combines drugs that were used for al...	9	5-Mar-17	35	Satisfied	[con combine, used, alc sn

	uniqueID	drugName	condition	review	rating	date	usefulCount	sentiment	cleaned_re
4	97768	Cyclafem 1 / 35	birth control	"I have been on this birth control for one cyc...	9	22-Oct-15	4	Satisfied	[birth, co cycle, rea review, t

5 rows × 28 columns

Topic Discovery

In [62]:

```
# Create a CFD by topic
review_words = source_df['cleaned_review'].values.tolist()
review_topic = source_df['dominant_topic'].values.tolist()
cfd_topic_words = list(zip(review_words, review_topic))
cfd = nltk.ConditionalFreqDist((review[1], word) for review in cfd_topic_words for word
```

In [63]:

```
# Explore the common words associated with topic 0 to infer what the topic is: Skin Hea
cfd['topic_0'].most_common(50)
```

Out[63]:

```
[('period', 10733),
 ('pain', 8708),
 ('get', 8611),
 ('like', 8403),
 ('day', 8392),
 ('would', 7834),
 ('month', 7814),
 ('got', 7767),
 ('year', 7031),
 ('skin', 6940),
 ('work', 6268),
 ('use', 6216),
 ('using', 6070),
 ('week', 5796),
 ('bad', 5703),
 ('doctor', 5688),
 ('bleeding', 5661),
 ('side', 5333),
 ('since', 5333),
 ('used', 5323),
 ('started', 5274),
 ('cramp', 5211),
 ('effect', 5202),
 ('take', 5130),
 ('insertion', 5120),
 ('pill', 5102),
 ('never', 4981),
 ('feel', 4786),
 ('product', 4726),
 ('went', 4553),
 ('still', 4539),
 ('really', 4404),
 ('acne', 4371),
 ('go', 4201),
 ('every', 4130),
 ('took', 3994),
```

```
('felt', 3898),
('inserted', 3890),
('put', 3832),
('painful', 3811),
('much', 3791),
('face', 3752),
('could', 3661),
('mirena', 3651),
('cream', 3541),
('little', 3497),
('getting', 3484),
('cramping', 3317),
('nothing', 3251),
('review', 3250)]
```

In [64]:

```
# Explore the common words associated with topic 1 to infer what the topic is: Pain
cfd['topic_1'].most_common(50)
```

Out[64]:

```
[('pain', 34052),
 ('year', 16825),
 ('effect', 15136),
 ('side', 14682),
 ('take', 14253),
 ('taking', 13024),
 ('work', 11883),
 ('medication', 10624),
 ('doctor', 10620),
 ('medicine', 9385),
 ('get', 9378),
 ('month', 9064),
 ('drug', 9020),
 ('started', 8442),
 ('life', 8084),
 ('migraine', 8027),
 ('day', 7798),
 ('would', 7400),
 ('like', 7257),
 ('feel', 6384),
 ('since', 6328),
 ('well', 6289),
 ('help', 6137),
 ('could', 5977),
 ('severe', 5931),
 ('week', 5913),
 ('still', 5771),
 ('took', 5694),
 ('headache', 5483),
 ('every', 5431),
 ('much', 5240),
 ('prescribed', 5149),
 ('relief', 5088),
 ('tried', 5085),
 ('better', 5058),
 ('worked', 4934),
 ('great', 4920),
 ('went', 4865),
 ('dose', 4509),
 ('bad', 4490),
 ('pill', 4431),
 ('go', 4423),
 ('got', 4221),
 ('problem', 4200),
 ('injection', 4177),
```

```
 ('hour', 4171),
 ('good', 4161),
 ('med', 4105),
 ('without', 4036),
 ('put', 3988)]
```

In [65]:

```
# Explore the common words associated with topic 2 to infer what the topic is: Gastro H
cfd['topic_2'].most_common(50)
```

Out[65]:

```
[('taking', 16145),
 ('effect', 15877),
 ('side', 15763),
 ('take', 14119),
 ('day', 13477),
 ('started', 13061),
 ('took', 10865),
 ('work', 10574),
 ('like', 10404),
 ('feel', 10264),
 ('pain', 10263),
 ('get', 10145),
 ('pill', 9551),
 ('weight', 9530),
 ('medication', 8719),
 ('doctor', 8449),
 ('lost', 7916),
 ('medicine', 7686),
 ('would', 7356),
 ('week', 7348),
 ('eat', 7297),
 ('nausea', 7102),
 ('stomach', 7088),
 ('still', 7082),
 ('dose', 6854),
 ('go', 6839),
 ('bad', 6612),
 ('went', 6547),
 ('hour', 6464),
 ('year', 6453),
 ('water', 6347),
 ('felt', 6280),
 ('month', 6226),
 ('better', 6217),
 ('much', 6178),
 ('could', 6111),
 ('really', 5696),
 ('headache', 5616),
 ('infection', 5592),
 ('got', 5437),
 ('good', 5412),
 ('well', 5147),
 ('morning', 5130),
 ('blood', 5129),
 ('feeling', 5033),
 ('pound', 4985),
 ('prescribed', 4912),
 ('lb', 4866),
 ('make', 4764),
 ('going', 4755)]
```

In [66]:

```
# Explore the common words associated with topic 3 to infer what the topic is: Menstrua
cfd['topic_3'].most_common(50)
```

Out[66]:

```
[('period', 38942),
 ('pill', 30706),
 ('month', 22071),
 ('control', 19987),
 ('birth', 18985),
 ('acne', 17222),
 ('weight', 16590),
 ('get', 16088),
 ('side', 14034),
 ('started', 13753),
 ('effect', 13709),
 ('taking', 13553),
 ('day', 13270),
 ('got', 13033),
 ('would', 12541),
 ('year', 11962),
 ('sex', 11655),
 ('like', 11612),
 ('mood', 11459),
 ('since', 10073),
 ('never', 9780),
 ('week', 9352),
 ('bleeding', 9271),
 ('gain', 9039),
 ('swing', 8591),
 ('cramp', 8572),
 ('bad', 8566),
 ('take', 8423),
 ('really', 8316),
 ('skin', 7969),
 ('feel', 7905),
 ('every', 7379),
 ('took', 7363),
 ('gained', 6874),
 ('getting', 6855),
 ('work', 6736),
 ('drive', 6245),
 ('went', 6195),
 ('spotting', 6150),
 ('pregnant', 6075),
 ('still', 5985),
 ('little', 5952),
 ('much', 5925),
 ('great', 5856),
 ('doctor', 5808),
 ('made', 5794),
 ('experience', 5506),
 ('good', 5271),
 ('pain', 5056),
 ('ever', 5033)]
```

In [67]:

```python
# Explore the common words associated with topic 4 to infer what the topic is: Mental H
cfd['topic_4'].most_common(50)
```

Out[67]:

```
[('anxiety', 25768),
 ('effect', 23600),
 ('taking', 22341),
 ('feel', 21221),
 ('side', 20736),
 ('take', 20600),
 ('year', 19839),
 ('like', 18100),
 ('medication', 16747),
```

```
 ('sleep', 16507),
 ('life', 16256),
 ('started', 15950),
 ('work', 15919),
 ('depression', 15842),
 ('get', 13985),
 ('would', 13303),
 ('medicine', 12424),
 ('month', 11628),
 ('doctor', 11554),
 ('help', 11393),
 ('week', 11145),
 ('felt', 10950),
 ('better', 10848),
 ('day', 10716),
 ('could', 10396),
 ('much', 10044),
 ('drug', 9767),
 ('took', 9519),
 ('really', 9360),
 ('still', 9157),
 ('feeling', 9130),
 ('good', 8641),
 ('tried', 8587),
 ('well', 8334),
 ('great', 7990),
 ('panic', 7984),
 ('dose', 7960),
 ('attack', 7882),
 ('thought', 7794),
 ('made', 7774),
 ('go', 7735),
 ('since', 7360),
 ('bad', 7231),
 ('went', 6988),
 ('helped', 6924),
 ('never', 6909),
 ('make', 6884),
 ('weight', 6766),
 ('prescribed', 6443),
 ('worked', 6155)]
```

In [68]:

```
d = {'Skin Health':cfd['topic_0'].most_common(50),
     'Pain':cfd['topic_1'].most_common(50),
     'Gastro Health':cfd['topic_2'].most_common(50),
     'Menstrual Health':cfd['topic_3'].most_common(50),
     'Mental Health':cfd['topic_4'].most_common(50)}

TOPICS_df = pd.DataFrame(data=d)

TOPICS_df.head(20)
```

Out[68]:

	Skin Health	Pain	Gastro Health	Menstrual Health	Mental Health
0	(period, 10733)	(pain, 34052)	(taking, 16145)	(period, 38942)	(anxiety, 25768)
1	(pain, 8708)	(year, 16825)	(effect, 15877)	(pill, 30706)	(effect, 23600)
2	(get, 8611)	(effect, 15136)	(side, 15763)	(month, 22071)	(taking, 22341)
3	(like, 8403)	(side, 14682)	(take, 14119)	(control, 19987)	(feel, 21221)
4	(day, 8392)	(take, 14253)	(day, 13477)	(birth, 18985)	(side, 20736)

	Skin Health	Pain	Gastro Health	Menstrual Health	Mental Health
5	(would, 7834)	(taking, 13024)	(started, 13061)	(acne, 17222)	(take, 20600)
6	(month, 7814)	(work, 11883)	(took, 10865)	(weight, 16590)	(year, 19839)
7	(got, 7767)	(medication, 10624)	(work, 10574)	(get, 16088)	(like, 18100)
8	(year, 7031)	(doctor, 10620)	(like, 10404)	(side, 14034)	(medication, 16747)
9	(skin, 6940)	(medicine, 9385)	(feel, 10264)	(started, 13753)	(sleep, 16507)
10	(work, 6268)	(get, 9378)	(pain, 10263)	(effect, 13709)	(life, 16256)
11	(use, 6216)	(month, 9064)	(get, 10145)	(taking, 13553)	(started, 15950)
12	(using, 6070)	(drug, 9020)	(pill, 9551)	(day, 13270)	(work, 15919)
13	(week, 5796)	(started, 8442)	(weight, 9530)	(got, 13033)	(depression, 15842)
14	(bad, 5703)	(life, 8084)	(medication, 8719)	(would, 12541)	(get, 13985)
15	(doctor, 5688)	(migraine, 8027)	(doctor, 8449)	(year, 11962)	(would, 13303)
16	(bleeding, 5661)	(day, 7798)	(lost, 7916)	(sex, 11655)	(medicine, 12424)
17	(side, 5333)	(would, 7400)	(medicine, 7686)	(like, 11612)	(month, 11628)
18	(since, 5333)	(like, 7257)	(would, 7356)	(mood, 11459)	(doctor, 11554)
19	(used, 5323)	(feel, 6384)	(week, 7348)	(since, 10073)	(help, 11393)

In [69]:

```
source_df.head()
```

Out[69]:

	uniqueID	drugName	condition	review	rating	date	usefulCount	sentiment	cleaned_re
0	163740	Mirtazapine	depression	"I've tried a few antidepressants over th...	10	28-Feb-12	22	Satisfied	[antidepre citalo
1	206473	Mesalamine	crohn's disease, maintenance	"My son has Crohn's disease and has done ...	8	17-May-09	17	Satisfied	[son, c disease, well, a co
2	159672	Bactrim	urinary tract infection	"Quick reduction of symptoms"	9	29-Sep-17	3	Satisfied	[redu symp
3	39293	Contrave	weight loss	"Contrave combines drugs that were used for al...	9	5-Mar-17	35	Satisfied	[con combine, used, alc sn
4	97768	Cyclafem 1 / 35	birth control	"I have been on this birth control for one cyc...	9	22-Oct-15	4	Satisfied	[birth, co cycle, rea review, t

5 rows × 28 columns

In [70]:

```
STATS0_DF = source_df.groupby('dominant_topic', as_index=False).agg({"vader_compound_po
STATS0_DF
```

Out[70]:

	dominant_topic	vader_compound_polarity_score	drugName
0	topic_0	0.067487	28059
1	topic_1	-0.054039	44955
2	topic_2	-0.017558	43022
3	topic_3	0.002476	42262
4	topic_4	0.016992	56765

In [71]:

```
Topics = TOPICS_df.columns
```

In [72]:

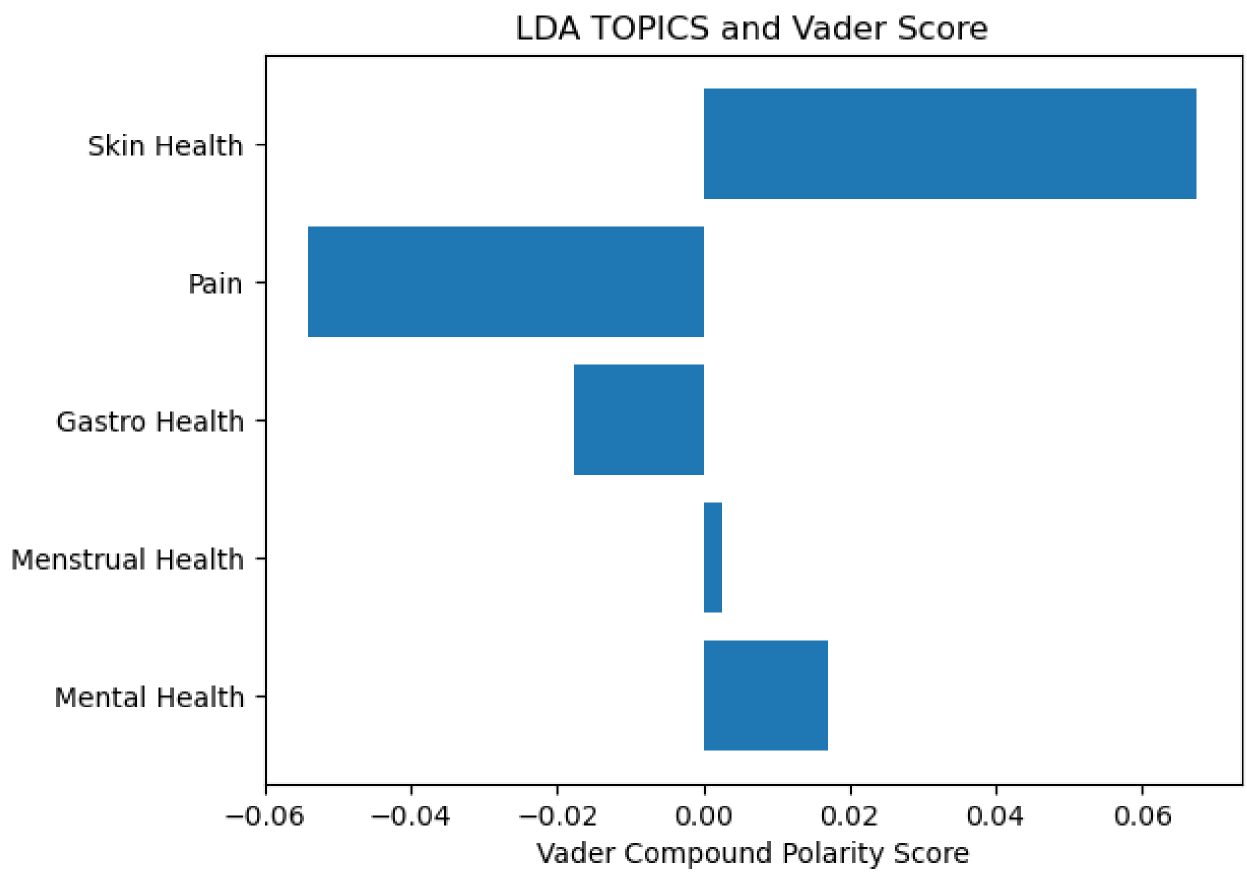
```
plt.rcdefaults()
fig, ax = plt.subplots()

ax.barh(Topics, STATS0_DF["vader_compound_polarity_score"], align='center')

#ax.set_yticks(STATS0_DF["dominant_topic"], labels=str(Topics)
ax.invert_yaxis()  # labels read top-to-bottom

ax.set_xlabel('Vader Compound Polarity Score')
ax.set_title('LDA TOPICS and Vader Score')

plt.show()
```



Aggregate the Sentiment Score for Each Topic By Drug

In [73]:

```
# Derive compound polarity score column for a given topic
source_df['topic_0_compound_polarity_score'] = source_df["vader_compound_polarity_score
source_df['topic_1_compound_polarity_score'] = source_df["vader_compound_polarity_score
source_df['topic_2_compound_polarity_score'] = source_df["vader_compound_polarity_score
source_df['topic_3_compound_polarity_score'] = source_df["vader_compound_polarity_score
source_df['topic_4_compound_polarity_score'] = source_df["vader_compound_polarity_score
```

In [74]:

```
len(source_df)
```

Out[74]: 215063

In [75]:

```
# Verify output is as expected
source_df[["vader_compound_polarity_score",'topic_0_compound_polarity_score','topic_1_c
```

Out[75]:

	vader_compound_polarity_score	topic_0_compound_polarity_score	topic_1_compound_polarity_score	to
0	-0.5267	-0.005267	-0.005267	
1	0.7539	0.007539	0.603120	
2	0.0000	0.000000	0.000000	
3	0.6810	0.000000	0.074910	

	vader_compound_polarity_score	topic_0_compound_polarity_score	topic_1_compound_polarity_score	to
4	0.9559	0.000000	0.000000	

In [76]:
```
source_df.columns
```

Out[76]:
```
Index(['uniqueID', 'drugName', 'condition', 'review', 'rating', 'date',
       'usefulCount', 'sentiment', 'cleaned_review', 'drugName_list',
       'drugName_list3', 'BrandName', 'Spending', 'drugName_list2',
       'cleaned_review_raw_vader', 'cleaned_review_raw_cv', 'Grade',
       'not_dissatisfied', 'satisfied', 'class_label', 'vader_polarity_scores',
       'vader_compound_polarity_score', 'topic_0', 'topic_1', 'topic_2',
       'topic_3', 'topic_4', 'dominant_topic',
       'topic_0_compound_polarity_score', 'topic_1_compound_polarity_score',
       'topic_2_compound_polarity_score', 'topic_3_compound_polarity_score',
       'topic_4_compound_polarity_score'],
      dtype='object')
```

In [77]:
```
import copy
source_df0 = copy.deepcopy(source_df)
```

In [78]:
```
# Select the columns that are relevant for the drug efficacy model
source_df=source_df[['drugName','condition','rating','Grade','Spending',"vader_compound
```

In [79]:
```
# Create a fdist to find the most common conditions
# Limit number of conditions to 50 since they provide more than 75% coverage for the re
# This is to reduce the number of dummy variables.
# The rest of the reviews will be bucketed into other.
conditions = source_df['condition'].values.tolist()
conditions_fdist = nltk.FreqDist(conditions)
total_count = len([w for (w,c) in conditions_fdist.most_common()])
common_count = sum([c for (w,c) in conditions_fdist.most_common(50)])
common_conditions = [w for (w,c) in conditions_fdist.most_common(50)]
print('total conditions:',total_count)
print('common condition coverage:',common_count/source_df.shape[0])
```

```
total conditions: 917
common condition coverage: 0.7664777297815059
```

In [80]:
```
# See what are the 50 most common conditions
common_conditions
```

Out[80]:
```
['birth control',
 'depression',
 'pain',
 'anxiety',
 'acne',
 'bipolar disorde',
 'insomnia',
 'weight loss',
 'obesity',
 'adhd',
 'diabetes, type 2',
 'emergency contraception',
```

```
 'high blood pressure',
 'vaginal yeast infection',
 'abnormal uterine bleeding',
 'bowel preparation',
 'smoking cessation',
 'ibromyalgia',
 'migraine',
 'anxiety and stress',
 'major depressive disorde',
 'constipation',
 'chronic pain',
 'panic disorde',
 'migraine prevention',
 'urinary tract infection',
 'muscle spasm',
 'osteoarthritis',
 'generalized anxiety disorde',
 'opiate dependence',
 'erectile dysfunction',
 'irritable bowel syndrome',
 'allergic rhinitis',
 'rheumatoid arthritis',
 'bacterial infection',
 'cough',
 nan,
 'sinusitis',
 'nausea/vomiting',
 'gerd',
 'hyperhidrosis',
 'overactive bladde',
 'multiple sclerosis',
 'hepatitis c',
 'hiv infection',
 'high cholesterol',
 'back pain',
 'restless legs syndrome',
 'psoriasis',
 'schizophrenia']
```

In [81]:

```
# Replace NaN with other
common_conditions[common_conditions.index(np.nan)]='other'
```

In [82]:

```
# Verify that NaNs were replaced as expected.
common_conditions
```

Out[82]:

```
['birth control',
 'depression',
 'pain',
 'anxiety',
 'acne',
 'bipolar disorde',
 'insomnia',
 'weight loss',
 'obesity',
 'adhd',
 'diabetes, type 2',
 'emergency contraception',
 'high blood pressure',
 'vaginal yeast infection',
 'abnormal uterine bleeding',
 'bowel preparation',
 'smoking cessation',
```

```
 'ibromyalgia',
 'migraine',
 'anxiety and stress',
 'major depressive disorde',
 'constipation',
 'chronic pain',
 'panic disorde',
 'migraine prevention',
 'urinary tract infection',
 'muscle spasm',
 'osteoarthritis',
 'generalized anxiety disorde',
 'opiate dependence',
 'erectile dysfunction',
 'irritable bowel syndrome',
 'allergic rhinitis',
 'rheumatoid arthritis',
 'bacterial infection',
 'cough',
 'other',
 'sinusitis',
 'nausea/vomiting',
 'gerd',
 'hyperhidrosis',
 'overactive bladde',
 'multiple sclerosis',
 'hepatitis c',
 'hiv infection',
 'high cholesterol',
 'back pain',
 'restless legs syndrome',
 'psoriasis',
 'schizophrenia']
```

In [83]:

```
# Replace uncommon conditions with other
source_df['condition']= source_df['condition'].apply(lambda condition: condition if con
```

In [84]:

```
# Groupby the drug name and condition while taking an average for each numeric column
drug_efficacy_df =source_df.groupby(by=['drugName','condition'],as_index = False).mean(
```

In [85]:

```
# Verify the output is as expected
print(drug_efficacy_df.shape)
drug_efficacy_df.head(5)
```

```
(5711, 11)
```

Out[85]:

	drugName	condition	rating	Grade	Spending	vader_compound_polarity_score	topic_0_comp
0	A + D Cracked Skin Relief	other	10.000000	24.700000	NaN	0.178800	
1	A / B Otic	other	10.000000	3.300000	NaN	-0.015600	
2	Abacavir / dolutegravir / lamivudine	hiv infection	8.414286	11.510000	NaN	0.206616	
3	Abacavir / lamivudine	hiv infection	10.000000	12.033333	26.98	0.336167	

	drugName	condition	rating	Grade	Spending	vader_compound_polarity_score	topic_0_comp
4	Abacavir / lamivudine / zidovudine	hiv infection	9.000000	3.900000	21.53	-0.077200	

In [86]:

```
# Import Bokeh libraries that are relevant for data visualization via diagrams

import pandas_bokeh

from bokeh.resources import INLINE
import bokeh.io
bokeh.io.output_notebook(INLINE)

# Import the figure and gridplot objects
from bokeh.plotting import figure
from bokeh.layouts import gridplot

from bokeh.io import output_file, show

from bokeh.models import FactorRange

from bokeh.transform import dodge

# ColumnDataSource is Bokeh's native data structure similar to a Pandas dataframe.
from bokeh.models import ColumnDataSource

# Used to change the color and shape of the markers
from bokeh.transform import factor_cmap, factor_mark
```

BokehJS 2.3.2 successfully loaded.

In [87]:

```
STATS1_DF = drug_efficacy_df.groupby('condition', as_index=False).agg({"rating":"mean",

STATS1_DF.reset_index()
STATS1_DF.set_index('condition')

# Drop "other" as a condition (done for graphing purposes)
STATS1_DF.drop(STATS1_DF.loc[STATS1_DF.condition=="other"].index, inplace=True)

STATS1_DF.reset_index(drop=True, inplace=True)

# Sort with respect to most medicines treating a condition
STATS1_DF = STATS1_DF.sort_values(by=["drugName"], ascending=False)

STATS1_DF.head()
```

Out[87]:

	condition	rating	vader_compound_polarity_score	Grade	drugName
38	pain	7.704966	-0.136220	9.238538	219
9	birth control	5.625135	-0.053215	8.725545	181

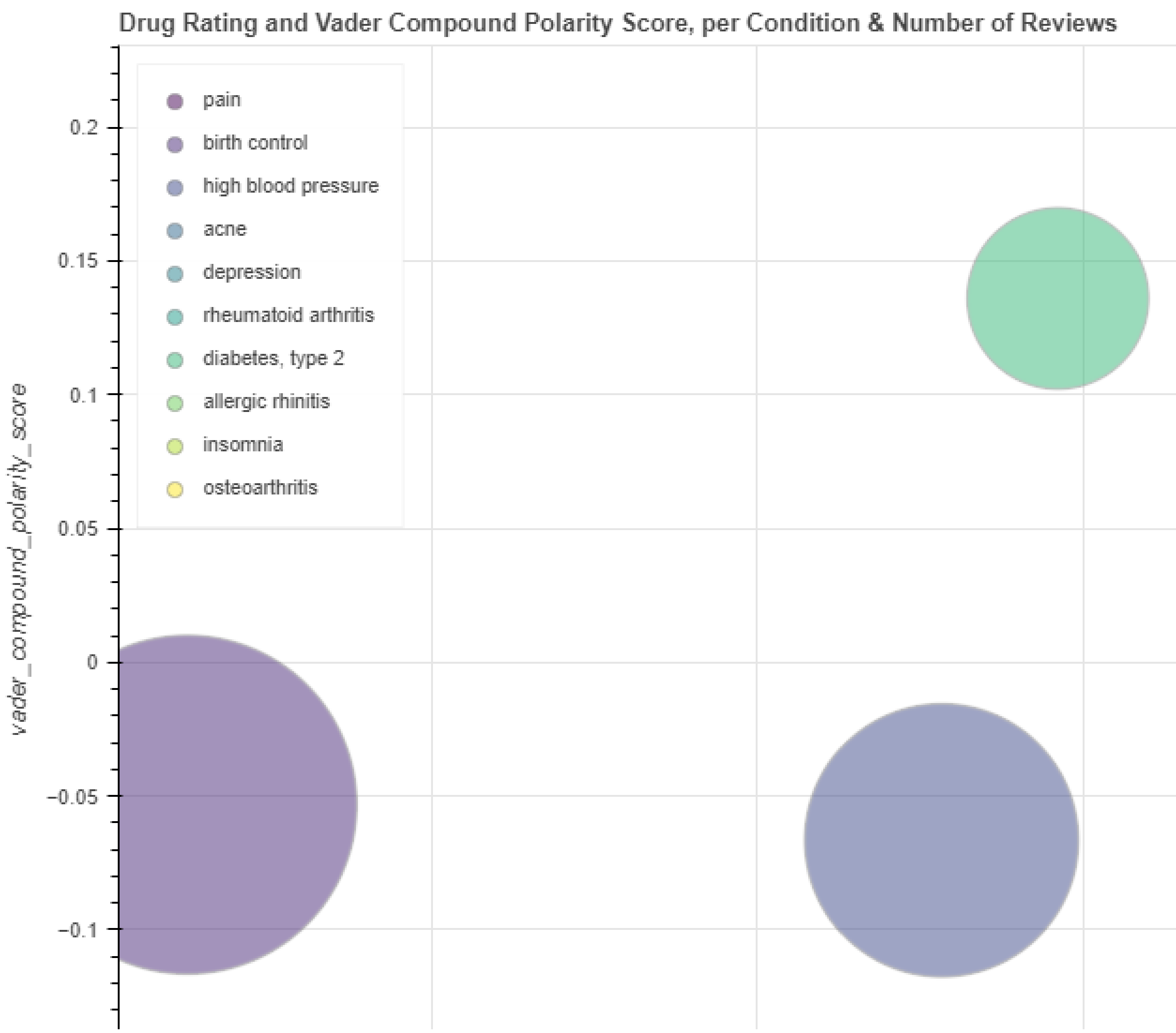
	condition	rating	vader_compound_polarity_score	Grade	drugName
21	high blood pressure	6.782431	-0.066555	10.199119	146
1	acne	7.554865	0.213127	8.083321	127
14	depression	7.697184	-0.058354	11.408852	115

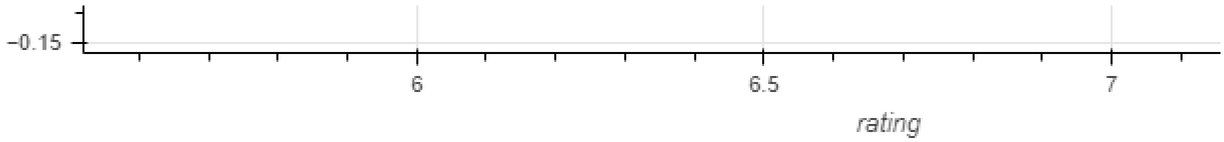
In [88]:

```
bkPlot_LL = STATS1_DF[:10].plot_bokeh.scatter('rating','vader_compound_polarity_score',
                                     figsize=(900,620),
                                     category='condition',
                                     colormap='Viridis',
                                     line_color='gray', line_width=1,
                                     fontsize_legend=8,
                                     legend="top_left",
                                     title='Drug Rating and Vader Compound Polarity Sco
                                     size="drugName", alpha=.5)

bkPlot_LL.grid.grid_line_color = None
bkPlot_LL.axis.minor_tick_line_color = None
```

```
C:\Users\Jose\anaconda3\lib\site-packages\pandas_bokeh\plot.py:1332: UserWarning: There
are more than 5 categories in the scatterplot. The legend might be crowded, to hide the
axis you can pass 'legend=False' as an optional argument.
  warnings.warn(
```





In [89]:

```
STATS2_DF = drug_efficacy_df.groupby('drugName', as_index=False).agg({"rating":"mean","

STATS2_DF.reset_index()

STATS2_DF.reset_index(drop=True, inplace=True)

# Sort with respect to most medicines treating a condition
STATS2_DF = STATS2_DF.sort_values(by=["topic_0_compound_polarity_score"], ascending=Fal

STATS2_DF.head()
```

Out[89]:

	drugName	rating	vader_compound_polarity_score	Grade	topic_0_compound_polarity_score
882	Cymbalta	6.728515	-0.253364	11.103008	11
1110	Duloxetine	6.733686	-0.270087	11.077636	11
3455	Venlafaxine	6.868043	-0.244189	11.234863	11
1144	Effexor XR	7.454979	-0.217033	10.852451	10
556	Bupropion	7.462045	0.108191	11.091517	10

In [90]:

```
# Confirm that the previous table is correct
STATS2_DFo = pd.pivot_table(drug_efficacy_df,index=["drugName"],values=["rating"],aggfu
STATS2_DFo = STATS2_DFo.sort_values(by=[( 'len', 'rating')], ascending=False)
STATS2_DFo[0:15]
```

Out[90]:

	mean	len
	rating	rating
drugName		
Cymbalta	6.728515	11.0
Duloxetine	6.733686	11.0
Venlafaxine	6.868043	11.0
Effexor XR	7.454979	10.0
Bupropion	7.462045	10.0
Amitriptyline	7.869153	10.0
Pristiq	7.671346	9.0
Gabapentin	7.796709	9.0
Tramadol	7.917723	9.0
Desvenlafaxine	7.722200	9.0

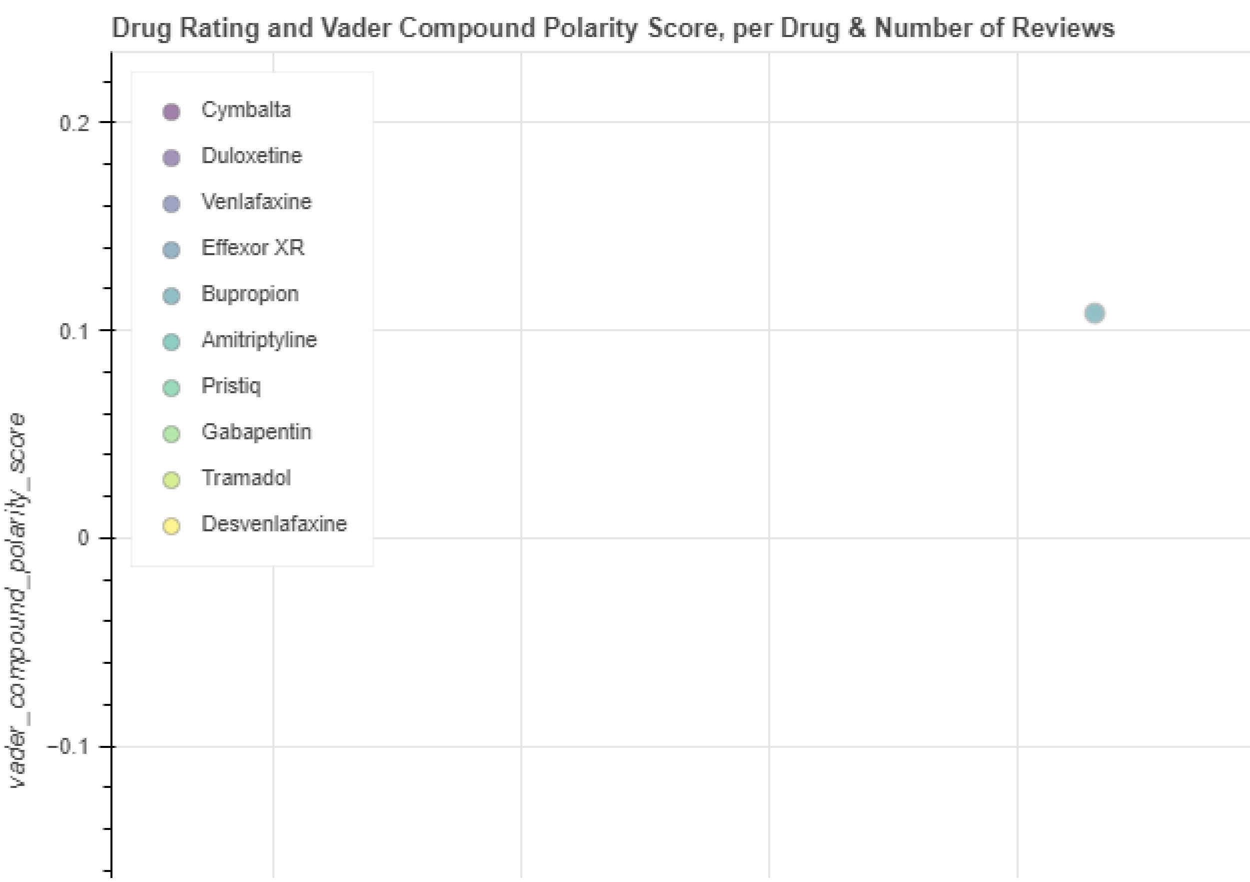
	mean rating	len rating
drugName		
Escitalopram	7.847241	8.0
Wellbutrin	6.869565	8.0
Neurontin	7.728018	8.0
Lexapro	8.175954	8.0
Clonidine	7.620083	8.0

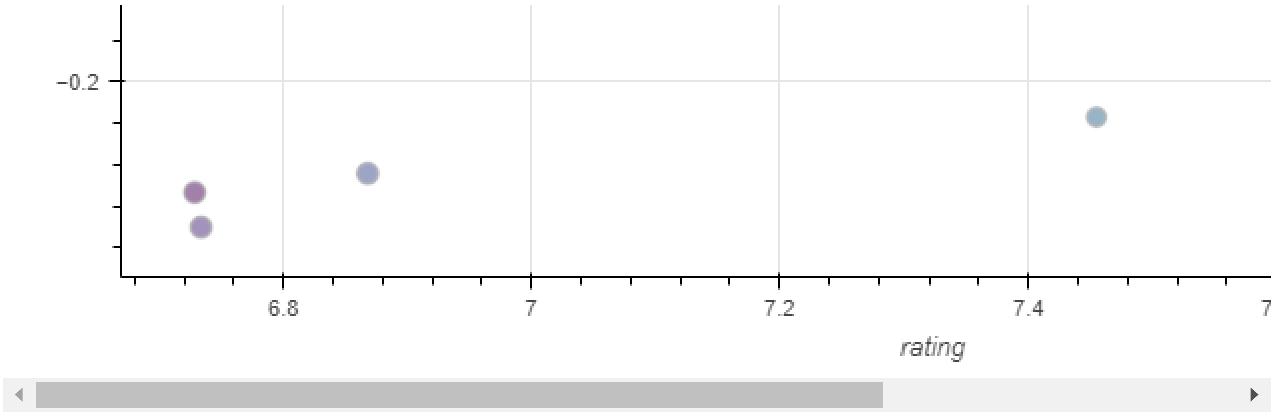
In [91]:

```
bkPlot_LL = STATS2_DF[:10].plot_bokeh.scatter('rating','vader_compound_polarity_score',
                                               figsize=(900,620),
                                               category='drugName',
                                               colormap='Viridis',
                                               line_color='gray', line_width=1,
                                               fontsize_legend=8,
                                               legend="top_left",
                                               title='Drug Rating and Vader Compound Polarity Sco
                                               size="topic_0_compound_polarity_score", alpha=.5)

bkPlot_LL.grid.grid_line_color = None
bkPlot_LL.axis.minor_tick_line_color = None
```

```
C:\Users\Jose\anaconda3\lib\site-packages\pandas_bokeh\plot.py:1332: UserWarning: There
are more than 5 categories in the scatterplot. The legend might be crowded, to hide the
axis you can pass 'legend=False' as an optional argument.
  warnings.warn(
```





In [92]:

```
# Prepare X and Y dataframes for use in spending analysis
GRAPHscat= drug_efficacy_df.drop(columns=['Grade'])
GRAPHscat.dropna(inplace=True)

STATS3_DF = GRAPHscat.groupby('drugName', as_index=False).agg({"rating":"mean","vader_c

STATS3_DF.reset_index()

STATS3_DF.reset_index(drop=True, inplace=True)
# Sort with respect to most medicines treating a condition
STATS3_DF = STATS3_DF.sort_values(by=["Spending"], ascending=False)

STATS3_DF.head()
```

Out[92]:

	drugName	rating	vader_compound_polarity_score	Spending
324	Cymbalta	6.728515	-0.253364	11
400	Effexor XR	7.454979	-0.217033	10
986	Pristiq	7.671346	0.209979	9
519	Gabapentin	7.796709	-0.062325	9
680	Lexapro	8.175954	0.101497	8

In [93]:

```
import math
```

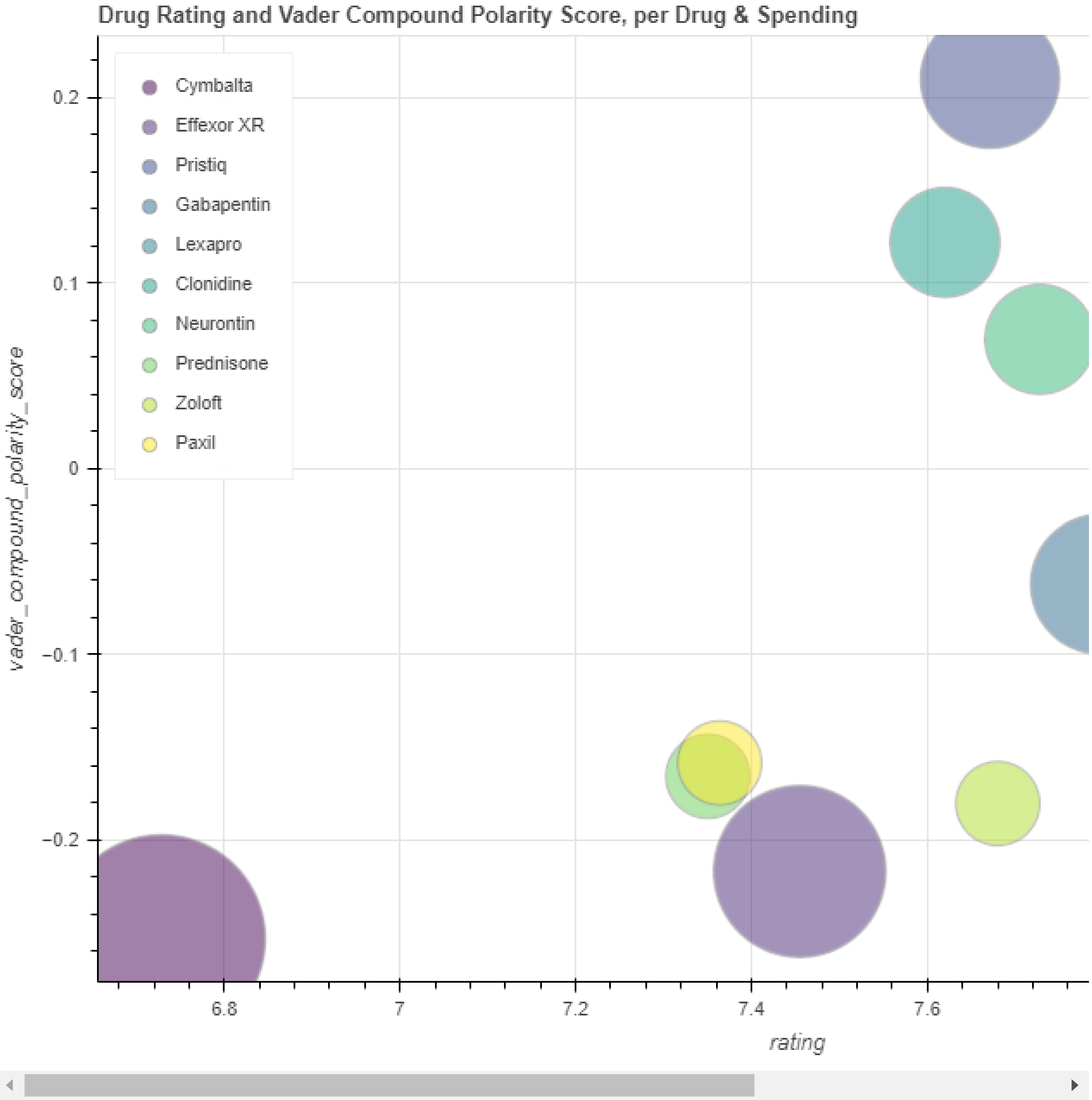
In [94]:

```
STATS3_DF["Spending10"] = (STATS3_DF["Spending"]**2 )

bkPlot_HH = STATS3_DF[:10].plot_bokeh.scatter('rating','vader_compound_polarity_score',
                                     figsize=(900,620),
                                     category='drugName',
                                     colormap='Viridis',
                                     line_color='gray', line_width=1,
                                     fontsize_legend=8,
                                     legend="top_left",
                                     title='Drug Rating and Vader Compound Polarity Sco
                                     size="Spending10", alpha=.5)

bkPlot_HH.grid.grid_line_color = None
bkPlot_HH.axis.minor_tick_line_color = None
```

```
C:\Users\Jose\anaconda3\lib\site-packages\pandas_bokeh\plot.py:1332: UserWarning: There are more than 5 categories in the scatterplot. The legend might be crowded, to hide the axis you can pass 'legend=False' as an optional argument.
  warnings.warn(
```



In [95]:

```
# Derive drug efficacy binary classification
drug_efficacy_df['effective_drug']=drug_efficacy_df['rating'].apply(lambda rating: rati
```

In [96]:

```
# Verify output is as expected
drug_efficacy_df.head(10)
```

Out[96]:

	drugName	condition	rating	Grade	Spending	vader_compound_polarity_score	topic_0_con
0	A + D Cracked Skin Relief	other	10.000000	24.700000	NaN	0.178800	
1	A / B Otic	other	10.000000	3.300000	NaN	-0.015600	

	drugName	condition	rating	Grade	Spending	vader_compound_polarity_score	topic_0_con
2	Abacavir / dolutegravir / lamivudine	hiv infection	8.414286	11.510000	NaN	0.206616	
3	Abacavir / lamivudine	hiv infection	10.000000	12.033333	26.98	0.336167	
4	Abacavir / lamivudine / zidovudine	hiv infection	9.000000	3.900000	21.53	-0.077200	
5	Abatacept	other	7.000000	7.950000	NaN	0.211650	
6	Abatacept	rheumatoid arthritis	7.000000	10.291304	NaN	0.090513	
7	Abilify	bipolar disorde	5.854271	10.662814	32.44	-0.041674	
8	Abilify	depression	6.845361	11.409278	32.44	0.034465	
9	Abilify	major depressive disorde	5.943396	12.877358	32.44	0.086066	

In [97]:

```
# Get dummies for each condition
drug_efficacy_df = pd.get_dummies(data=drug_efficacy_df, drop_first = False, columns=['
```

In [98]:

```
drug_efficacy_df = drug_efficacy_df.drop('condition_other', 1)
```

In [99]:

```
drug_efficacy_df.columns
```

Out[99]:

```
Index(['drugName', 'rating', 'Grade', 'Spending',
       'vader_compound_polarity_score', 'topic_0_compound_polarity_score',
       'topic_1_compound_polarity_score', 'topic_2_compound_polarity_score',
       'topic_3_compound_polarity_score', 'topic_4_compound_polarity_score',
       'effective_drug', 'condition_abnormal uterine bleeding',
       'condition_acne', 'condition_adhd', 'condition_allergic rhinitis',
       'condition_anxiety', 'condition_anxiety and stress',
       'condition_back pain', 'condition_bacterial infection',
       'condition_bipolar disorde', 'condition_birth control',
       'condition_bowel preparation', 'condition_chronic pain',
       'condition_constipation', 'condition_cough', 'condition_depression',
       'condition_diabetes, type 2', 'condition_emergency contraception',
       'condition_erectile dysfunction',
       'condition_generalized anxiety disorde', 'condition_gerd',
       'condition_hepatitis c', 'condition_high blood pressure',
       'condition_high cholesterol', 'condition_hiv infection',
       'condition_hyperhidrosis', 'condition_ibromyalgia',
       'condition_insomnia', 'condition_irritable bowel syndrome',
       'condition_major depressive disorde', 'condition_migraine',
       'condition_migraine prevention', 'condition_multiple sclerosis',
       'condition_muscle spasm', 'condition_nausea/vomiting',
       'condition_obesity', 'condition_opiate dependence',
```

```
       'condition_osteoarthritis', 'condition_overactive bladde',
       'condition_pain', 'condition_panic disorde', 'condition_psoriasis',
       'condition_restless legs syndrome', 'condition_rheumatoid arthritis',
       'condition_schizophrenia', 'condition_sinusitis',
       'condition_smoking cessation', 'condition_urinary tract infection',
       'condition_vaginal yeast infection', 'condition_weight loss'],
      dtype='object')
```

In [100…

```python
# Verify output is as expected
drug_efficacy_df.head(5)
```

Out[100…

	drugName	rating	Grade	Spending	vader_compound_polarity_score	topic_0_compound_polar
0	A + D Cracked Skin Relief	10.000000	24.700000	NaN	0.178800	
1	A / B Otic	10.000000	3.300000	NaN	-0.015600	
2	Abacavir / dolutegravir / lamivudine	8.414286	11.510000	NaN	0.206616	
3	Abacavir / lamivudine	10.000000	12.033333	26.98	0.336167	
4	Abacavir / lamivudine / zidovudine	9.000000	3.900000	21.53	-0.077200	

5 rows × 60 columns

In [101…

```python
del drug_efficacy_df["vader_compound_polarity_score"]
drug_efficacy_df.head(5)
```

Out[101…

	drugName	rating	Grade	Spending	topic_0_compound_polarity_score	topic_1_compound_pol
0	A + D Cracked Skin Relief	10.000000	24.700000	NaN	0.173436	
1	A / B Otic	10.000000	3.300000	NaN	0.028961	
2	Abacavir / dolutegravir / lamivudine	8.414286	11.510000	NaN	0.020531	
3	Abacavir / lamivudine	10.000000	12.033333	26.98	0.006852	
4	Abacavir / lamivudine / zidovudine	9.000000	3.900000	21.53	-0.002316	

5 rows × 59 columns

Prepare Train and Test Datasets

In [102…

```
# Prepare X and y dataframes
X= drug_efficacy_df.drop(columns=['drugName','rating','effective_drug','Spending'])
y= drug_efficacy_df['effective_drug']
```

In [103…

```
# Prepare train_test_split
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.3, random_state
```

In [104…

```
# Prepare X and y dataframes for spending analysis
X_price= drug_efficacy_df.drop(columns=['drugName','rating','effective_drug'])
y_price= drug_efficacy_df['effective_drug']
y_price_ols= drug_efficacy_df['rating']

Xy = copy.deepcopy(X_price)
Xy["y"] = y_price
Xy["y_ols"] = y_price_ols

Xy.dropna(inplace=True)

y_price = Xy["y"]
y_price_ols = Xy["y_ols"]

del Xy["y"]
del Xy["y_ols"]
X_price=Xy

X_price['Spending'] = np.log(X_price['Spending'])
```

In [105…

```
y_price
```

Out[105…

```
3        True
4        True
7       False
8       False
9       False
        ...
5703     True
5704     True
5707     True
5708    False
5709     True
Name: y, Length: 2322, dtype: bool
```

In [106…

```
X_price
```

Out[106…

	Grade	Spending	topic_0_compound_polarity_score	topic_1_compound_polarity_score	topic_2_c

	Grade	Spending	topic_0_compound_polarity_score	topic_1_compound_polarity_score	topic_2_c
3	12.033333	3.295096	0.006852	0.308813	
4	3.900000	3.069447	-0.002316	-0.069480	
7	10.662814	3.479392	-0.001603	-0.005119	
8	11.409278	3.479392	0.003226	-0.002387	
9	12.877358	3.479392	-0.008838	0.001501	
...	...	...	...	...	
5703	23.600000	4.340749	-0.051981	-0.008679	
5704	8.538095	4.340749	0.015160	-0.014524	
5707	8.711268	3.607127	-0.032656	-0.010046	
5708	6.150000	1.465568	-0.006391	-0.139711	
5709	15.500000	1.465568	0.003606	-0.048854	

2322 rows × 56 columns

OLS Regression

In [152…

```
y_ols = drug_efficacy_df['rating']

# Prepare train_test_split
train_Xols, valid_Xols, train_yols, valid_yols = train_test_split(X, y_ols, test_size=0
```

In [158…

```
efficacy_ols = LinearRegression()
efficacy_ols.fit(train_Xols,train_yols)

OLS_PD = pd.DataFrame({'Predictor':X.columns,'coefficient':efficacy_ols.coef_})

OLS_PD["coeff_abs"] = abs(OLS_PD['coefficient'])

OLS_PD.sort_values(["coeff_abs"], ascending=[False], inplace = True)

print(OLS_PD)

regressionSummary(train_yols,efficacy_ols.predict(train_Xols))
```

```
                              Predictor  coefficient  coeff_abs
4          topic_3_compound_polarity_score     3.333795   3.333795
1          topic_0_compound_polarity_score     2.723465   2.723465
3          topic_2_compound_polarity_score     2.488326   2.488326
5          topic_4_compound_polarity_score     2.033608   2.033608
16             condition_bowel preparation    -2.029432   2.029432
43             condition_overactive bladde    -1.869403   1.869403
6     condition_abnormal uterine bleeding    -1.796651   1.796651
45                condition_panic disorde     1.781147   1.781147
15                condition_birth control    -1.678020   1.678020
```

```
2        topic_1_compound_polarity_score     1.620220   1.620220
53      condition_vaginal yeast infection    -1.025668   1.025668
22      condition_emergency contraception     0.930673   0.930673
11           condition_anxiety and stress     0.877200   0.877200
12                    condition_back pain     0.852163   0.852163
21            condition_diabetes, type 2     -0.713716   0.713716
13          condition_bacterial infection    -0.691198   0.691198
44                         condition_pain     0.644486   0.644486
46                    condition_psoriasis     0.587431   0.587431
24  condition_generalized anxiety disorde     0.583354   0.583354
8                          condition_adhd    -0.570041   0.570041
20                   condition_depression     0.566259   0.566259
54                  condition_weight loss     0.517314   0.517314
10                      condition_anxiety     0.513715   0.513715
28            condition_high cholesterol     -0.507025   0.507025
25                         condition_gerd    -0.503828   0.503828
7                          condition_acne    -0.503010   0.503010
41           condition_opiate dependence      0.487720   0.487720
33     condition_irritable bowel syndrome     0.464469   0.464469
36          condition_migraine prevention     0.462187   0.462187
35                     condition_migraine     0.437094   0.437094
40                      condition_obesity     0.428457   0.428457
18                 condition_constipation     0.426043   0.426043
17                 condition_chronic pain     0.411467   0.411467
27          condition_high blood pressure    -0.340707   0.340707
29                condition_hiv infection    -0.306958   0.306958
14              condition_bipolar disorde    -0.280057   0.280057
48         condition_rheumatoid arthritis     0.278486   0.278486
38                 condition_muscle spasm     0.276913   0.276913
52      condition_urinary tract infection    -0.260802   0.260802
51           condition_smoking cessation     -0.237591   0.237591
49                condition_schizophrenia    -0.223402   0.223402
42              condition_osteoarthritis      0.202027   0.202027
37           condition_multiple sclerosis     0.199335   0.199335
39              condition_nausea/vomiting     0.193168   0.193168
31                  condition_ibromyalgia     0.185955   0.185955
9            condition_allergic rhinitis      0.139610   0.139610
23         condition_erectile dysfunction    -0.109046   0.109046
30                condition_hyperhidrosis     0.097014   0.097014
34     condition_major depressive disorde    -0.075655   0.075655
32                     condition_insomnia     0.070445   0.070445
26                  condition_hepatitis c     0.067810   0.067810
47       condition_restless legs syndrome    -0.033844   0.033844
0                                   Grade    -0.020372   0.020372
19                        condition_cough     0.018236   0.018236
50                    condition_sinusitis    -0.008988   0.008988

Regression statistics

                      Mean Error (ME) : -0.0000
       Root Mean Squared Error (RMSE) : 1.8716
            Mean Absolute Error (MAE) : 1.3994
          Mean Percentage Error (MPE) : -19.4793
Mean Absolute Percentage Error (MAPE) : 34.8552
```

Build and Optimize Efficacy Models

a. Decision Tree Optimization

Parameters chosen for optimization are maximum depth and criterion

In [109…

```
# Decision Tree with hyperparameter optimization
param_grid = {
    'max_depth': [number for number in range(1,16)],
    'criterion':['gini', 'entropy'],
}

grid_search_dt = GridSearchCV(DecisionTreeClassifier(random_state=1),param_grid,scoring
grid_search_dt.fit(train_X, train_y)
efficacy_dt = grid_search_dt.best_estimator_
print(grid_search_dt.best_params_)
```

```
{'criterion': 'entropy', 'max_depth': 10}
```

Optimized decision tree scoring

In [110…

```
# Score the optimized Decision Tree
prediction_dt_valid = efficacy_dt.predict(valid_X)
print(accuracy_score(valid_y, prediction_dt_valid))
```

```
0.6878646441073513
```

In [111…

```
# Score the optimized Decision Tree

prediction_dt_train = efficacy_dt.predict(train_X)

opt_dt_score = ca_score_model(train_y, prediction_dt_train, valid_y, prediction_dt_vali
```

```
Training Set Metrics:
Accuracy on the train is: 0.7995996997748311
Confusion Matrix (Accuracy 0.7996)

       Prediction
Actual    0    1
     0 1094  489
     1  312 2102
The Precision on the train is: 0.811269780007719
The Recall on the train is: 0.8707539353769677
The F-Measure on the train is: 0.83996003996004

Testing Set Metrics:
Accuracy on the test is: 0.6878646441073513
Confusion Matrix (Accuracy 0.6879)

       Prediction
Actual   0   1
     0 334 280
     1 255 845
The Precision on the test is: 0.7511111111111111
The Recall on the test is: 0.7681818181818182
The F-Measure on the test is: 0.7595505617977527
```

b. Random forest Optimization

Parameters chosen for optimization are number of estimators, criterion and bootstrap

In [112…

```
# Random Forest with hyperparameter optimization
```

```
param_grid = {
    'n_estimators': [100,200,300],
    'criterion':['gini', 'entropy'],
    'bootstrap': [True, False],
}

grid_search_forest = GridSearchCV(RandomForestClassifier(random_state=1),param_grid,sco
grid_search_forest.fit(train_X, train_y)
efficacy_forest = grid_search_forest.best_estimator_
print(grid_search_forest.best_params_)
```

```
{'bootstrap': True, 'criterion': 'entropy', 'n_estimators': 100}
```

Optimized random forest scoring

In [113…

```
# Score the optimized Random Forest
prediction_forest_valid = efficacy_forest.predict(valid_X)
print(accuracy_score(valid_y, prediction_forest_valid))
```

```
0.7491248541423571
```

In [114…

```
# Score the Random Forest

prediction_forest_train = efficacy_forest.predict(train_X)

opt_forest_score = ca_score_model(train_y, prediction_forest_train, valid_y, prediction
```

```
Training Set Metrics:
Accuracy on the train is: 0.9984988741556167
Confusion Matrix (Accuracy 0.9985)

       Prediction
Actual    0    1
     0 1581    2
     1    4 2410
The Precision on the train is: 0.9991708126036484
The Recall on the train is: 0.9983429991714996
The F-Measure on the train is: 0.9987567343555739

Testing Set Metrics:
Accuracy on the test is: 0.7491248541423571
Confusion Matrix (Accuracy 0.7491)

       Prediction
Actual   0   1
     0 378 236
     1 194 906
The Precision on the test is: 0.7933450087565674
The Recall on the test is: 0.8236363636363636
The F-Measure on the test is: 0.8082069580731489
```

c. Boosted trees Optimization

Parameters chosen for optimization are number of estimators and loss

In [115…

```
# Boosted Tree with hyperparameter optimization
param_grid = {
```

```
    'loss':['deviance', 'exponential'],
    'n_estimators': [100,200,300]
}

grid_search_boost = GridSearchCV(GradientBoostingClassifier(random_state=1),param_grid,
grid_search_boost.fit(train_X, train_y)
efficacy_boost = grid_search_boost.best_estimator_
print(grid_search_boost.best_params_)
```

```
{'loss': 'exponential', 'n_estimators': 300}
```

Optimized gradient boosted tree scoring

In [116…

```
# Score the optimized Boosted Tree
prediction_boost_valid = efficacy_boost.predict(valid_X)
print(accuracy_score(valid_y, prediction_boost_valid))
```

```
0.7187864644107351
```

In [117…

```
prediction_boost_train = efficacy_boost.predict(train_X)

opt_boost_score = ca_score_model(train_y, prediction_boost_train, valid_y, prediction_b
```

```
Training Set Metrics:
Accuracy on the train is: 0.8128596447335502
Confusion Matrix (Accuracy 0.8129)

       Prediction
Actual    0    1
     0 1098  485
     1  263 2151
The Precision on the train is: 0.8160091047040972
The Recall on the train is: 0.8910521955260977
The F-Measure on the train is: 0.8518811881188119

Testing Set Metrics:
Accuracy on the test is: 0.7187864644107351
Confusion Matrix (Accuracy 0.7188)

       Prediction
Actual   0   1
     0 352 262
     1 220 880
The Precision on the test is: 0.7705779334500875
The Recall on the test is: 0.8
The F-Measure on the test is: 0.7850133809099019
```

Combine Optimized Eficacy Models into Ensembles

We build an ensemble of classification trees, random forests and boosted trees using the optimized parameters determined in the previous steps.

Ensemble - Stacking

In [118…
```
# Populate the estimators based on the optimized standalone models
estimators = [
('gbm', GradientBoostingClassifier(loss= 'exponential', n_estimators= 300, random_state
('rf', RandomForestClassifier(bootstrap= True, criterion= 'entropy', n_estimators= 100,
('dt', DecisionTreeClassifier(criterion= 'entropy', max_depth= 10, random_state = 1))
]
```

In [119…
```
# Prepare the Stacking Classifier
efficacy_stack = StackingClassifier(
estimators=estimators, final_estimator=LogisticRegression(penalty= 'l2', solver= 'newto
)

efficacy_stack.fit(train_X, train_y)
```

Out[119…
```
StackingClassifier(estimators=[('gbm',
                                GradientBoostingClassifier(loss='exponential',
                                                           n_estimators=300,
                                                           random_state=1)),
                               ('rf',
                                RandomForestClassifier(criterion='entropy',
                                                       random_state=1)),
                               ('dt',
                                DecisionTreeClassifier(criterion='entropy',
                                                       max_depth=10,
                                                       random_state=1))],
                   final_estimator=LogisticRegression(C=1e+42, random_state=1,
                                                      solver='newton-cg'))
```

In [120…
```
# Score the Stacking Classifier
prediction_stack_valid = efficacy_stack.predict(valid_X)
print(accuracy_score(valid_y, prediction_stack_valid))
```

```
0.749708284714119
```

In [121…
```
prediction_stack_train = efficacy_stack.predict(train_X)

stack_score = ca_score_model(train_y, prediction_stack_train, valid_y, prediction_stack
```

```
Training Set Metrics:
Accuracy on the train is: 0.9984988741556167
Confusion Matrix (Accuracy 0.9985)

       Prediction
Actual    0    1
     0 1579    4
     1    2 2412
The Precision on the train is: 0.9983443708609272
The Recall on the train is: 0.9991714995857498
The F-Measure on the train is: 0.9987577639751554

Testing Set Metrics:
Accuracy on the test is: 0.749708284714119
Confusion Matrix (Accuracy 0.7497)

       Prediction
Actual   0   1
     0 383 231
     1 198 902
The Precision on the test is: 0.7961165048543689
```

```
The Recall on the test is: 0.82
The F-Measure on the test is: 0.8078817733990147
```

Ensemble - Voting

Build a voting ensemble of random forest, boosted tree, decision tree using the optimized parameters determined in previous steps

In [122…

```
# Prepare the Voting Classifier
efficacy_vote = VotingClassifier(estimators = estimators)
efficacy_vote.fit(train_X, train_y)
```

Out[122…

```
VotingClassifier(estimators=[('gbm',
                              GradientBoostingClassifier(loss='exponential',
                                                         n_estimators=300,
                                                         random_state=1)),
                             ('rf',
                              RandomForestClassifier(criterion='entropy',
                                                     random_state=1)),
                             ('dt',
                              DecisionTreeClassifier(criterion='entropy',
                                                     max_depth=10,
                                                     random_state=1))])
```

In [123…

```
# Score the Voting Classifier
prediction_vote_valid = efficacy_vote.predict(valid_X)
print(accuracy_score(valid_y, prediction_vote_valid))
```

```
0.7333722287047841
```

In [124…

```
# Score the Voting Classifier
prediction_vote_train = efficacy_vote.predict(train_X)
prediction_vote_valid = efficacy_vote.predict(valid_X)

vote_score = ca_score_model(train_y, prediction_vote_train, valid_y, prediction_vote_va
```

```
Training Set Metrics:
Accuracy on the train is: 0.8836627470602952
Confusion Matrix (Accuracy 0.8837)

       Prediction
Actual    0    1
     0 1266  317
     1  148 2266
The Precision on the train is: 0.8772744870305846
The Recall on the train is: 0.9386909693454847
The F-Measure on the train is: 0.9069441664999001

Testing Set Metrics:
Accuracy on the test is: 0.7333722287047841
Confusion Matrix (Accuracy 0.7334)

       Prediction
Actual   0   1
     0 358 256
     1 201 899
The Precision on the test is: 0.7783549783549784
The Recall on the test is: 0.8172727272727273
The F-Measure on the test is: 0.7973392461197341
```

In [125…

```
# Compare accuracy, precision, recall, and F-Measure for Ensemble Stacking and Voting C

model_comp(stack_score,vote_score,'Ensemble Stacking:','Voting Classifier:')
```

```
Model Comparison            Accuracy      Precision     Recall        F-Measure
Ensemble Stacking:          0.7497        0.7961        0.8200        0.8079
Voting Classifier:          0.7334        0.7784        0.8173        0.7973
```

Based on our target for maximum precision the VotingClassifier gives us the optimal result

We will extract the importances of our features from our best performing model which is the stacking classifier and use them to provide insights as to the relation of our features and their significance in determining customer satisfaction

In [126…

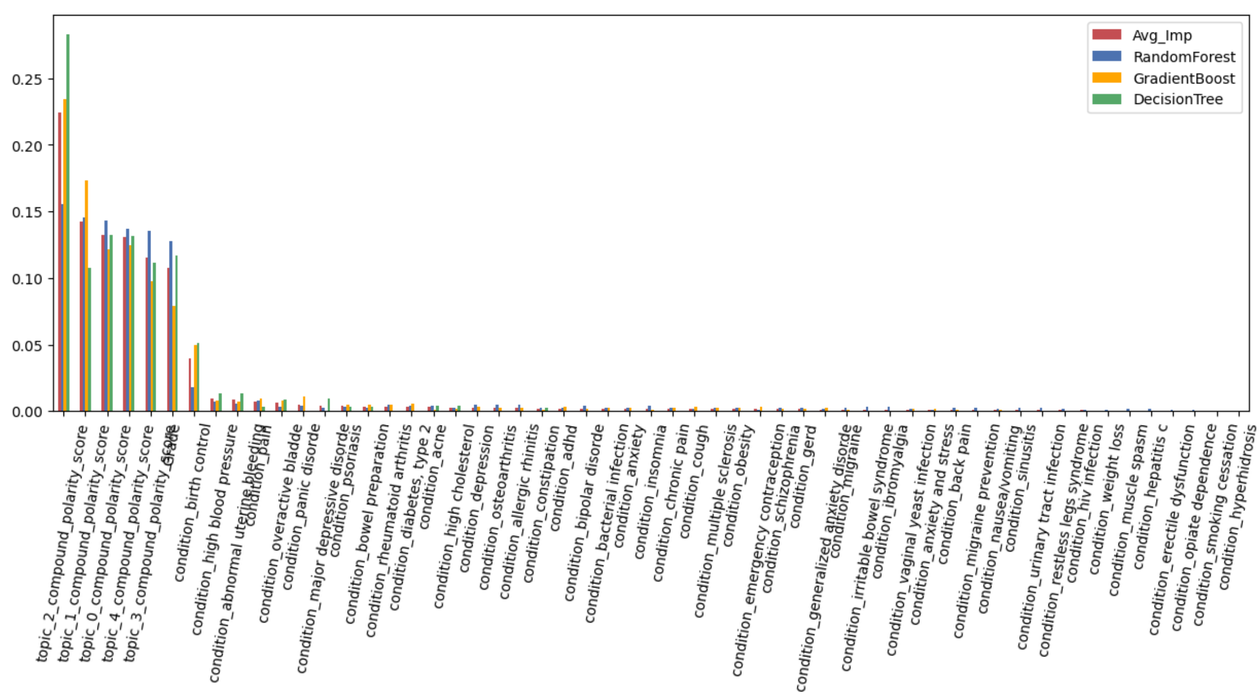
```
voterf =efficacy_stack.named_estimators_['rf'].feature_importances_
votegbm =efficacy_stack.named_estimators_['gbm'].feature_importances_
votedt =efficacy_stack.named_estimators_['dt'].feature_importances_
votes=list(zip(voterf,votegbm,votedt))
```

In [127…

```
voimp = dict(zip(train_X.columns,votes))
label=['RandomForest','GradientBoost','DecisionTree']
final=pd.DataFrame(voimp.values(), index=voimp.keys(),columns=label)
final['Avg_Imp'] = final.mean(axis=1)
```

In [128…

```
poslabel=['Avg_Imp','RandomForest','GradientBoost','DecisionTree']
final.sort_values(by =['Avg_Imp'], ascending = False).plot(y=poslabel,kind = 'bar',colo
```



In [129…

```
final
```

Out[129…

	RandomForest	GradientBoost	DecisionTree	Avg_Imp
Grade	0.127751	0.078737	0.117174	0.107887
topic_0_compound_polarity_score	0.143387	0.121385	0.132401	0.132391
topic_1_compound_polarity_score	0.145412	0.173302	0.107600	0.142105
topic_2_compound_polarity_score	0.155565	0.234097	0.283219	0.224294
topic_3_compound_polarity_score	0.135755	0.097923	0.111608	0.115095
topic_4_compound_polarity_score	0.136825	0.124451	0.131583	0.130953
condition_abnormal uterine bleeding	0.005429	0.007189	0.013337	0.008651
condition_acne	0.004267	0.001109	0.003944	0.003107
condition_adhd	0.002927	0.003375	0.000000	0.002101
condition_allergic rhinitis	0.004827	0.002277	0.000000	0.002368
condition_anxiety	0.002845	0.002586	0.000000	0.001810
condition_anxiety and stress	0.000968	0.002182	0.000000	0.001050
condition_back pain	0.002371	0.000654	0.000000	0.001008
condition_bacterial infection	0.002701	0.002753	0.000000	0.001818
condition_bipolar disorde	0.003890	0.002160	0.000000	0.002016
condition_birth control	0.018405	0.049694	0.051189	0.039763
condition_bowel preparation	0.002422	0.005044	0.003083	0.003516
condition_chronic pain	0.002437	0.002820	0.000000	0.001752
condition_constipation	0.002387	0.001111	0.002850	0.002116
condition_cough	0.001502	0.003651	0.000000	0.001718
condition_depression	0.004654	0.003282	0.000000	0.002645
condition_diabetes, type 2	0.004232	0.005730	0.000000	0.003321
condition_emergency contraception	0.001309	0.003685	0.000000	0.001665
condition_erectile dysfunction	0.000768	0.000339	0.000000	0.000369
condition_generalized anxiety disorde	0.001523	0.002543	0.000000	0.001355
condition_gerd	0.002848	0.001514	0.000000	0.001454
condition_hepatitis c	0.001630	0.000000	0.000000	0.000543
condition_high blood pressure	0.007491	0.007665	0.013442	0.009533
condition_high cholesterol	0.002328	0.002137	0.004334	0.002933
condition_hiv infection	0.001394	0.000621	0.000000	0.000672
condition_hyperhidrosis	0.000459	0.000000	0.000000	0.000153
condition_ibromyalgia	0.003306	0.000000	0.000000	0.001102
condition_insomnia	0.004102	0.001228	0.000000	0.001777

	RandomForest	GradientBoost	DecisionTree	Avg_Imp
condition_irritable bowel syndrome	0.003029	0.000579	0.000000	0.001203
condition_major depressive disorde	0.002712	0.000151	0.009223	0.004028
condition_migraine	0.002687	0.001071	0.000000	0.001252
condition_migraine prevention	0.002444	0.000405	0.000000	0.000950
condition_multiple sclerosis	0.002338	0.002778	0.000000	0.001705
condition_muscle spasm	0.001699	0.000000	0.000000	0.000566
condition_nausea/vomiting	0.002030	0.000699	0.000000	0.000910
condition_obesity	0.002524	0.002503	0.000000	0.001676
condition_opiate dependence	0.000697	0.000000	0.000000	0.000232
condition_osteoarthritis	0.005072	0.002809	0.000000	0.002627
condition_overactive bladde	0.003219	0.007813	0.008448	0.006493
condition_pain	0.008195	0.009816	0.003369	0.007127
condition_panic disorde	0.004052	0.010847	0.000000	0.004966
condition_psoriasis	0.003530	0.005126	0.003194	0.003950
condition_restless legs syndrome	0.001695	0.000584	0.000000	0.000760
condition_rheumatoid arthritis	0.005102	0.005137	0.000000	0.003413
condition_schizophrenia	0.002465	0.002094	0.000000	0.001520
condition_sinusitis	0.002386	0.000269	0.000000	0.000885
condition_smoking cessation	0.000471	0.000000	0.000000	0.000157
condition_urinary tract infection	0.002450	0.000000	0.000000	0.000817
condition_vaginal yeast infection	0.001775	0.001475	0.000000	0.001083
condition_weight loss	0.001309	0.000598	0.000000	0.000636

In [130…

```
as_list = final.index.tolist()
as_list

idx0 = as_list.index('topic_0_compound_polarity_score')
as_list[idx0] = 'Skin Health score'

idx1 = as_list.index('topic_1_compound_polarity_score')
as_list[idx1] = 'Pain score'

idx2 = as_list.index('topic_2_compound_polarity_score')
as_list[idx2] = 'Gastro Health score'

idx3 = as_list.index('topic_3_compound_polarity_score')
as_list[idx3] = 'Menstrual Health score'

idx4 = as_list.index('topic_4_compound_polarity_score')
as_list[idx4] = 'Mental Health score'
```

```
final.index = as_list
```

In [131…

```
final
```

Out[131…

	RandomForest	GradientBoost	DecisionTree	Avg_Imp
Grade	0.127751	0.078737	0.117174	0.107887
Skin Health score	0.143387	0.121385	0.132401	0.132391
Pain score	0.145412	0.173302	0.107600	0.142105
Gastro Health score	0.155565	0.234097	0.283219	0.224294
Menstrual Health score	0.135755	0.097923	0.111608	0.115095
Mental Health score	0.136825	0.124451	0.131583	0.130953
condition_abnormal uterine bleeding	0.005429	0.007189	0.013337	0.008651
condition_acne	0.004267	0.001109	0.003944	0.003107
condition_adhd	0.002927	0.003375	0.000000	0.002101
condition_allergic rhinitis	0.004827	0.002277	0.000000	0.002368
condition_anxiety	0.002845	0.002586	0.000000	0.001810
condition_anxiety and stress	0.000968	0.002182	0.000000	0.001050
condition_back pain	0.002371	0.000654	0.000000	0.001008
condition_bacterial infection	0.002701	0.002753	0.000000	0.001818
condition_bipolar disorde	0.003890	0.002160	0.000000	0.002016
condition_birth control	0.018405	0.049694	0.051189	0.039763
condition_bowel preparation	0.002422	0.005044	0.003083	0.003516
condition_chronic pain	0.002437	0.002820	0.000000	0.001752
condition_constipation	0.002387	0.001111	0.002850	0.002116
condition_cough	0.001502	0.003651	0.000000	0.001718
condition_depression	0.004654	0.003282	0.000000	0.002645
condition_diabetes, type 2	0.004232	0.005730	0.000000	0.003321
condition_emergency contraception	0.001309	0.003685	0.000000	0.001665
condition_erectile dysfunction	0.000768	0.000339	0.000000	0.000369
condition_generalized anxiety disorde	0.001523	0.002543	0.000000	0.001355
condition_gerd	0.002848	0.001514	0.000000	0.001454
condition_hepatitis c	0.001630	0.000000	0.000000	0.000543
condition_high blood pressure	0.007491	0.007665	0.013442	0.009533
condition_high cholesterol	0.002328	0.002137	0.004334	0.002933
condition_hiv infection	0.001394	0.000621	0.000000	0.000672

	RandomForest	GradientBoost	DecisionTree	Avg_Imp
condition_hyperhidrosis	0.000459	0.000000	0.000000	0.000153
condition_ibromyalgia	0.003306	0.000000	0.000000	0.001102
condition_insomnia	0.004102	0.001228	0.000000	0.001777
condition_irritable bowel syndrome	0.003029	0.000579	0.000000	0.001203
condition_major depressive disorde	0.002712	0.000151	0.009223	0.004028
condition_migraine	0.002687	0.001071	0.000000	0.001252
condition_migraine prevention	0.002444	0.000405	0.000000	0.000950
condition_multiple sclerosis	0.002338	0.002778	0.000000	0.001705
condition_muscle spasm	0.001699	0.000000	0.000000	0.000566
condition_nausea/vomiting	0.002030	0.000699	0.000000	0.000910
condition_obesity	0.002524	0.002503	0.000000	0.001676
condition_opiate dependence	0.000697	0.000000	0.000000	0.000232
condition_osteoarthritis	0.005072	0.002809	0.000000	0.002627
condition_overactive bladde	0.003219	0.007813	0.008448	0.006493
condition_pain	0.008195	0.009816	0.003369	0.007127
condition_panic disorde	0.004052	0.010847	0.000000	0.004966
condition_psoriasis	0.003530	0.005126	0.003194	0.003950
condition_restless legs syndrome	0.001695	0.000584	0.000000	0.000760
condition_rheumatoid arthritis	0.005102	0.005137	0.000000	0.003413
condition_schizophrenia	0.002465	0.002094	0.000000	0.001520
condition_sinusitis	0.002386	0.000269	0.000000	0.000885
condition_smoking cessation	0.000471	0.000000	0.000000	0.000157
condition_urinary tract infection	0.002450	0.000000	0.000000	0.000817
condition_vaginal yeast infection	0.001775	0.001475	0.000000	0.001083
condition_weight loss	0.001309	0.000598	0.000000	0.000636

In [132…

```
final.reset_index()
final.set_index('Avg_Imp')

final = final.sort_values(by=["Avg_Imp"], ascending=False)

final.head()
```

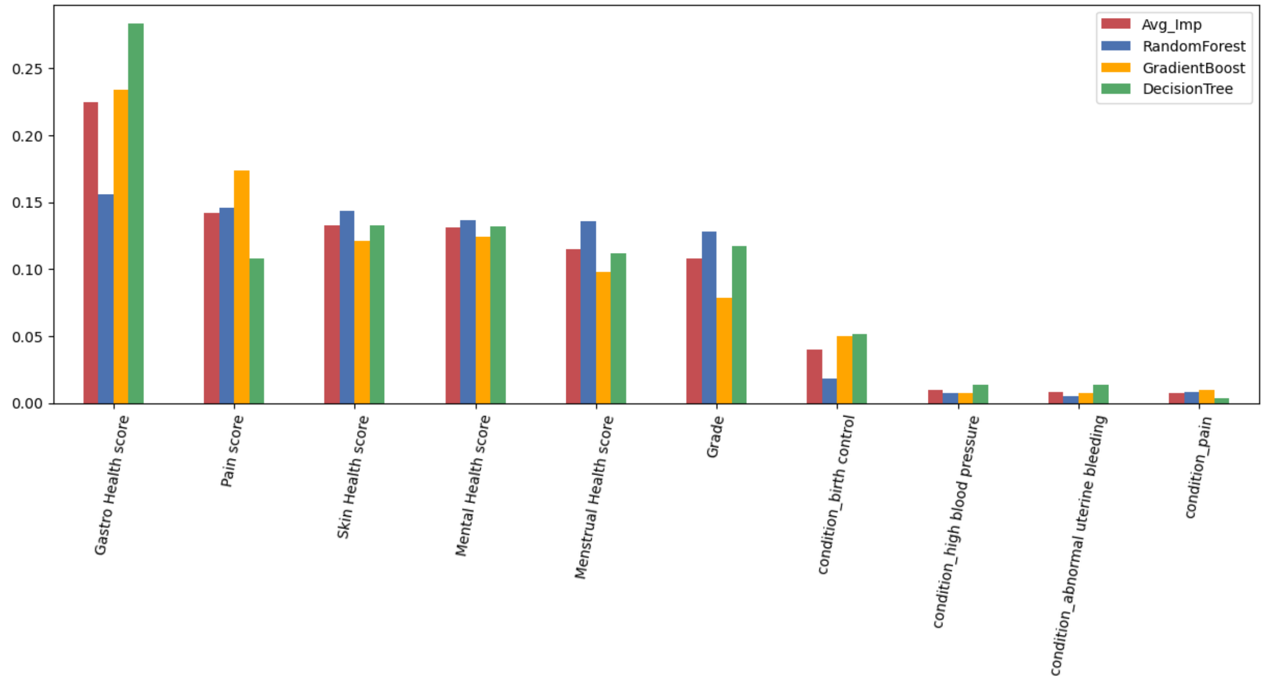
Out[132…

	RandomForest	GradientBoost	DecisionTree	Avg_Imp
Gastro Health score	0.155565	0.234097	0.283219	0.224294
Pain score	0.145412	0.173302	0.107600	0.142105

	RandomForest	GradientBoost	DecisionTree	Avg_Imp
Skin Health score	0.143387	0.121385	0.132401	0.132391
Mental Health score	0.136825	0.124451	0.131583	0.130953
Menstrual Health score	0.135755	0.097923	0.111608	0.115095

In [133…

```
final[:10].sort_values(by =['Avg_Imp'], ascending = False).plot(y=poslabel,kind = 'bar'
```



Redoing OLS regression adding pricing data

In [134…

```
# Prepare train_test_split
train_Xols, valid_Xols, train_yols, valid_yols = train_test_split(X_price, y_price_ols,

efficacy_ols = LinearRegression()
efficacy_ols.fit(train_Xols,train_yols)

print(pd.DataFrame({'Predictor':X_price.columns,'coefficient':efficacy_ols.coef_}))

regressionSummary(train_yols,efficacy_ols.predict(train_Xols))
```

```
                              Predictor   coefficient
0                                 Grade -1.076232e-02
1                              Spending -3.762625e-03
2           topic_0_compound_polarity_score  2.501559e+00
3           topic_1_compound_polarity_score  1.457395e+00
4           topic_2_compound_polarity_score  2.606301e+00
5           topic_3_compound_polarity_score  3.239500e+00
6           topic_4_compound_polarity_score  2.470652e+00
7   condition_abnormal uterine bleeding -2.268665e+00
8                        condition_acne -1.255622e-01
9                        condition_adhd -6.260172e-02
10          condition_allergic rhinitis -7.733016e-01
11                    condition_anxiety  5.705858e-01
```

```
12           condition_anxiety and stress  5.802601e-01
13                    condition_back pain  7.962029e-01
14            condition_bacterial infection -3.322508e-01
15               condition_bipolar disorde -5.563781e-01
16                 condition_birth control -1.583771e+00
17             condition_bowel preparation -1.515596e+00
18                  condition_chronic pain  1.003185e-01
19                  condition_constipation -2.300679e+00
20                          condition_cough  7.457771e-01
21                     condition_depression  5.674599e-02
22            condition_diabetes, type 2 -6.548159e-01
23        condition_emergency contraception -1.687539e-14
24          condition_erectile dysfunction  5.877925e-01
25 condition_generalized anxiety disorde  6.814163e-01
26                           condition_gerd  1.265227e-01
27                    condition_hepatitis c -1.917133e-01
28            condition_high blood pressure -6.347724e-01
29               condition_high cholesterol -5.794083e-01
30                  condition_hiv infection  1.029388e-01
31                  condition_hyperhidrosis -8.421807e-01
32                    condition_ibromyalgia  2.902840e-01
33                       condition_insomnia -2.704278e-01
34       condition_irritable bowel syndrome  1.063547e+00
35       condition_major depressive disorde -3.306296e-01
36                       condition_migraine  8.302331e-01
37            condition_migraine prevention  7.108555e-01
38           condition_multiple sclerosis -1.501268e-01
39                   condition_muscle spasm  5.867349e-01
40              condition_nausea/vomiting -7.597977e-02
41                        condition_obesity  8.969085e-01
42             condition_opiate dependence -2.191358e-01
43                 condition_osteoarthritis  7.911810e-01
44             condition_overactive bladde -3.836780e-01
45                           condition_pain  4.420374e-01
46                  condition_panic disorde  2.035329e+00
47                      condition_psoriasis  5.850209e-01
48         condition_restless legs syndrome  5.310565e-01
49          condition_rheumatoid arthritis  3.643779e-01
50                  condition_schizophrenia -7.345336e-01
51                      condition_sinusitis -5.841739e-01
52             condition_smoking cessation -1.550425e+00
53        condition_urinary tract infection -2.252236e-01
54        condition_vaginal yeast infection -6.532544e-01
55                    condition_weight loss  1.413653e+00

Regression statistics

                      Mean Error (ME) : -0.0000
       Root Mean Squared Error (RMSE) : 1.6594
            Mean Absolute Error (MAE) : 1.2469
          Mean Percentage Error (MPE) : -13.7015
Mean Absolute Percentage Error (MAPE) : 27.7966
```

Redoing Data Mining analysis adding pricing data

In [135…

```
# Prepare train_test_split
train_X, valid_X, train_y, valid_y = train_test_split(X_price, y_price, test_size=0.3,
```

Build and Optimize Efficacy Models

a. Decision Tree Optimization

Parameters chosen for optimization are maximum depth and criterion

In [136…

```
# Decision Tree with hyperparameter optimization
param_grid = {
    'max_depth': [number for number in range(1,16)],
    'criterion':['gini', 'entropy'],
}

grid_search_dt = GridSearchCV(DecisionTreeClassifier(random_state=1),param_grid,scoring
grid_search_dt.fit(train_X, train_y)
efficacy_dt = grid_search_dt.best_estimator_
print(grid_search_dt.best_params_)
print("")

#Score the optimized decision tree
prediction_dt_valid = efficacy_dt.predict(valid_X)
print("Accuracy: ", accuracy_score(valid_y, prediction_dt_valid))
```

```
{'criterion': 'entropy', 'max_depth': 4}

Accuracy:  0.6757532281205165
```

Accuracy did not improve for the Decision tree.

In [137…

```
# Score the optimized Decision Tree

prediction_dt_train = efficacy_dt.predict(train_X)

opt_dt_score = ca_score_model(train_y, prediction_dt_train, valid_y, prediction_dt_vali
```

```
Training Set Metrics:
Accuracy on the train is: 0.7083076923076923
Confusion Matrix (Accuracy 0.7083)

       Prediction
Actual   0   1
     0 403 273
     1 201 748
The Precision on the train is: 0.732615083251714
The Recall on the train is: 0.7881981032665965
The F-Measure on the train is: 0.7593908629441624

Testing Set Metrics:
Accuracy on the test is: 0.6757532281205165
Confusion Matrix (Accuracy 0.6758)

       Prediction
Actual   0   1
     0 157 118
     1 108 314
The Precision on the test is: 0.7268518518518519
The Recall on the test is: 0.7440758293838863
The F-Measure on the test is: 0.7353629976580797
```

b. Random forest Optimization

Parameters chosen for optimization are number of estimators, criterion and bootstrap

In [138…

```
# Random Forest with hyperparameter optimization
param_grid = {
    'n_estimators': [100,200,300],
    'criterion':['gini', 'entropy'],
    'bootstrap': [True, False],
}

grid_search_forest = GridSearchCV(RandomForestClassifier(random_state=1),param_grid,sco
grid_search_forest.fit(train_X, train_y)
efficacy_forest = grid_search_forest.best_estimator_
print(grid_search_forest.best_params_)
print("")

# Score the optimized Random Forest
prediction_forest_valid = efficacy_forest.predict(valid_X)
print("Accuracy: ", accuracy_score(valid_y, prediction_forest_valid))
```

```
{'bootstrap': True, 'criterion': 'entropy', 'n_estimators': 200}

Accuracy:  0.7474892395982783
```

Accuracy did not improve for the Random Forest

In [139…

```
# Score the Random Forest

prediction_forest_train = efficacy_forest.predict(train_X)

opt_forest_score = ca_score_model(train_y, prediction_forest_train, valid_y, prediction
```

```
Training Set Metrics:
Accuracy on the train is: 1.0
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual   0   1
     0 676   0
     1   0 949
The Precision on the train is: 1.0
The Recall on the train is: 1.0
The F-Measure on the train is: 1.0

Testing Set Metrics:
Accuracy on the test is: 0.7474892395982783
Confusion Matrix (Accuracy 0.7475)

       Prediction
Actual   0   1
     0 165 110
     1  66 356
The Precision on the test is: 0.7639484978540773
The Recall on the test is: 0.8436018957345972
The F-Measure on the test is: 0.8018018018018018
```

c. Boosted trees Optimization

Parameters chosen for optimization are number of estimators and loss

In [140…

```
# Boosted Tree with hyperparameter optimization
param_grid = {
    'loss':['deviance', 'exponential'],
    'n_estimators': [100,200,300]
}

grid_search_boost = GridSearchCV(GradientBoostingClassifier(random_state=1),param_grid,
grid_search_boost.fit(train_X, train_y)
efficacy_boost = grid_search_boost.best_estimator_
print(grid_search_boost.best_params_)
```

```
{'loss': 'exponential', 'n_estimators': 100}
```

In [141…

```
# Score the optimized Boosted Tree
prediction_boost_valid = efficacy_boost.predict(valid_X)
print("Accuracy: ", accuracy_score(valid_y, prediction_boost_valid))
```

```
Accuracy:  0.727403156384505
```

Accuracy improved slightly for the Boosted trees (previously: 0.72)

In [142…

```
prediction_boost_train = efficacy_boost.predict(train_X)

opt_boost_score = ca_score_model(train_y, prediction_boost_train, valid_y, prediction_b
```

```
Training Set Metrics:
Accuracy on the train is: 0.8055384615384615
Confusion Matrix (Accuracy 0.8055)

       Prediction
Actual   0   1
     0 460 216
     1 100 849
The Precision on the train is: 0.7971830985915493
The Recall on the train is: 0.8946259220231823
The F-Measure on the train is: 0.8430983118172791

Testing Set Metrics:
Accuracy on the test is: 0.727403156384505
Confusion Matrix (Accuracy 0.7274)

       Prediction
Actual   0   1
     0 161 114
     1  76 346
The Precision on the test is: 0.7521739130434782
The Recall on the test is: 0.8199052132701422
The F-Measure on the test is: 0.7845804988662132
```

Combine Optimized Eficacy Models into Ensembles

Ensemble - Stacking

In [143…

```
# Populate the estimators based on the optimized standalone models
estimators = [
('gbm', GradientBoostingClassifier(loss= 'exponential', n_estimators= 100, random_state
('rf', RandomForestClassifier(bootstrap= True, criterion= 'entropy', n_estimators= 200,
('dt', DecisionTreeClassifier(criterion= 'entropy', max_depth= 4, random_state = 1))
]

# Prepare the Stacking Classifier
efficacy_stack = StackingClassifier(
estimators=estimators, final_estimator=LogisticRegression(penalty= 'l2', solver= 'newto
)

efficacy_stack.fit(train_X, train_y)

# Score the Stacking Classifier
prediction_stack_valid = efficacy_stack.predict(valid_X)
print(accuracy_score(valid_y, prediction_stack_valid))
```

```
0.7517934002869441
```

In [144…

```
# Score the Stacking Classifier
prediction_stack_train = efficacy_stack.predict(train_X)

stack_score = ca_score_model(train_y, prediction_stack_train, valid_y, prediction_stack
```

```
Training Set Metrics:
Accuracy on the train is: 0.9833846153846154
Confusion Matrix (Accuracy 0.9834)

       Prediction
Actual   0   1
     0 651  25
     1   2 947
The Precision on the train is: 0.9742798353909465
The Recall on the train is: 0.9978925184404637
The F-Measure on the train is: 0.9859448204060386

Testing Set Metrics:
Accuracy on the test is: 0.7517934002869441
Confusion Matrix (Accuracy 0.7518)

       Prediction
Actual   0   1
     0 172 103
     1  70 352
The Precision on the test is: 0.7736263736263737
The Recall on the test is: 0.8341232227488151
The F-Measure on the test is: 0.8027366020524515
```

Accuracy did not improve for this new Stacking Ensemble.

Ensemble - Voting

In [145…

```
#Prepare the voting classifier
efficacy_vote = VotingClassifier(estimators = estimators)
```

```
efficacy_vote.fit(train_X, train_y)
#Score the voting classifier
prediction_vote_valid = efficacy_vote.predict(valid_X)
print(accuracy_score(valid_y, prediction_vote_valid))
```

```
0.733142037302726
```

Accuracy did not improve for this new Voting Ensemble.

In [146…

```
# Score the Voting Classifier
prediction_vote_train = efficacy_vote.predict(train_X)
prediction_vote_valid = efficacy_vote.predict(valid_X)

vote_score = ca_score_model(train_y, prediction_vote_train, valid_y, prediction_vote_va
```

```
Training Set Metrics:
Accuracy on the train is: 0.8381538461538461
Confusion Matrix (Accuracy 0.8382)

       Prediction
Actual   0   1
     0 495 181
     1  82 867
The Precision on the train is: 0.8272900763358778
The Recall on the train is: 0.9135932560590094
The F-Measure on the train is: 0.8683024536805207

Testing Set Metrics:
Accuracy on the test is: 0.733142037302726
Confusion Matrix (Accuracy 0.7331)

       Prediction
Actual   0   1
     0 167 108
     1  78 344
The Precision on the test is: 0.7610619469026548
The Recall on the test is: 0.8151658767772512
The F-Measure on the test is: 0.7871853546910756
```

In [147…

```
# Compare accuracy, precision, recall, and F-Measure for Ensemble Stacking and Voting C

model_comp(stack_score,vote_score,'Ensemble Stacking:','Voting Classifier:')
```

```
Model Comparison            Accuracy      Precision     Recall        F-Measure
Ensemble Stacking:          0.7518        0.7736        0.8341        0.8027
Voting Classifier:          0.7331        0.7611        0.8152        0.7872
```

We will extract the importances of our features from our best performing model which is the stacking classifier and use them to provide insights as to the relation of our features and their significance in determining customer satisfaction

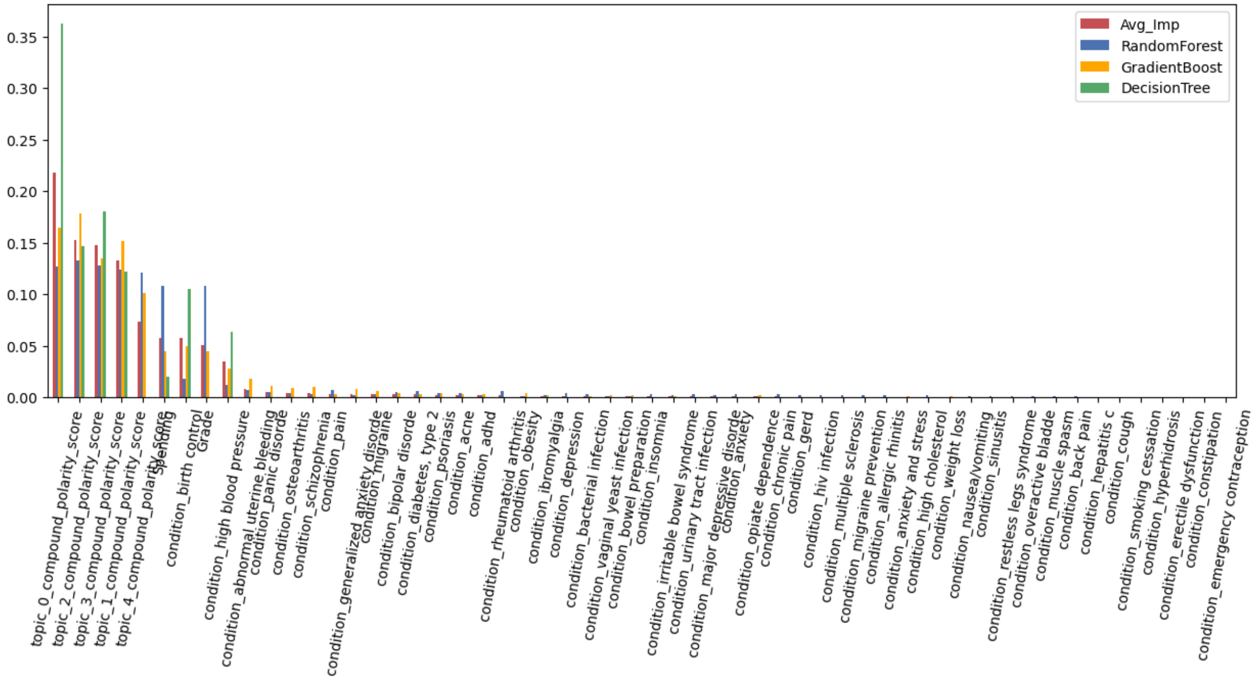
In [148…

```
voterf =efficacy_stack.named_estimators_['rf'].feature_importances_
votegbm =efficacy_stack.named_estimators_['gbm'].feature_importances_
votedt =efficacy_stack.named_estimators_['dt'].feature_importances_
votes=list(zip(voterf,votegbm,votedt))

voimp = dict(zip(train_X.columns,votes))
```

```python
label=['RandomForest','GradientBoost','DecisionTree']
final=pd.DataFrame(voimp.values(), index=voimp.keys(),columns=label)
final['Avg_Imp'] = final.mean(axis=1)

poslabel=['Avg_Imp','RandomForest','GradientBoost','DecisionTree']
final.sort_values(by =['Avg_Imp'], ascending = False).plot(y=poslabel,kind = 'bar',colo
```



In [149…

```python
as_list = final.index.tolist()
as_list

idx0 = as_list.index('topic_0_compound_polarity_score')
as_list[idx0] = 'Skin Health score'

idx1 = as_list.index('topic_1_compound_polarity_score')
as_list[idx1] = 'Pain score'

idx2 = as_list.index('topic_2_compound_polarity_score')
as_list[idx2] = 'Gastro Health score'

idx3 = as_list.index('topic_3_compound_polarity_score')
as_list[idx3] = 'Menstrual Health score'

idx4 = as_list.index('topic_4_compound_polarity_score')
as_list[idx4] = 'Mental Health score'

final.index = as_list

final
```

Out[149…

	RandomForest	GradientBoost	DecisionTree	Avg_Imp
Grade	0.108403	0.044953	0.000000	0.051119
Spending	0.108622	0.044666	0.020157	0.057815
Skin Health score	0.126940	0.164114	0.362767	0.217940

	RandomForest	GradientBoost	DecisionTree	Avg_Imp
Pain score	0.123836	0.151994	0.122219	0.132683
Gastro Health score	0.133228	0.177957	0.146270	0.152485
Menstrual Health score	0.127780	0.134535	0.180086	0.147467
Mental Health score	0.120648	0.100803	0.000000	0.073817
condition_abnormal uterine bleeding	0.006986	0.017708	0.000000	0.008231
condition_acne	0.003885	0.002963	0.000000	0.002283
condition_adhd	0.002429	0.003522	0.000000	0.001984
condition_allergic rhinitis	0.002197	0.000000	0.000000	0.000732
condition_anxiety	0.003037	0.000000	0.000000	0.001012
condition_anxiety and stress	0.000751	0.001425	0.000000	0.000725
condition_back pain	0.001113	0.000000	0.000000	0.000371
condition_bacterial infection	0.003133	0.001006	0.000000	0.001380
condition_bipolar disorde	0.005082	0.004014	0.000000	0.003032
condition_birth control	0.017819	0.049908	0.105076	0.057601
condition_bowel preparation	0.001477	0.002261	0.000000	0.001246
condition_chronic pain	0.002822	0.000000	0.000000	0.000941
condition_constipation	0.000134	0.000000	0.000000	0.000045
condition_cough	0.000544	0.000000	0.000000	0.000181
condition_depression	0.004321	0.000000	0.000000	0.001440
condition_diabetes, type 2	0.005951	0.003062	0.000000	0.003004
condition_emergency contraception	0.000000	0.000000	0.000000	0.000000
condition_erectile dysfunction	0.000161	0.000000	0.000000	0.000054
condition_generalized anxiety disorde	0.002197	0.008533	0.000000	0.003577
condition_gerd	0.002440	0.000000	0.000000	0.000813
condition_hepatitis c	0.000831	0.000000	0.000000	0.000277
condition_high blood pressure	0.012256	0.028192	0.063425	0.034624
condition_high cholesterol	0.001992	0.000000	0.000000	0.000664
condition_hiv infection	0.002423	0.000000	0.000000	0.000808
condition_hyperhidrosis	0.000287	0.000000	0.000000	0.000096
condition_ibromyalgia	0.002726	0.001924	0.000000	0.001550
condition_insomnia	0.003008	0.000635	0.000000	0.001214
condition_irritable bowel syndrome	0.002652	0.000845	0.000000	0.001166
condition_major depressive disorde	0.002666	0.000771	0.000000	0.001146

	RandomForest	GradientBoost	DecisionTree	Avg_Imp
condition_migraine	0.003772	0.005953	0.000000	0.003242
condition_migraine prevention	0.002252	0.000000	0.000000	0.000751
condition_multiple sclerosis	0.002313	0.000000	0.000000	0.000771
condition_muscle spasm	0.001385	0.000000	0.000000	0.000462
condition_nausea/vomiting	0.001753	0.000000	0.000000	0.000584
condition_obesity	0.001377	0.003850	0.000000	0.001743
condition_opiate dependence	0.000924	0.001908	0.000000	0.000944
condition_osteoarthritis	0.004505	0.009719	0.000000	0.004741
condition_overactive bladde	0.001479	0.000000	0.000000	0.000493
condition_pain	0.007083	0.003648	0.000000	0.003577
condition_panic disorde	0.005657	0.011072	0.000000	0.005577
condition_psoriasis	0.003999	0.004237	0.000000	0.002745
condition_restless legs syndrome	0.001482	0.000000	0.000000	0.000494
condition_rheumatoid arthritis	0.005860	0.000000	0.000000	0.001953
condition_schizophrenia	0.003762	0.009867	0.000000	0.004543
condition_sinusitis	0.001721	0.000000	0.000000	0.000574
condition_smoking cessation	0.000465	0.000000	0.000000	0.000155
condition_urinary tract infection	0.003483	0.000000	0.000000	0.001161
condition_vaginal yeast infection	0.001289	0.002684	0.000000	0.001324
condition_weight loss	0.000662	0.001271	0.000000	0.000644

In [150…

```
final.reset_index()
final.set_index('Avg_Imp')

final = final.sort_values(by=["Avg_Imp"], ascending=False)

final.head(10)
```

Out[150…

	RandomForest	GradientBoost	DecisionTree	Avg_Imp
Skin Health score	0.126940	0.164114	0.362767	0.217940
Gastro Health score	0.133228	0.177957	0.146270	0.152485
Menstrual Health score	0.127780	0.134535	0.180086	0.147467
Pain score	0.123836	0.151994	0.122219	0.132683
Mental Health score	0.120648	0.100803	0.000000	0.073817
Spending	0.108622	0.044666	0.020157	0.057815
condition_birth control	0.017819	0.049908	0.105076	0.057601

	RandomForest	GradientBoost	DecisionTree	Avg_Imp
Grade	0.108403	0.044953	0.000000	0.051119
condition_high blood pressure	0.012256	0.028192	0.063425	0.034624
condition_abnormal uterine bleeding	0.006986	0.017708	0.000000	0.008231

In [151…

```
final[:10].sort_values(by =['Avg_Imp'], ascending = False).plot(y=poslabel,kind = 'bar'
```