PROJECT REPORT

on

"Sentiment Analysis of Drug Review by Patients"

Submitted by

Group 5

Mentor: Anjana Agarwal

For the award of the degree

POST GRADUATE PROGRAM

in

DATA SCIENCE ENGINEERING

Great Lakes Institute of Management, Chennai

July 2020

Submitted By

Akash K Shetty,
Gautham S Nair,
Saurabh Sudhakar Karpe,
Saurabh Prashant Nagvekar,
Shashank Mrunalchandra Paralkar

Project Summary

Batch details	Mumbai July 2020
Team members	Akash K Shetty, Gautham S Nair, Saurabh Sudhakar Karpe, Saurabh Prashant Nagvekar, Shashank Mrunalchandra Paralkar
Domain of Project	Healthcare
Proposed project title	Sentiment Analysis of Drug Review by Patients
Group Number	5
Team Leader	Saurabh Sudhakar Karpe
Mentor Name	Anjana Agarwal

Date: 22/04/2021

Signature of the Mentor

Signature of the Team Leader

Table of Contents

ABSTRACT

Medication review is often recommended to optimize medication use. To reduce the number of preventable adverse drug events and hospital admissions, medication review is often recommended, incorporated in several guidelines, and also frequently reimbursed by health care insurers in various countries.

Drug utilization reviews will help ensure that drugs are used appropriately (for individual patients). In the drug utilization review, medicine and health history including all phases of dispensing for a patient is exactly listed. Also, this review is designed to attempt to attain proper decision-making therapeutically and gain a positive outcome for the patient.

In this project we consider the retrospective drug utilization review which refers to drug therapy review after patients have got the medication. The retrospective drug utilization review is a typical process wherein we categorize the review made by the patient based on NLP algorithms. The reviews are categorized into positive, negative and neutral words based on Countvectorizer and TF-IDFVectorizer.

CHAPTER 1

INTRODUCTION

The introductory chapter lays out the business problem, its significance and the objective.

1.1 Business Problem

Data from drug utilization research is an invaluable resource for all stake holders involved in drug and health policies. Drug utilization is “marketing, distribution, prescription and use of drugs in a society, with special emphasis on the resulting medical, social and economic consequences”.

Research on drug utilization includes factors related to prescribing, dispensing, administering and intake of medication and its associated events.

The ultimate purpose of drug utilization research is to estimate the optimal quality of drug therapy by identifying, documenting, analysing problems in drug utilization and monitoring the consequences. It encourages the prescribers to prescribe correct drug at appropriate dose and affordable price. It contributes to the knowledge of rational use of drugs in the society; whether the drug is being prescribed appropriately, whether the drug is taken in correct dosage, whether the drug is available at affordable price or misused. It provides valuable feedback about the rationality of the prescription to the doctors. It also assesses whether an intervention affects the drug use in the population by examining the outcomes of different types of intervention given to improve rationality in drug use.

Drug utilization research can be qualitative or quantitative and can be done by various methods. This review highlights the understanding of various aspects, different designs and WHO guidelines for conducting drug utilization research.

CHAPTER 2

DATASET DESCRIPTION

This chapter gives an insight of the dataset such as a description of the dataset with all the features along with Exploratory Data Analysis explaining the dataset and performing the needed pre-processing steps.

The dataset provides patient reviews on specific drugs along with related conditions and a 10star patient rating reflecting overall patient satisfaction. The data was obtained by crawling online pharmaceutical review sites.

2.1 Drug Review Dataset

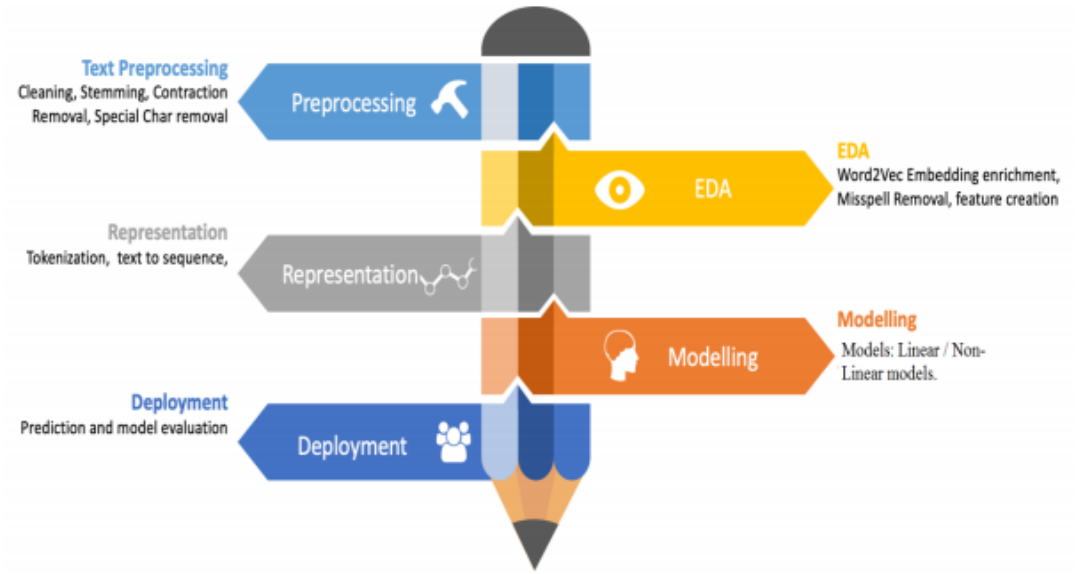
The dataset was originally published on the UCI Machine Learning repository.

The feature names as well as the description of each features are as described below:

Feature Name	Type	Description & Values
Unique ID	Numerical	Unique ID of registration of each patient.
Drug Name	Categorical	The name of the drug used by the patient.
Condition	Categorical	The medical condition of the patient.
Review	Categorical	The review of the drug by the patient. (includes side effects, positive reviews, consequences after usage)
Rating	Numerical	The rating of each drug based on the usage by patients. Rating from 1 to 10 (1-lowest, 10-highest)
Date	Categorical	The date on which the patient has entered the review of the drug.
Useful Count	Numerical	The number of users who found the reviews posted to be useful.

2.1 Pre-Processing

Pre-processing refers to the transformations applied to data before feeding it to the algorithm. Data pre-processing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.



Since the dataset deals with text columns which have raw text, that is pretty messy for these reviews so before we do any analytics, we have to pre-process the data and clean it before analysis.

- The null values for the whole dataset had been analysed and it was found out that the feature: Condition has 899 missing values. When the percentage computation is done compared to the whole data it comprises only 0.557% of the whole data which is only a meagre value. Hence, we had decided on to drop the null values and this is not affecting the further analysis.

- The condition feature was then analysed using span data and all the rows which had “No condition listed” (noisy data) was taken which computed to 439 and again it comprises of only 0.273% of the whole

data and hence this data is as well dropped as it is a meagre value in comparison to the whole data.

- A new feature “Sentiment” has been introduced into the dataset which has been categorized into “Positive”, “Negative” and “Neutral” on the basis of the rating provided by the customer where a rating below 4 is categorized as “Negative” and rating above 7 is categorized as “Positive”. The rating between 4 and 7 is categorized under “Neutral”.

- The date column which provides the date of review made by the patient as well as the unique Id which provides the registration Id of the patient at hospital is redundant for further analysis and henceforth it is dropped from the dataset.

- Lemmatization (Word Net) has been applied on the review column in order to filter the useful/insightful words from the review. Lemmatization is powerful and it looks beyond word reduction and considers a language’s full vocabulary to apply a morphological analysis to words, aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as lemma.

- A new column named as “Clean review” is introduced which comprises of the filtered/lemmatized words from the review column.

2.1 Exploratory Data Analysis

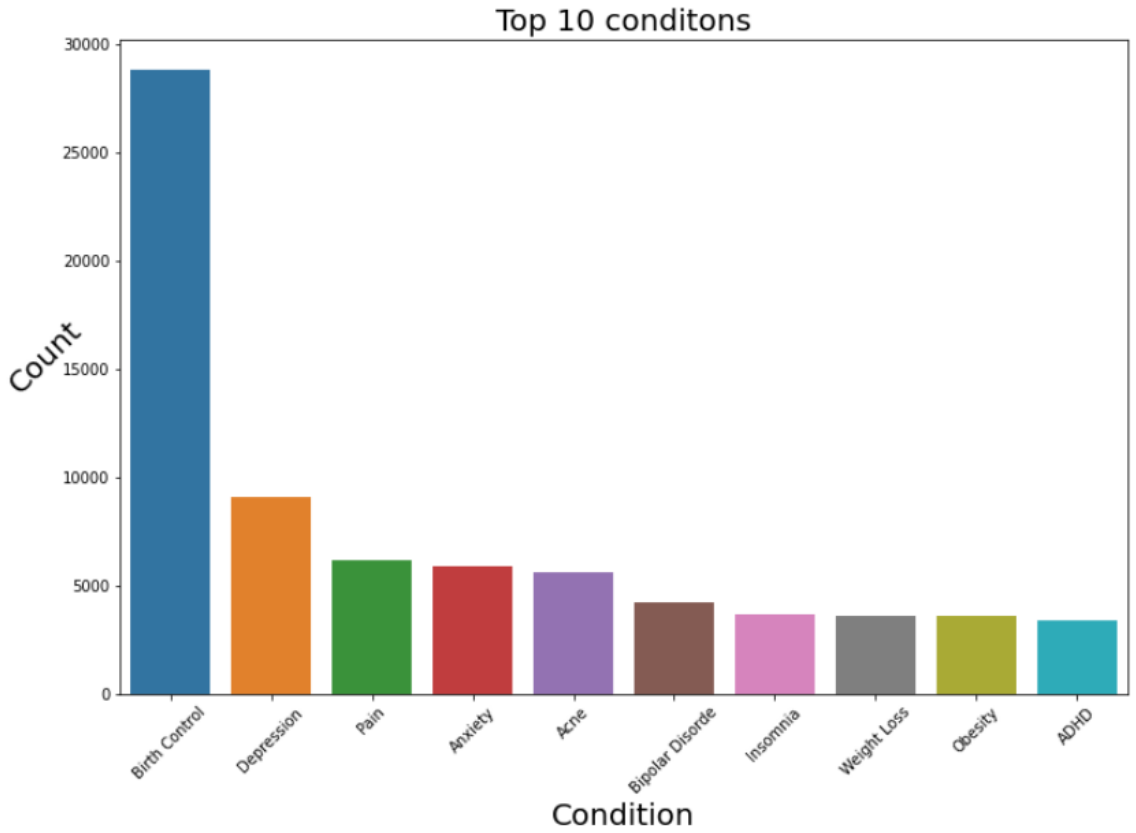
Univariate as well as bivariate analysis is done on the data set in order to draw more insights before deploying the machine learning models.

Exploratory Data Analysis can help answer questions about standard deviations, categorical variables, and confidence intervals. Once EDA is complete and insights are drawn, its features can then be used for more sophisticated data analysis or modelling, including machine learning.

In this dataset scenario of ours we are only dealing with categorical columns and not numerical, we have performed the below visualization of the data.

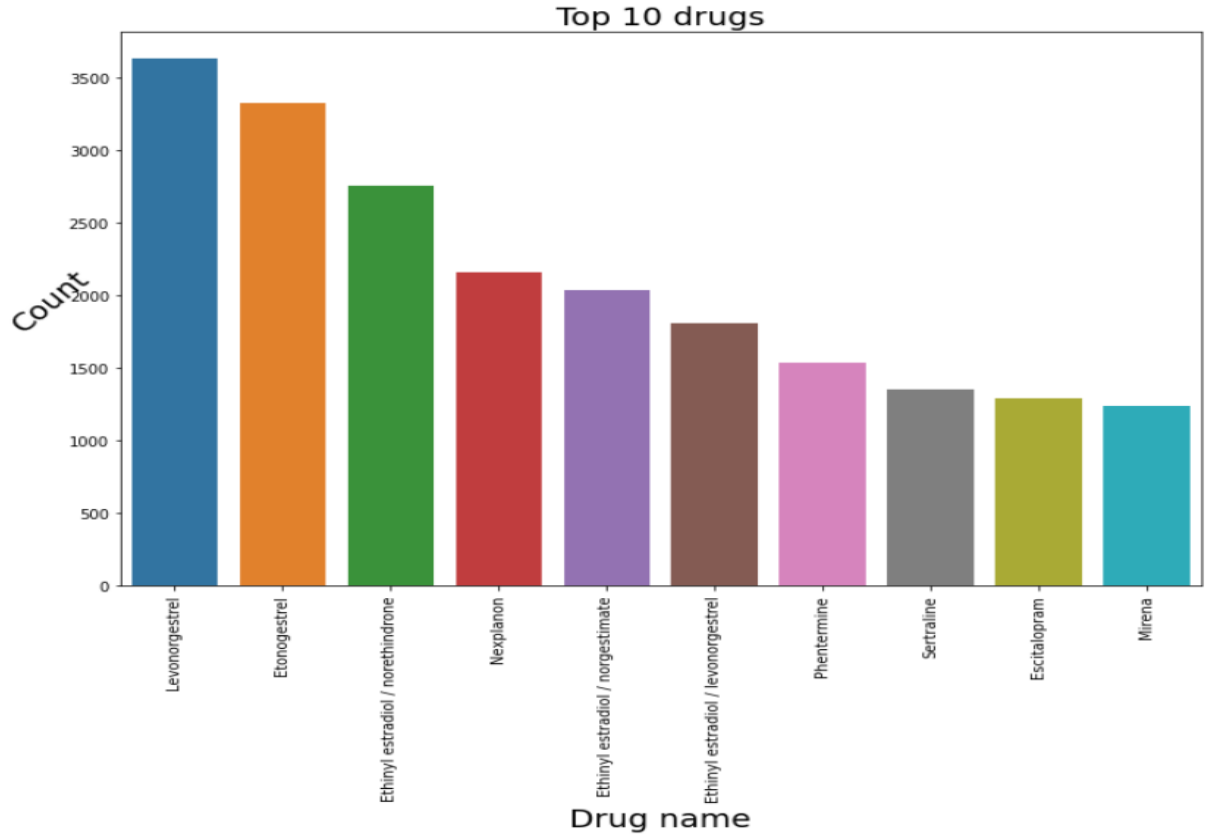
It has been identified that there are 3431 unique drug names in the dataset.

Top 10 conditions faced by the patients over a span of the last 10 years

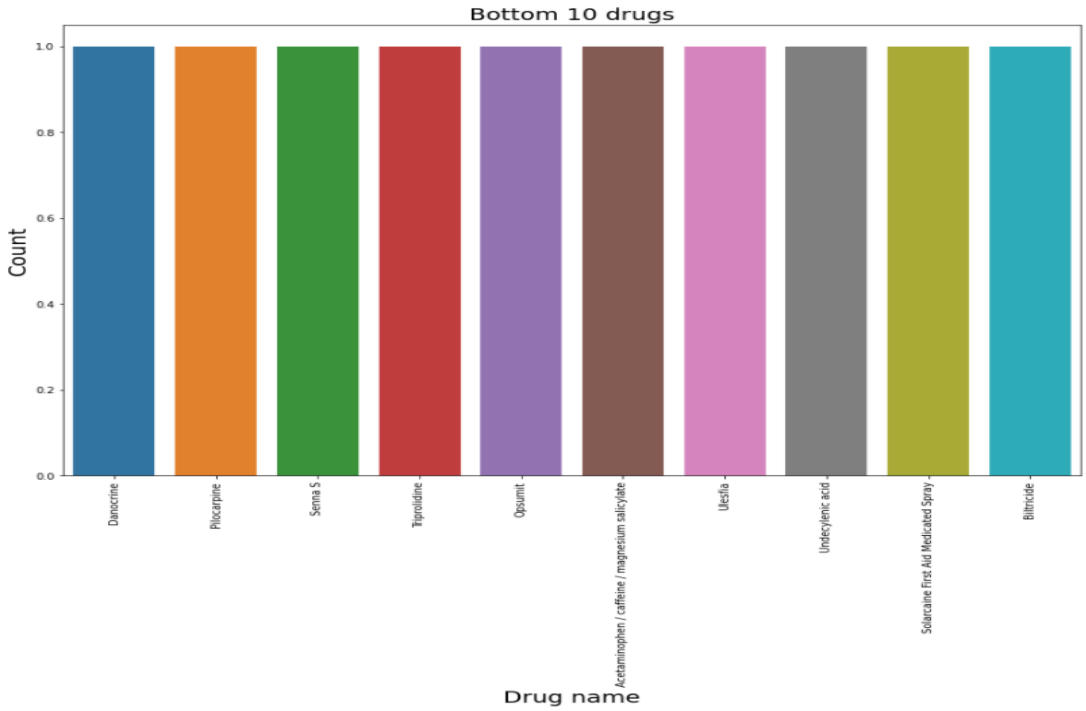


It can be observed from the above graph that the condition that has been seen widespread among all the patients is “Birth Control” followed by “ Depression”.

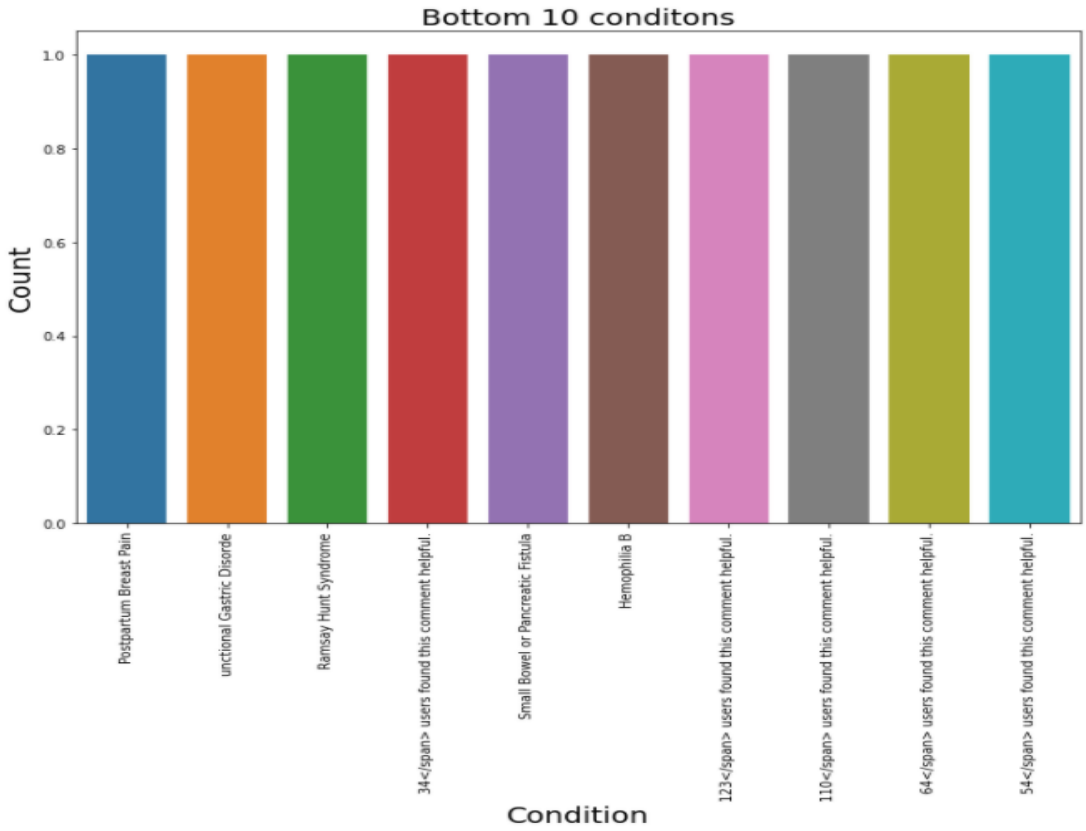
Top 10 most drugs used over a span of the last 10 years



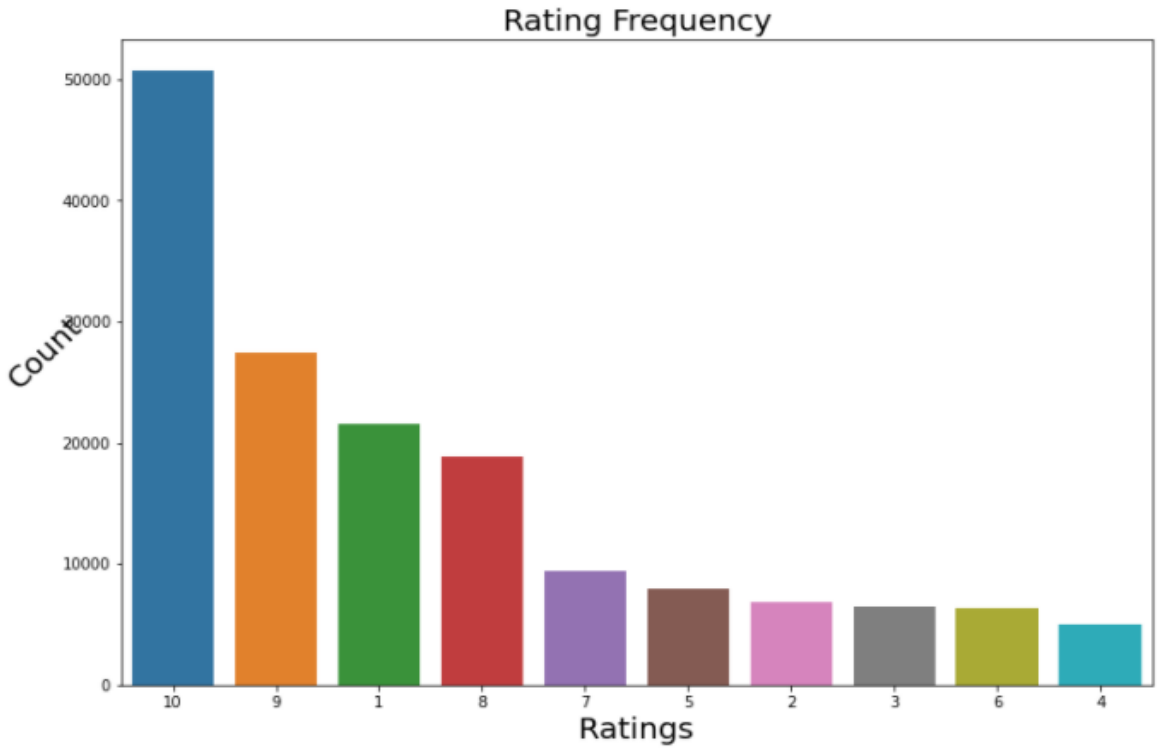
Top 10 least used drugs over a span of the last 10 years



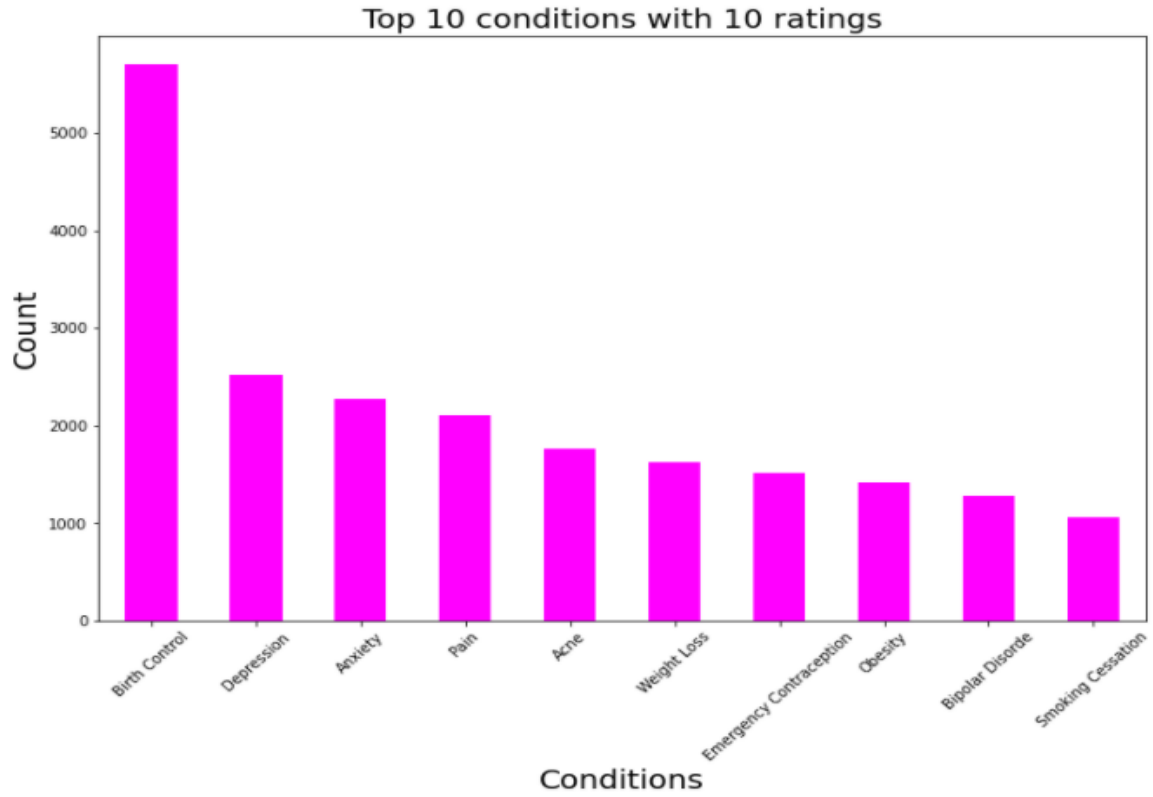
Bottom 10 conditions faced by the patients over a span of the last 10 years



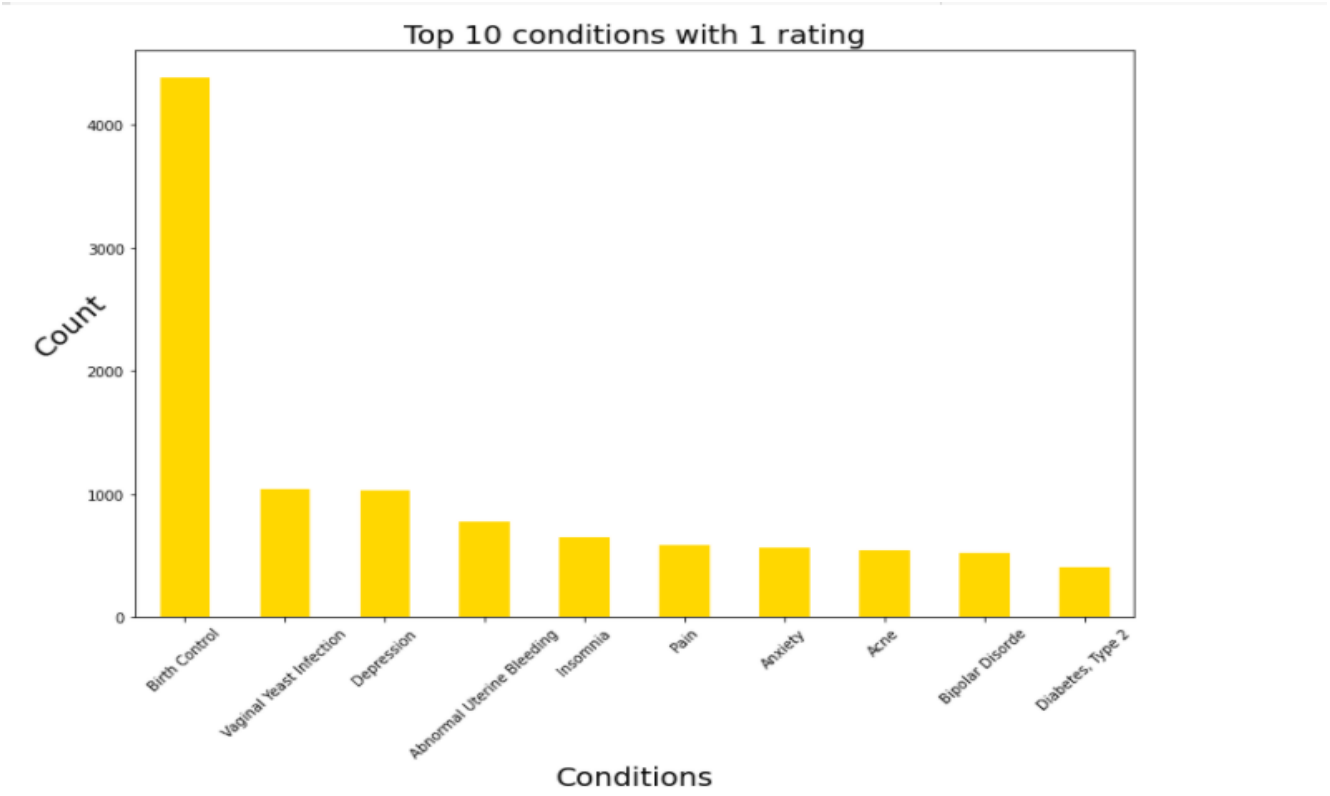
Bar-Graph of Frequency of each Rating



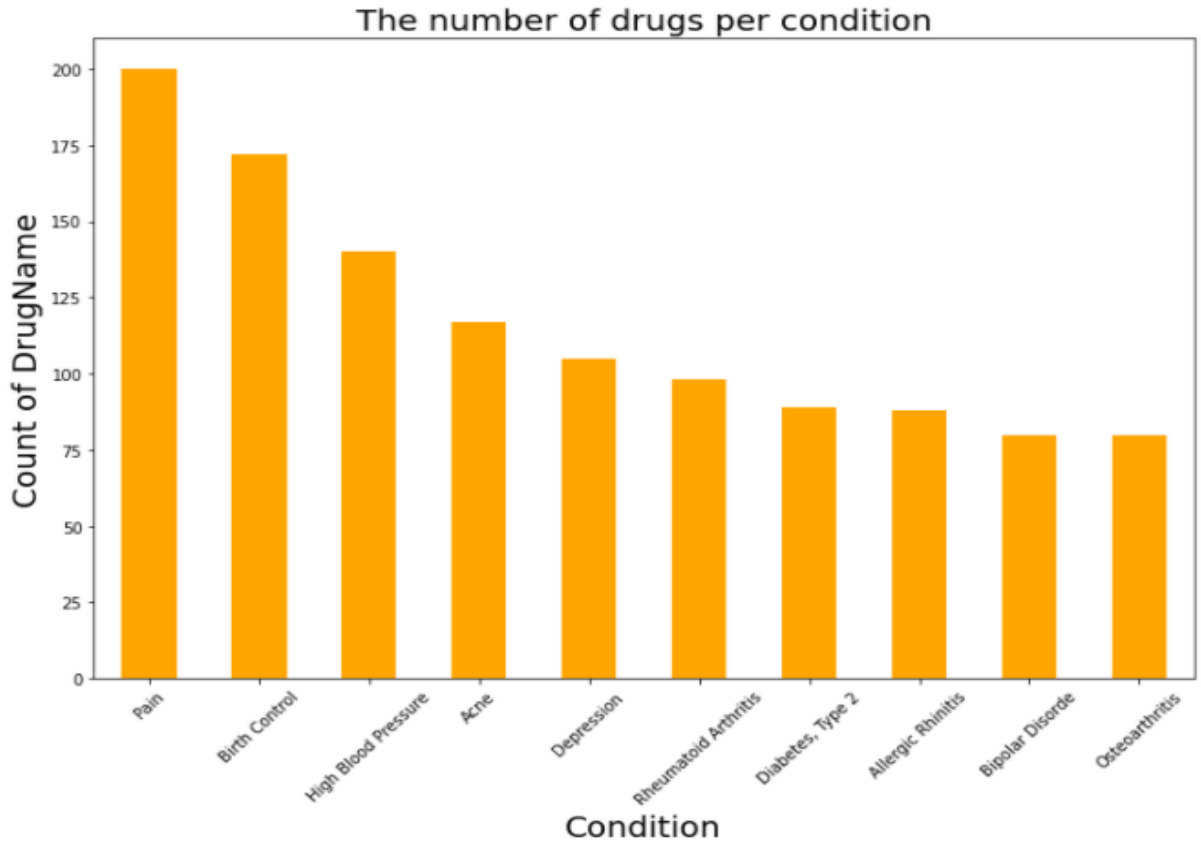
Top 10 conditions with rating equals 10



Top 10 conditions with rating equals 1

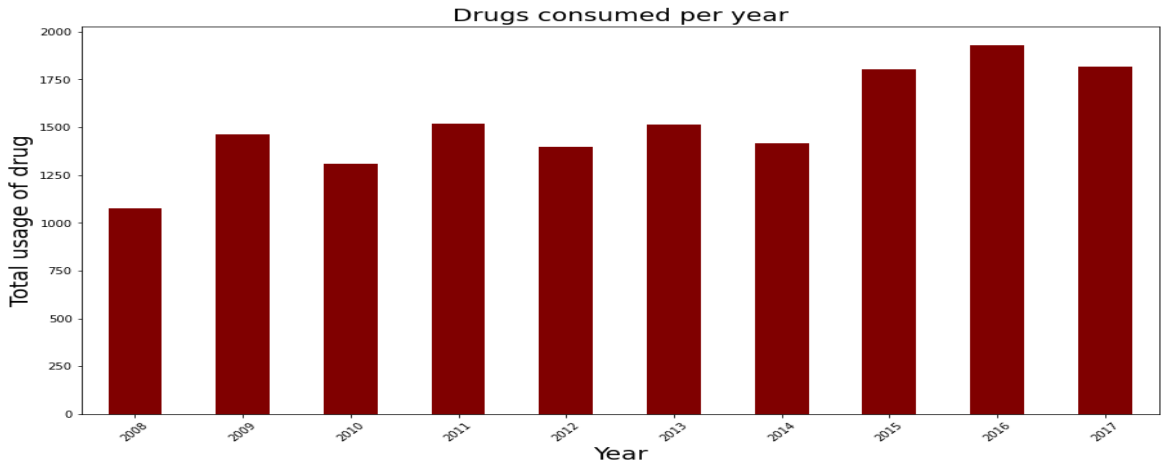


Number of drugs that has been prescribed for the same medical condition

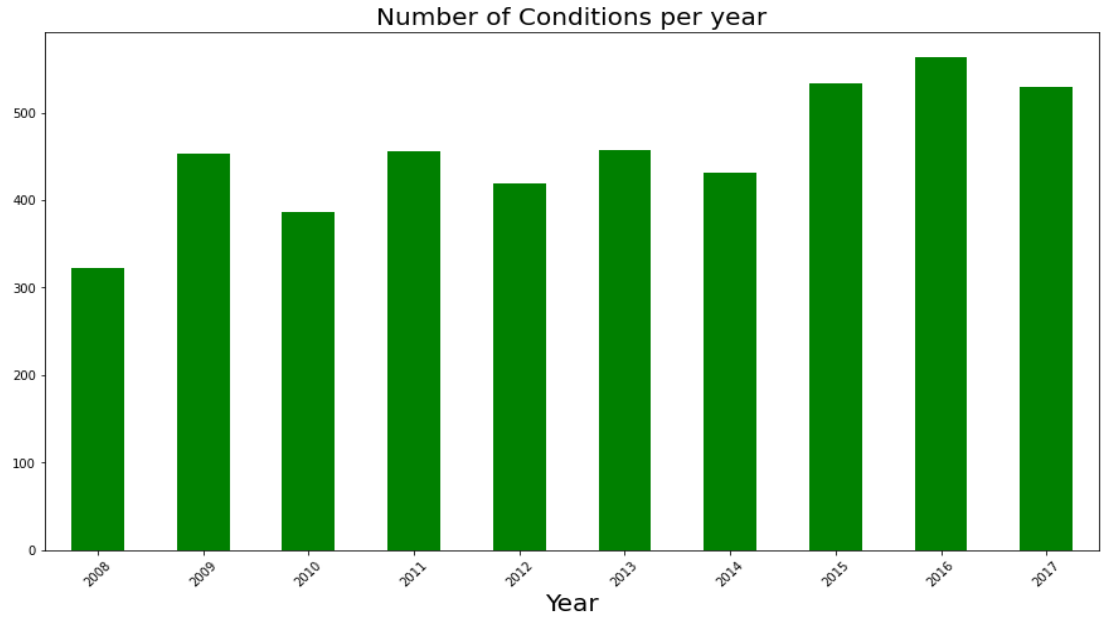


The count of drugs that are administered for each medical condition has been plotted as a bar graph and it is observed that the pain condition has several drugs prescribed as this is a common medical condition seen among patients.

Count of drugs consumed per year

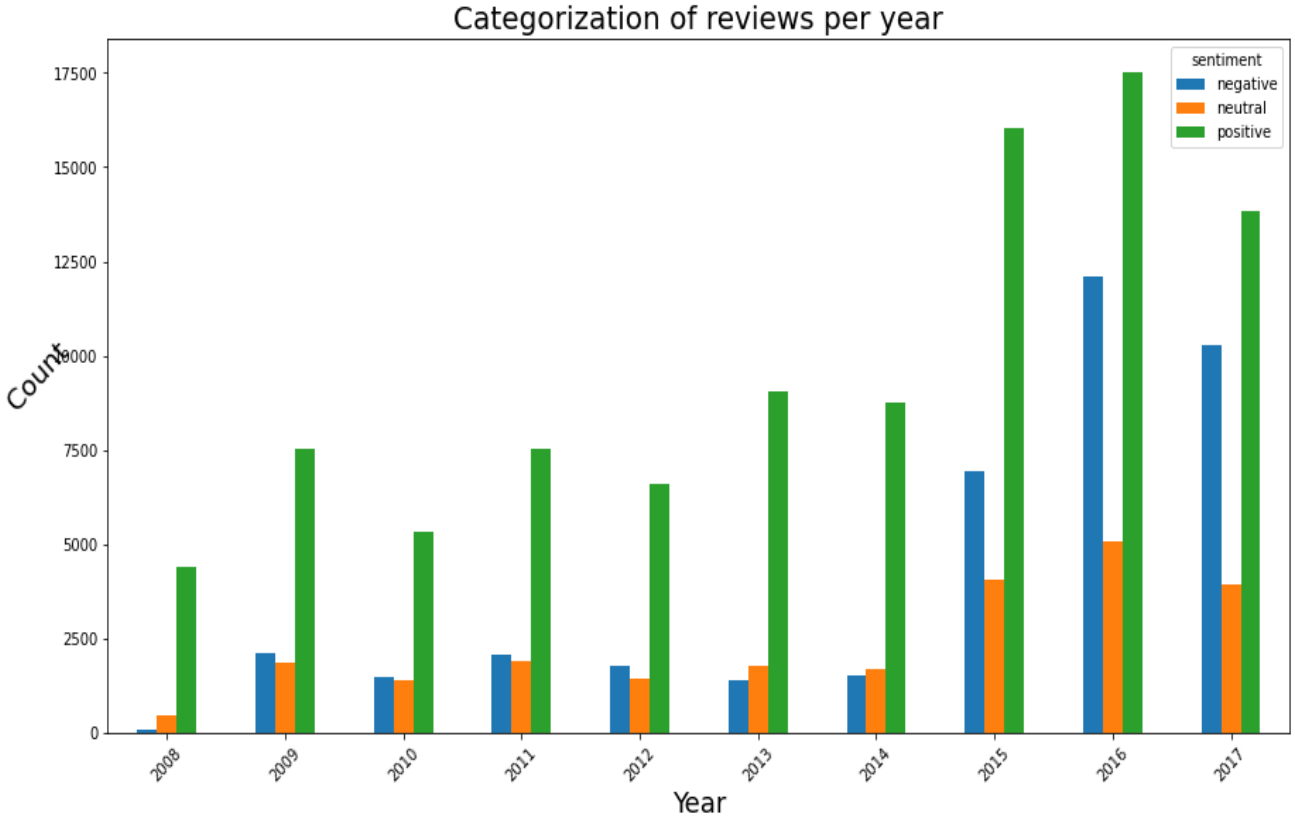


Conditions observed in patient on year-wise basis



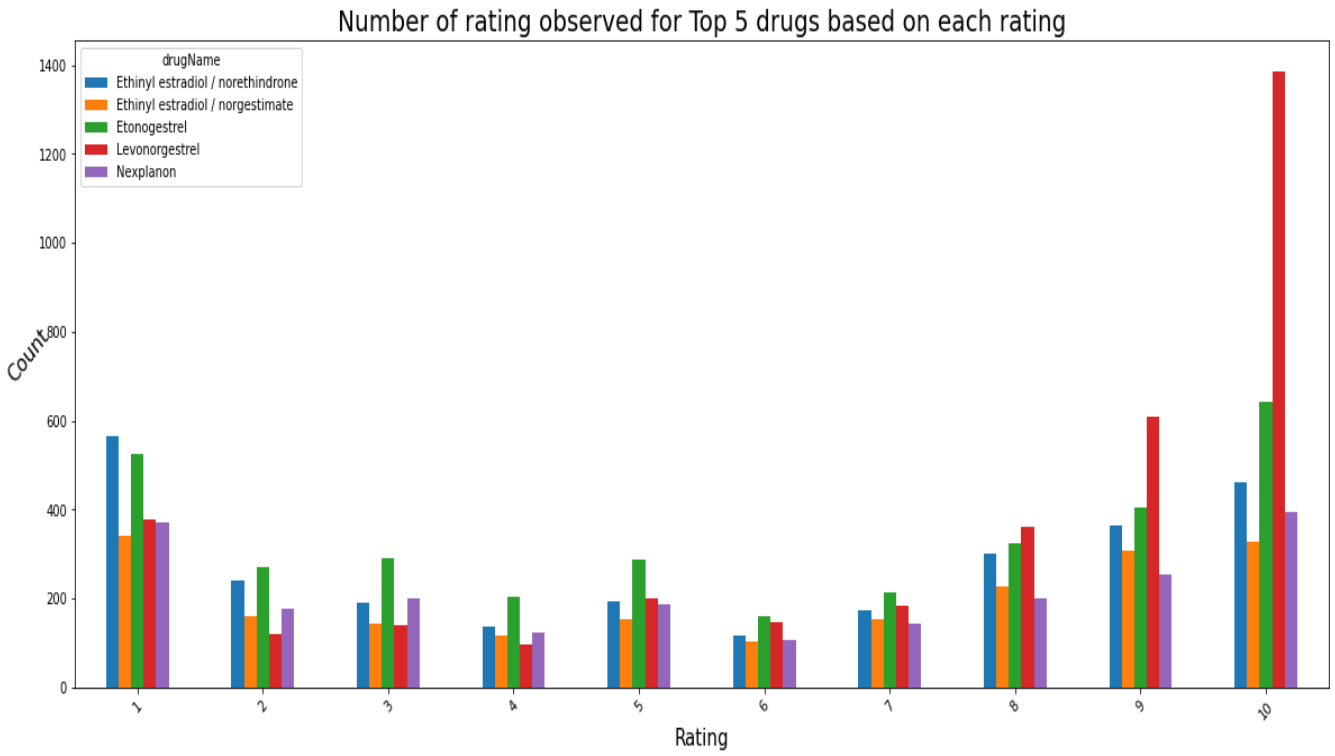
It is observed that the maximum number of conditions was observed in the year 2016.

Categorization of the review observed every year



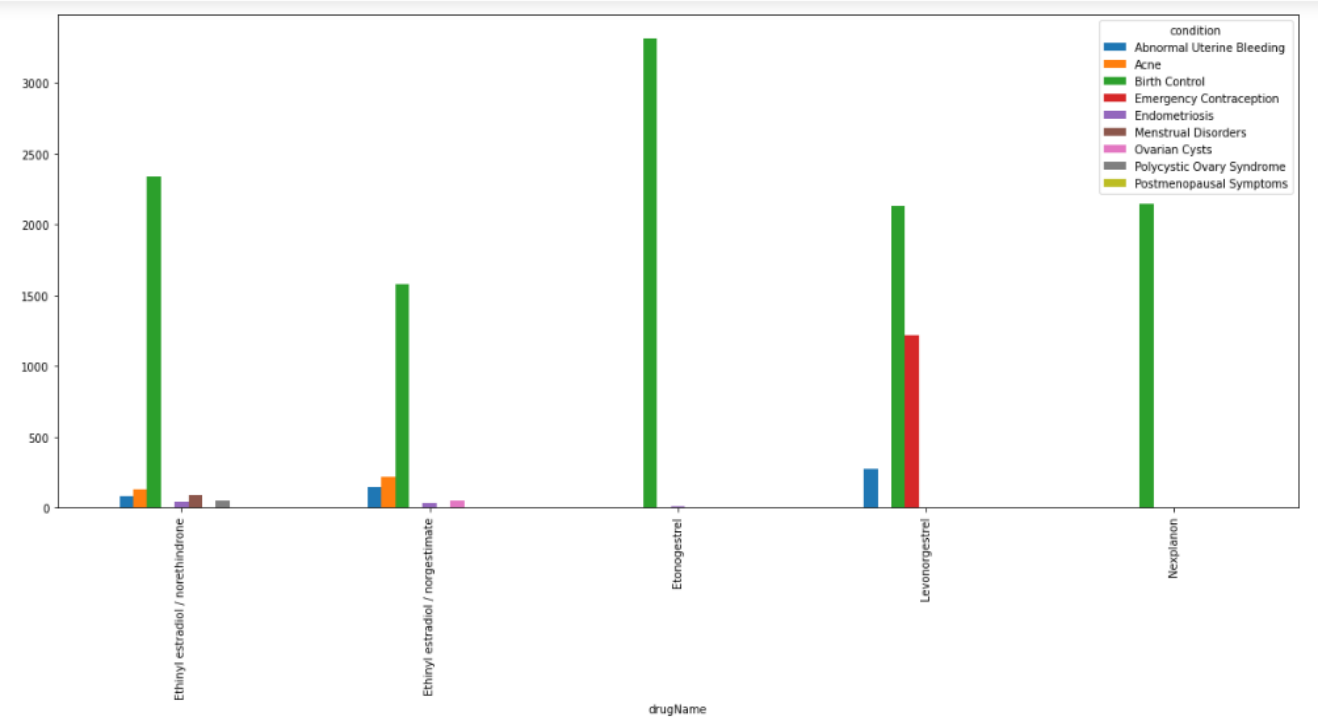
From the number of reviews, the highest number of reviews is observed in the year 2016(17500) and then followed by 2015(16000).

Count of each rating for the top 5 Drugs



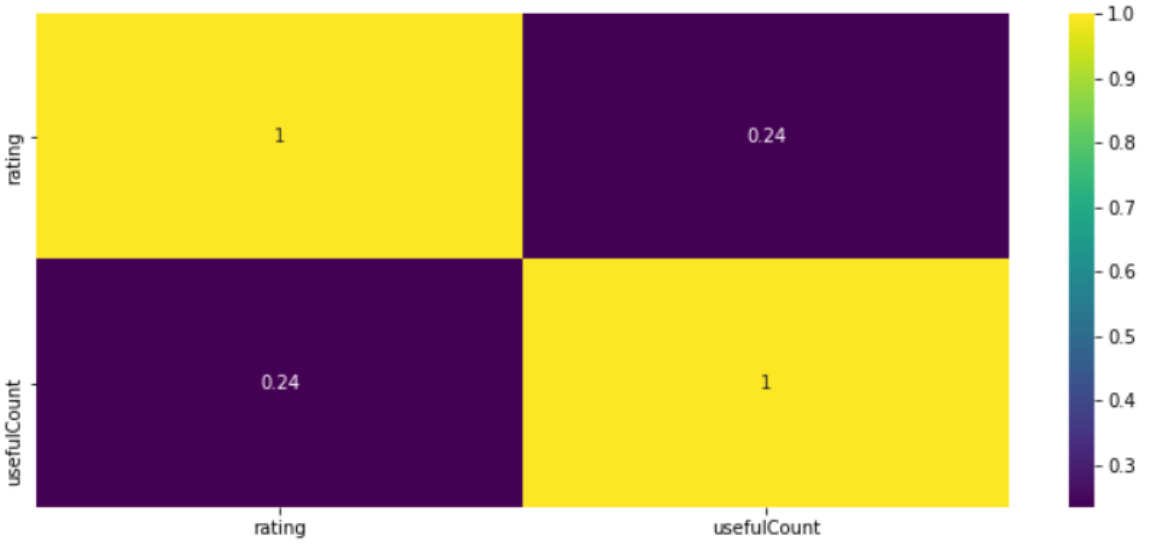
The above graph provides an insight into the number of ratings of the top 5 rated drugs. It can be inferred that the rating “10” has a higher frequency which clearly indicates that the rating of the drugs are positive and been effective in treating the patients ailments.

Drug used for particular Conditions :



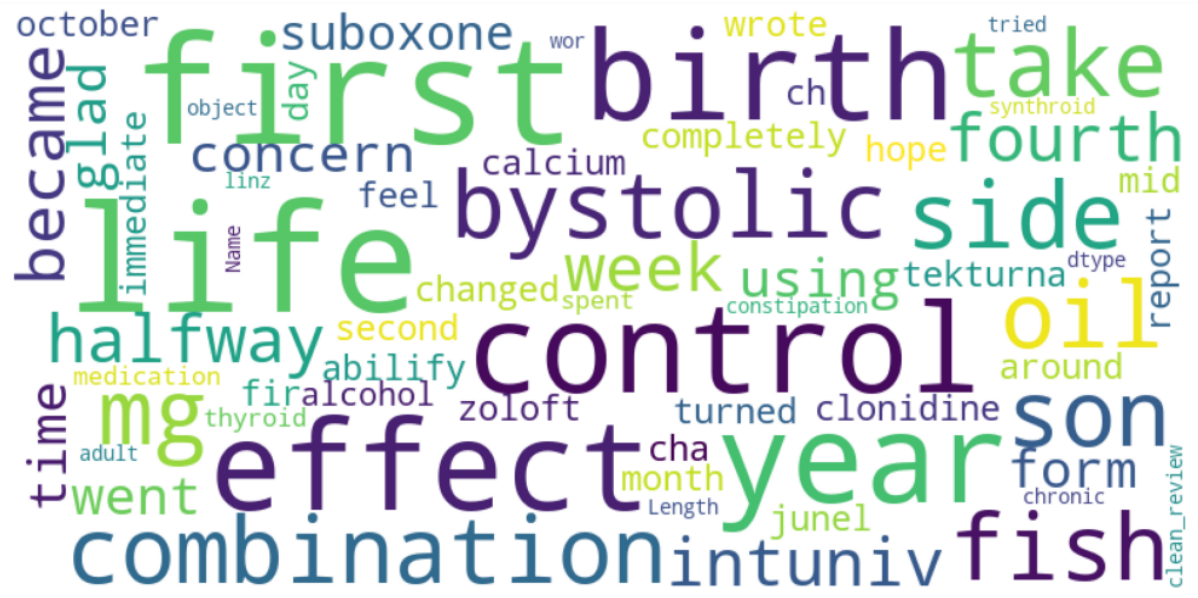
It is observed from the above plot that the rating 10 has the highest number of counts, which does indicate that the majority of the patients are satisfied with the drug that they have consumed. The rating 8 and 9 also do have a decent count which proves that the drugs are working fine in curing the illness. However, the second highest count is seen in the case of the rating 1 which proves that there are frequent cases wherein either the wrong drug was prescribed by the doctor to the patient or else the correct drug has not been majorly effective in treating the patient's illness.

Correlation Heatmap b/w Useful Count and Rating

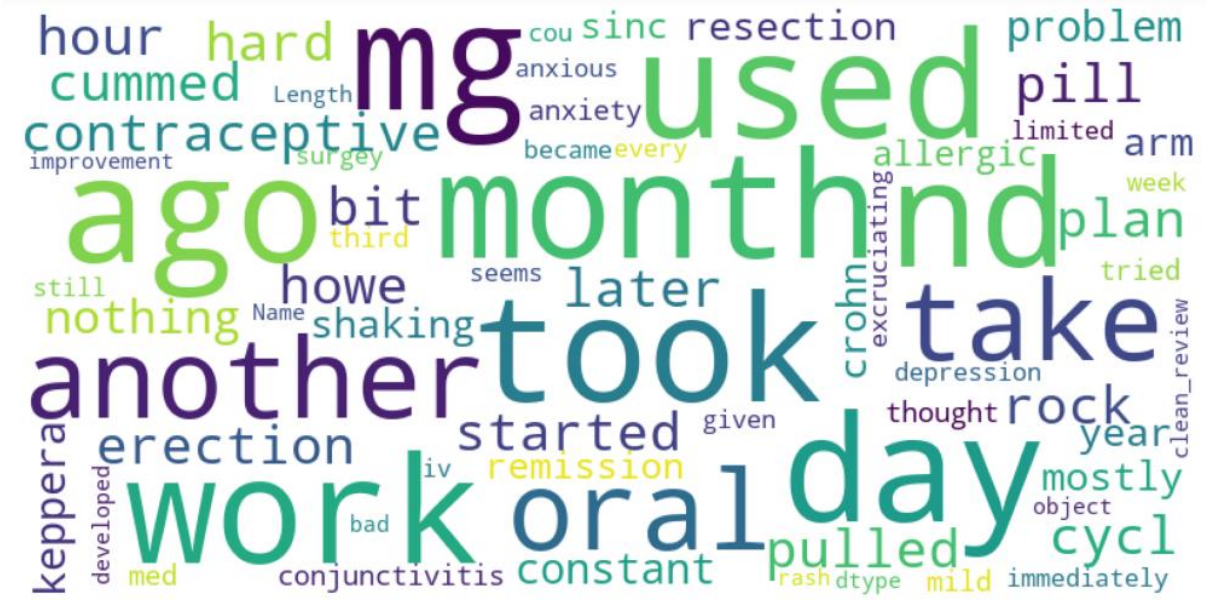


It is observed from the above correlation heatmap that there is not a good relationship between the numerical variables Useful Count and Rating.

Word Cloud for Positive Review:



Word Cloud for Negative Review:



2.2 Inferences from EDA

1. There are 3431 unique drugs that is considered in the dataset.
2. 884 unique conditions are experienced by the patients and this is considered in the dataset.
3. The drug that is most frequently used is “levonorgestrel”.
4. It is also observed that the patients majorly use a set of 79 drugs and the rest of 805 drugs are used in rare or unique conditions.
5. The condition that is observed in most of the patients is “Birth Control” and people look for treatment for this specific condition.
6. 142 conditions are rarely seen among the patients.

7. Most of the patients have rated 10 as part of the respective drug review, however it is also observed that the rating 1 has the second highest number of reviews in comparison to the rating 7 and 8.

8. The drug that has been rated 1 the most is Miconazole.

9. A unique observation that is seen is that irrespective of the rating, “Birth Control” is omnipresent in all the ranges of the rating. This clearly states that the effectiveness of the pills is questionable in terms of preventing the birth.

10. For 39 conditions, drug named “prednisone” is the most frequently used one.

11. It is also observed that variety of drugs are prescribed for the conditions pain as well as birth control.

12. The drug review for Fluoxetine and Gabapentin are found out to be useful for not only 1 condition but many conditions, which proves that these drugs are effective on multiple conditions.

13. The highest amount of drug consumption was observed in the year 2016.

14. While plotting the correlation heatmap between the numerical variables “Rating” and Useful Count” it is observed that the correlation value is not high and therefore establishes no relationship among the only numerical columns present.

CHAPTER 3

TEXT VECTORIZER MODEL

3.1 Need for Text Vectorizer Model

Text data requires special preparation before you can start using it for predictive modelling. The text must be parsed to remove words, called tokenization. Then the words need to be encoded as integers or floating-point values for use as input to a machine learning algorithm, called feature extraction (or vectorization).

The scikit-learn library provides 2 different schemes that can be used, and so will briefly look at each one.

3.2 Words count with Count Vectorizer

The Count Vectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary.

It can be used as follows:

1. Create an instance of the Count Vectorizer class.

2. Call the fit() function in order to learn a vocabulary from one or more documents.

3. Call the transform() function on one or more documents as needed to encode each as a vector.

An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document. Because these vectors will contain a lot of zeros, call them sparse.

Python provides an efficient way of handling sparse vectors in the scipy.sparse package. The vectors returned from a call to transform() will be sparse vectors, and you can transform them back to numpy arrays to look and better understand what is going on by calling the toarray() function.

3.3 Words frequencies with TF-IDF Vectorizer

Word counts are good to start while working with text data but is very basic.

One issue with simple counts is that some words like “the” will appear many times and their large counts will not be very meaningful in the encoded vectors. An alternative is to calculate word frequencies, and by far the most popular method is called TF-IDF.

This is an acronym that stands for “Term Frequency – Inverse Document” Frequency which are the components of the resulting scores assigned to each word.

- **Term Frequency:** This summarizes how often a given word appears within a document.

- **Inverse Document Frequency:** This downscales words that appear a lot across documents.

TF-IDF is word frequency scores that tries to highlight interesting words, e.g., frequent in a document but not across documents. The Tfidf Vectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents. Alternately, if you already have a learned Count Vectorizer, you can use it with a Tfidf Transformer to just calculate the inverse document frequencies and start encoding documents. The same create, fit, and transform process is used as with the Count Vectorizer.

3.4 Selection of Text to Vector model

Since both CountVectorizer and TF-IDF are popular models to convert a string of text to numerical values, both these models are tried and tested in this project, so the better ones can be chosen for further applying predictive models. It is also important to apply some of the most advanced NLP techniques to process or clean the text. These text cleaning or processing techniques that are considered for this project are Regex – Regular expressions, Removal of stop words, Lemmatization, Stemming – Potter and Snowball, and spelling mistake correction. All these techniques improve the model by reducing the redundancy, data size, and thereby inefficient prediction by the model. Detailed explanations about these NLP text processing techniques are included in the above sections of this report.

Since each of these techniques works differently, some don't go well with another, therefore all different possible combinations of these text processing techniques are applied and converted into a sparse matrix using CountVectorizer and TF-IDF, then MultinomialNB , Decision Tree Classifier and Random Forest Classifier models are used for prediction of the sentiment. In the training data, they have positive, neutral and negative statements, the model's performance will depend on how efficiently the text is processed for the model to learn and differentiate between the two distinctive sentiments. The resulting comparison will help choose which of the two techniques will be considered for our project.

In CountVectorizer we only count the number of times a word appears in the document which results in biasing in favour of most frequent words. this ends up in ignoring rare words which could have helped in processing our data more efficiently.

In TF-IDFVectorizer we consider overall document weightage of a word. It helps us in dealing with most frequent words. Using it we can penalize them. TF-IDFVectorizer weights the word counts by a measure of how often they appear in the documents.

CountVectorizer counts the frequency of all words in our corpus, sorts them and grabs the most recurring features (using max features hyperparameter). But these results are mostly biased and our model might lose out on some of

the important less frequent features. Hence, we have decided to built a model using TF-IDF vectorizer.

CountVectorizer for RandomForest:

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(ngram_range=(2,2),max_features=5000)

X = cv.fit_transform(df['clean_review'].values).toarray()
```

```
y = df.sentiment.replace({'positive':0,'negative':1,'neutral':2})
```

```
f1 score: 0.62762862645858
```

Tfidf for RandomForest:

```
from sklearn.feature_extraction.text import TfidfVectorizer
cv = TfidfVectorizer(ngram_range=(2,2),max_features=5000)

X = cv.fit_transform(df['clean_review'].values).toarray()
```

```
y = df.sentiment.replace({'positive':0,'negative':1,'neutral':2})
```

```
f1 score: 0.6373429873599333
```

We got F1 score comparatively high in Tfidf in our base model. In TF-IDFVectorizer we consider overall document weightage of a word. It helps us in dealing with most frequent words. Using it we can penalize them. TF-IDFVectorizer weights the word counts by a measure of how often they appear in the documents.

3.5 Selection of N-Grams

In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sample of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application. The n-grams typically are collected from a text or speech corpus. When the items are words, n-grams may also be called shingles.

Turns out that is the simplest bit, an N-gram is simply a sequence of N words. For instance, let us take a look at the following examples.

San Francisco (is a 2-gram)

The Three Musketeers (is a 3-gram)

She stood up slowly (is a 4-gram)

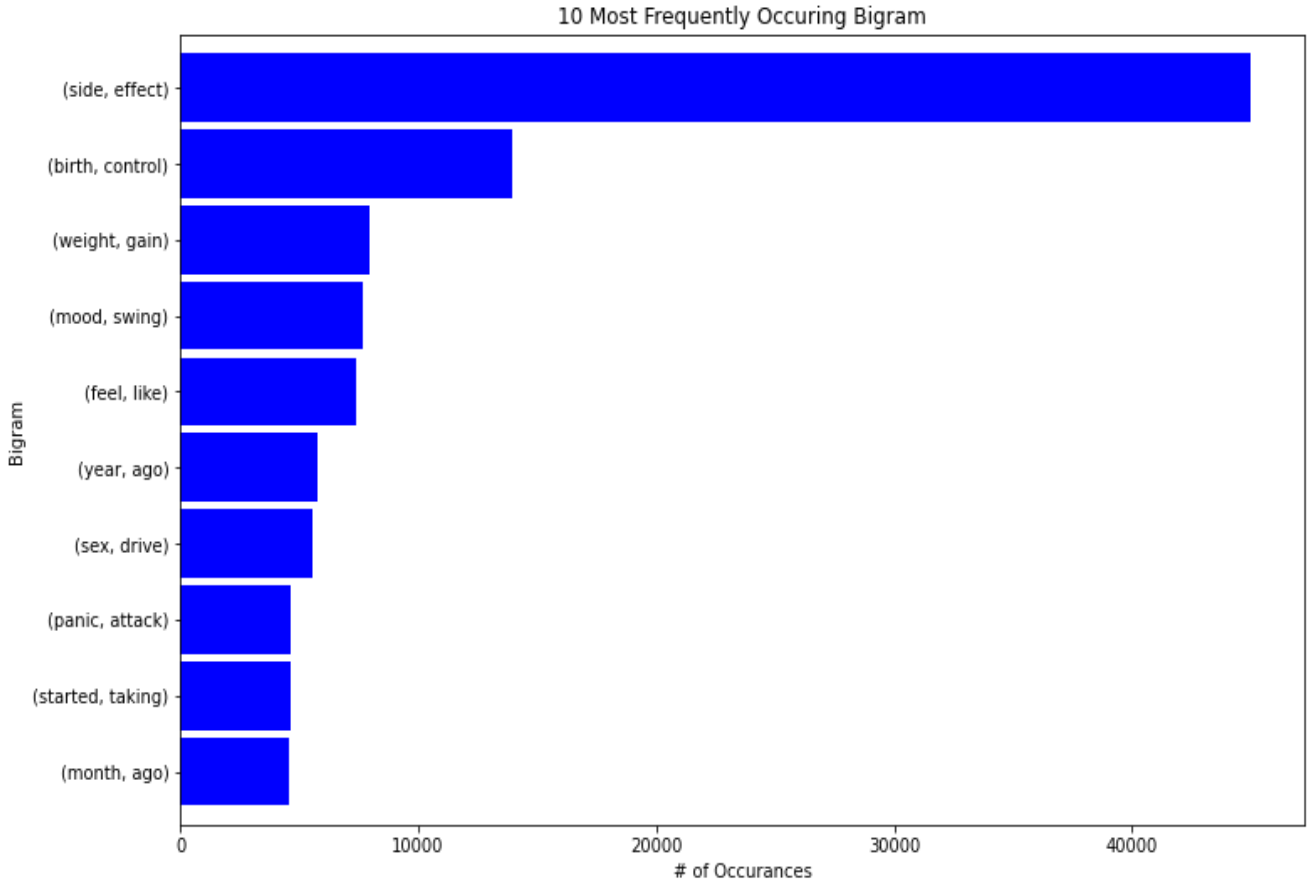
Now which of these three N-grams have you seen quite frequently? Probably, “San Francisco” and “The Three Musketeers”. On the other hand, you might not have seen “She stood up slowly” that frequently. Basically, “She stood up slowly” is an example of an N-gram that does not occur as often in sentences as Examples 1 and 2.

It can also help make next word predictions. Say you have the partial sentence “Please hand over your”. Then it is more likely that the next word is going to be “test” or “assignment” or “paper” than the next word being “school”.

It can also help to make spelling error corrections. For instance, the sentence “drink cofee” could be corrected to “drink coffee” if you knew that the word “coffee” had a high probability of occurrence after the word “drink” and also the overlap of letters between “cofee” and “coffee” is high. As you can see, assigning these probabilities has a huge potential in the NLP domain.

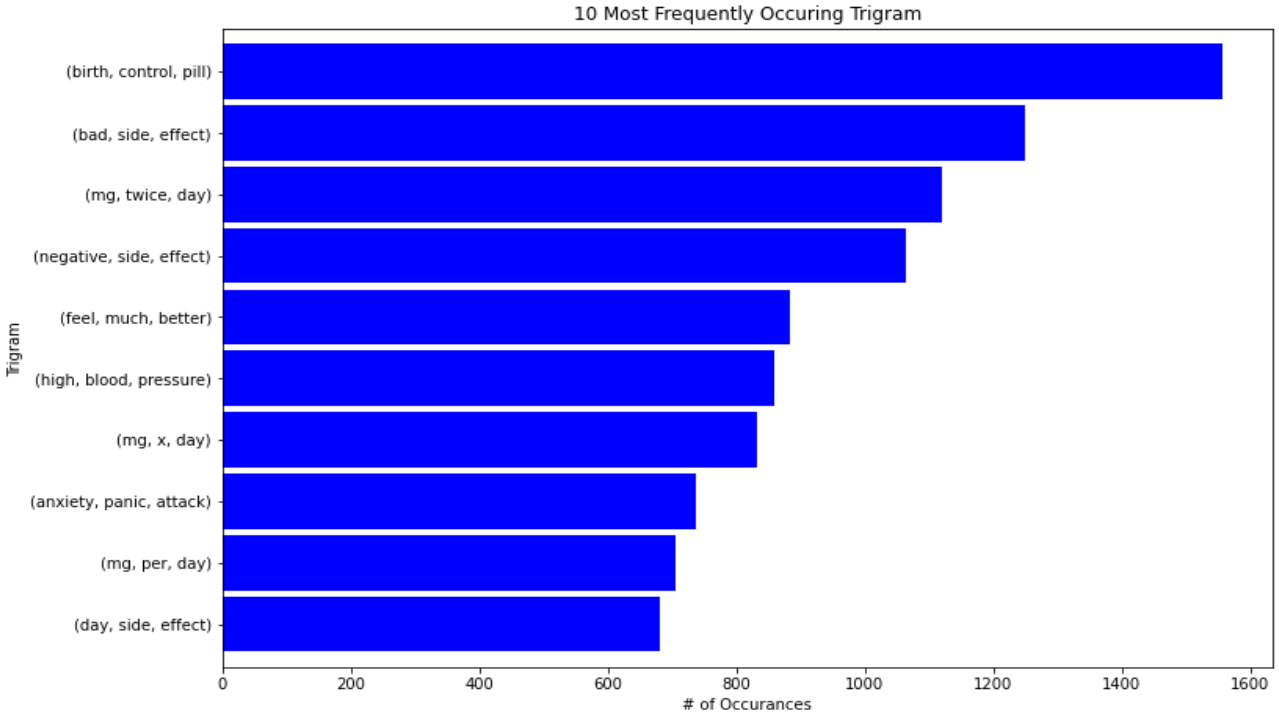
For sentiment analysis purpose on reviews, it is simply sufficient to train the model based on bigrams (2 letters) that will include the words that give out the negative sentiment when paired for example: do not, did not, have not, were not, etc. These words give the entire negative meaning only then they are combined as bigrams. Now, in text vectorization technique, there's option to selection N_gram.

We had applied **Bigram** on the dataset and it is inferred from the below graph that the most frequently occurring words in the sentences are plotted below.



It is observed from the above horizontal bar graph that the count of the combination of the words "side" and "effect" is very high and above 40,000 which clearly indicates that most of the patients could be feeling side effects after the consumption of the drugs. "Birth" and "control" is the combination that comes in the second place followed by "weight" and "gain".

We had also tried applying **Trigram** on the dataset and from model implementation it is observed that the trigram model gives less accuracy, precision and F1 score and hence we have decided to select bigram from further analysis and model deployment.



The trigram gives us a different combination and it is observed that with the word “pill” the frequency of birth control is now with the highest frequency and side effect comes at second place.

CHAPTER 4

Libraries Used in Project

1. **NumPy:** NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

2. **Pandas:** pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

3. **Matplotlib:** matplotlib. pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

4. **Seaborn:** Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics

5. **RegEx:** A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern. RegEx can be used to check if a string contains the specified search pattern.

6. **Train Test_Split:** The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model.

7. **Beautiful Soup:** Beautiful Soup is a Python package for parsing HTML and XML documents. It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping.

8. **NLP (Natural Language Processing):** Natural language processing (NLP) is a branch of artificial intelligence that helps computers understand, interpret and manipulate human language. NLP draws from many disciplines, including computer science and computational linguistics, in its pursuit to fill the gap between human communication and computer understanding.
   Natural language processing (NLP) is a method to translate between computer and human languages. It is a method of getting a computer to understandably read a line of text without the computer being fed some sort of clue or calculation. In other words, NLP automates the translation process between computers and humans.

9. **Nltk(Natural Language Toolkit) :** The Natural Language Toolkit (NLTK) is a platform used for building Python programs that work with human language data for applying in statistical natural language processing (NLP). It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning. It also includes graphical demonstrations and sample data sets as well as accompanied by a cook book and a book which explains the principles behind the underlying language processing tasks that NLTK supports.

10. **CountVectorizer /Bag of Words:** Bag of Words model is used to preprocess the text by converting it into a bag of words, which keeps a count of the total occurrences of most frequently used words.

    This model can be visualized using a table, which contains the count of words corresponding to the word itself.

Eg:-

- Sentence 1 = “He is good boy” ---> “good boy” after tokenization and lemmatization
- Sentence 2 = “He is good girl” ---> “good girl” after tokenization and lemmatization
- “good” = 2 | “boy” = 1 | “girl” = 1

Vectors:

Word	good	boy	girl
Sentence1	1	1	0
Sentence2	1	0	1

11. **Tf-idf:** TF-IDF or (Term Frequency (TF) — Inverse Dense Frequency (IDF) is a technique which is used to find meaning of sentences consisting of words and cancels out the incapability of Bag of Words technique which is good for text classification or for helping a machine read words in numbers.

Formula for tf-idf:

$$\textbf{TF} = \frac{\textbf{(no of repetition in sentence)}}{\textbf{(no of words in sentences)}}$$

$$\textbf{IDF} = \frac{\textbf{log(no of sentences)}}{\textbf{(no of sentences containing that words)}}$$

Eg:

- Sentence 1 = “He is good boy” ---> “good boy” after tokenization and lemmatization
- Sentence 2 = “He is good girl” ---> “good girl” after tokenization and lemmatization
- Sentence 3 = “boys and girls are good” ---> “good boy girl”
- “good” = 2 | “boy = 1” | “girl = 1”

TF e.g:

Word	Sen1	Sen2	Sen3
good	1/2	1/2	1/3
boy	1/2	0	1/3
girl	0	1/2	1/3

IDF e.g:

Word	IDF
good	log(3/3)=0
boy	log(3/2)
girl	log(3/2)

Vectors = TF * IDF

Word	good	boy	girl
Sen1	1/2*log(3/3)	1/2*log(3/2)	0*log(3/2)
Sen2	1/2*log(3/3)	0*log(3/2)	1/2*log(3/2)
Sen3	1/3*log(3/3)	1/3*log(3/2)	1/3*log(3/2)

12. **WordCloud :** A word cloud (also called tag cloud or weighted list) is a visual representation of text data. Words are usually single words, and the importance of each is shown with font size or colour. Python fortunately has a word cloud library allowing to build them.

13. **WordNetLemitizer :-** Stemming and Lemmatization both generate the root form of the inflected words. The difference is that stem might not be an actual word whereas, lemma is an actual language word. Stemming just removes or stems the last few characters of a word, often leading to incorrect meanings and spelling. Lemmatization considers the context and converts the word to its meaningful base form, which is called Lemma. Sometimes, the same word can have multiple different Lemmas

stemmimg eg :- history , historical --> histor , eating, eats, eaten --> eat.

lemmatization eg : - history , historical --> history , final , finally , finalize ---> final

14. **StopWords :** Stopwords are the most common words in any natural language. **Stopwords** are the English **words** which does not add much **meaning** to a sentence. For the purpose of analyzing text data and building NLP models, these stopwords might not add much value to the meaning of the document. Generally, the most common words used in a text are “the”, “is”, “in”, “for”, “where”, “when”, “to”, “at” etc. They can safely be ignored without sacrificing the **meaning** of the sentence. Such **words** are already captured this in corpus named corpus.

15. **Sentimental Analysis :** Sentiment Analysis is the process of determining whether a piece of writing is positive, negative or neutral. A sentiment analysis system for text analysis combines natural language processing (NLP) and machine learning techniques to assign weighted sentiment scores to the entities, topics, themes and categories within a sentence or phrase.

CHAPTER 5

CLASSIFICATION MODELS

5.1 Multinomial Naive Bayes

The general term Naive Bayes refers to the strong independence assumptions in the model, rather than the particular distribution of each feature. A Naive Bayes model assumes that each of the features it uses is conditionally independent of one another given some class. Naive Bayes is used in Text classification/ Spam Filtering/ Sentiment Analysis. It is used in text classification (it can predict multiple classes and doesn't mind dealing with irrelevant features).

Multinomial Naive Bayes classifier is a specific instance of a Naive Bayes classifier that uses a multinomial distribution for each of the features. Multinomial Naïve Bayes consider a feature vector where a given term represents the number of times it appears or very often.

Advantages:

- Low computation cost.
- It can effectively work with large datasets.
- For small sample sizes, Naive Bayes can outperform the most powerful alternatives.
- Easy to implement, fast and accurate method of prediction.
- Can work with multiclass prediction problems.
- It performs well in text classification problems.

Disadvantages:

- Independence of features does not hold: The fundamental Naive Bayes assumption is that each feature makes an independent and equal contribution to the outcome. However, this condition is not met most of the time.

- 2. Bad estimator: Probability outputs from predict proba are not to be taken too seriously.

```
[16] from sklearn.metrics import classification_report
     print(classification_report(y_test,y_pred_lemma_cv))
```

```
                precision    recall  f1-score   support

             0       0.64      0.63      0.63      7999
             1       0.56      0.72      0.63      7500
             2       0.51      0.36      0.42      7000

      accuracy                           0.58     22499
     macro avg       0.57      0.57      0.56     22499
  weighted avg       0.57      0.58      0.57     22499
```

```
from sklearn.metrics import precision_score , recall_score , f1_score
print('precision score:',precision_score(y_test,y_pred_lemma_cv,average='weighted'))
print('recall score:',recall_score(y_test,y_pred_lemma_cv,average='weighted'))
print('f1 score:',f1_score(y_test,y_pred_lemma_cv,average='weighted'))
```

```
precision score: 0.5712692299734554
recall score: 0.5751811191608516
f1 score: 0.5660364757256917
```

We had tried train model using Multinomial NB and it is observed that there is bias in the data and model does not perform as per the expectation.

Hence, we decided to train the model using Decision Tree Classifier.

5.2 Decision Tree Classifier

Given data of attributes together with its classes, a decision tree produces a sequence of rules that can be used to classify the data. The decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like a tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

Advantages:

- Simple, Fast in processing, and effective
- Does well with noisy data and missing data
- Handles numeric and categorical variables
- Automatic Feature selection: Irrelevant features won't affect decision trees

Disadvantages:

- Often biased towards splits or features have a large number of levels
- May not be optimum as modelling some relations on an axis parallel basis is not optimal
- Small changes in training data can result in large changes to the logic
- Large trees can be difficult to interpret

```
[14] from sklearn.metrics import classification_report
     print(classification_report(y_test,y_pred_lemma_cv))
```

```
                precision    recall  f1-score   support

             0       0.63      0.56      0.59      7999
             1       0.58      0.69      0.63      7500
             2       0.55      0.52      0.54      7000

      accuracy                           0.59     22499
     macro avg       0.59      0.59      0.59     22499
  weighted avg       0.59      0.59      0.59     22499
```

```
from sklearn.metrics import precision_score , recall_score , f1_score
print('precision score:',precision_score(y_test,y_pred_lemma_cv,average='weighted'))
print('recall score:',recall_score(y_test,y_pred_lemma_cv,average='weighted'))
print('f1 score:',f1_score(y_test,y_pred_lemma_cv,average='weighted'))
```

```
precision score: 0.5909757952457296
recall score: 0.5896706520289791
f1 score: 0.5879733672840003
```

Decision Tree performed better than MultinomialNB but still there is a bias while training the model.

Hence thereby, we decided to train the dataset on Random Forest Classifier model.

5.3 Random Forest Classifier

Random Forest is a trademarked term for an ensemble of decision trees. In Random Forest, we have a collection of decision trees (so-known as "Forest"). To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

It works in four steps:

1. Select random samples from a given dataset.
2. Construct a decision tree for each sample and get a prediction result from each decision tree.
3. Perform a vote for each predicted result.
4. Select the prediction result with the most votes as the final prediction.

Advantages:

- Reduced error: Random forest is an ensemble of decision trees. For predicting the outcome of a particular row, a random forest takes inputs from all the trees and then predicts the outcome. This ensures that the individual errors of trees are minimized, and overall variance and error are reduced.

- Good Performance on Imbalanced datasets: It can also handle errors in imbalanced data (one class is the majority and another class is the minority)

- Good handling of missing data: It can handle missing data very well. So, if there is a large amount of missing data in your model, it will give good results.

- No problem of overfitting: Random forest considers only a subset of features, and the outcome depends on all the trees. So, there is more generalization and less overfitting

- Useful to extract feature importance (we can use it for feature selection)

Disadvantages:

- Features need to have some predictive power else they won't work.
- Predictions of the trees need to be uncorrelated.
- Appears as Black Box: It is tough to know what is happening. We can at best try different parameters and random seeds to change the outcomes and performance.

```
[16] from sklearn.metrics import classification_report
     print(classification_report(y_test,y_pred_lemma_cv))
```

```
                  precision    recall  f1-score   support

               0       0.68      0.64      0.66      7999
               1       0.61      0.76      0.68      7500
               2       0.63      0.51      0.56      7000

        accuracy                           0.64     22499
       macro avg       0.64      0.64      0.64     22499
    weighted avg       0.64      0.64      0.64     22499
```

```
from sklearn.metrics import precision_score , recall_score , f1_score
print('precision score:',precision_score(y_test,y_pred_lemma_cv,average='weighted'))
print('recall score:',recall_score(y_test,y_pred_lemma_cv,average='weighted'))
print('f1 score:',f1_score(y_test,y_pred_lemma_cv,average='weighted'))
```

```
precision score: 0.6433743320472399
recall score: 0.6410507133650385
f1 score: 0.6373429873599333
```

Random Forest Classifier model performed better than both MultinomialNB as well as Decision Tree Classifier.

Inference: From the above, it can be inferred, for both Accuracy & F-1 Scoring metrics for Random Forest Classifier is good. But Random Forest Classifier shows bias in predicting classes. Hence, we have decided to keep 2 classes instead of 3 for better prediction.

Metrics	MNB	RF	DT	XGB	GB
Precision_Score	0.571	0.643	0.590	0.495	0.506
Recall_Score	0.575	0.641	0.589	0.465	0.477
F1_Score	0.566	0.637	0.587	0.434	0.451

As we are proceeding, considering 2 classes as our target variable we will also be training model on Logistic Regression.

5.4 Logistic Regression

Definition: Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1/0, Yes / No, True / False) given a set of independent variables. To represent the binary/categorical outcome, we use dummy variables.

Logistic regression is a special case of linear regression when the outcome variable is categorical, where we are using a log of odds as the dependent variable. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function.

Logistic regression becomes a classification technique only when a decision threshold is brought into the picture. The setting of the threshold value is a particularly important aspect of Logistic regression and is dependent on the classification problem itself.

The decision for the value of the threshold value is majorly affected by the values of precision and recall. Ideally, we want both precision and recall being 1, but this seldom is the case. In the case of a Precision-Recall trade-off, we use the following arguments to decide upon the threshold.

- **Low Precision/High Recall:** In applications where we want to reduce the number of false negatives without necessarily reducing the

number of false positives, we choose a decision value which has a low value of Precision or a high value of Recall. For example: In the case of the medical field, employees churn or attrition rate, etc false negative would be costlier, thus the focus should be on reducing the number of false negatives in comparison to false positives.

- **High Precision/Low Recall:** In applications where we want to reduce the number of false positives without necessarily reducing the number of false negatives, we choose a decision value that has a high value of Precision or a low value of Recall. For example: In the case of email spam detection, cybersecurity space, etc, false-positive would be costlier, thus the focus should be on reducing the number of false-positive in comparison to false negatives.

Advantages:

- Effective and simple to implement.
- It is most useful for understanding the influence of several independent variables on a single outcome variable.
- Does not require input features to be scaled (can work with scaled features too but doesn't require scaling).

Disadvantages:

- Poor performance on non-linear data (image data e.g.)
- Poor performance with irrelevant and highly correlated features.
- Not a powerful algorithm and can be easily outperformed by other algorithms.

```
[ ] from sklearn.metrics import classification_report
    print(classification_report(y_test,y_pred_lemma_cv))
```

```
              precision    recall  f1-score   support

           0       0.69      0.74      0.71     10688
           1       0.75      0.69      0.72     11811

    accuracy                           0.72     22499
   macro avg       0.72      0.72      0.72     22499
weighted avg       0.72      0.72      0.72     22499
```

```
from sklearn.metrics import precision_score , recall_score , f1_score
print('precision score:',precision_score(y_test,y_pred_youden))
print('recall score:',recall_score(y_test,y_pred_youden))
print('f1 score:',f1_score(y_test,y_pred_youden))
```

```
precision score: 0.738747553816047
recall score: 0.7031580729828126
f1 score: 0.7205135990977312
```

Using Logistic Regression model on 2 classes the f1-score, recall as well as the precision score are nearer to each other and the bias error is rectified in this model here.

5.5 2-Class Decision Tree Classifier

The results after training Decision Tree Classifier on 2-class is provided below:

```
from sklearn.tree import DecisionTreeClassifier
nl_model = DecisionTreeClassifier(random_state=0).fit(X_train, y_train)

y_pred_lemma_cv=nl_model.predict(X_test)
```

```
] from sklearn.metrics import accuracy_score,confusion_matrix
  print('accuracy score:',accuracy_score(y_test,y_pred_lemma_cv))

  accuracy score: 0.7166985199342193
```

```
] from sklearn.metrics import classification_report
  print(classification_report(y_test,y_pred_lemma_cv))

              precision    recall  f1-score   support

           0       0.68      0.75      0.72     10688
           1       0.75      0.68      0.72     11811

    accuracy                           0.72     22499
   macro avg       0.72      0.72      0.72     22499
weighted avg       0.72      0.72      0.72     22499
```

```
] from sklearn.metrics import precision_score , recall_score , f1_score
  print('precision score:',precision_score(y_test,y_pred_lemma_cv))
  print('recall score:',recall_score(y_test,y_pred_lemma_cv))
  print('f1  score:',f1_score(y_test,y_pred_lemma_cv))

  precision score: 0.7541366738337851
  recall score: 0.683007366014732
  f1  score: 0.7168118002488003
```

In Decision Tree precision is slightly higher than recall but in Logistic Regression both precision score as well as F1score is nearer to each other.

5.6 2-Class Multinomial NB

The results after training Multinomial NB on 2-class is provided below:

```
[12] # Training model using Naive bayes classifier

     from sklearn.naive_bayes import MultinomialNB
     nl_model = MultinomialNB().fit(X_train, y_train)

     y_pred_lemma_cv=nl_model.predict(X_test)
```

```
[13] from sklearn.metrics import accuracy_score,confusion_matrix
     print('accuracy score:',accuracy_score(y_test,y_pred_lemma_cv))

     accuracy score: 0.7173652162318326
```

```
     from sklearn.metrics import classification_report
     print(classification_report(y_test,y_pred_lemma_cv))
```

```
                  precision    recall  f1-score   support

               0       0.69      0.74      0.71     10688
               1       0.75      0.69      0.72     11811

        accuracy                           0.72     22499
       macro avg       0.72      0.72      0.72     22499
    weighted avg       0.72      0.72      0.72     22499
```

Multinomial NB is also performing similar to Decision Tree; hence we have decided to train the model on Random Forest Classifier.

5.7 2-Class Random Forest Classifier

The results after training Random Forest Classifier on 2-class is provided below:

```
[15] from sklearn.ensemble import RandomForestClassifier
     clf3=  RandomForestClassifier(random_state=0)
```

```
[16] spam_detect_model_rf = clf3.fit(X_train, y_train)

     y_pred_lemma_cv=spam_detect_model_rf.predict(X_test)
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred_lemma_cv))
```

```
              precision    recall  f1-score   support

           0       0.72      0.79      0.75     10688
           1       0.79      0.73      0.76     11811

    accuracy                           0.76     22499
   macro avg       0.76      0.76      0.76     22499
weighted avg       0.76      0.76      0.76     22499
```

Metrics	MNB	RF	DT	XGB	GB	LR
Precision_Score	0.749	0.791	0.754	0.714	0.713	0.738
Recall_Score	0.693	0.726	0.683	0.428	0.434	0.703
F1_Score	0.720	0.757	0.716	0.535	0.539	0.720

Inference: From the above, it can be inferred, Random Forest Classifier gives the best values for F-1 Scoring metrics. Thus, choosing the Random Forest Classifier for evaluating our testing model.

Note: The above results were obtained using the default parameters for all models.

Chapter 6

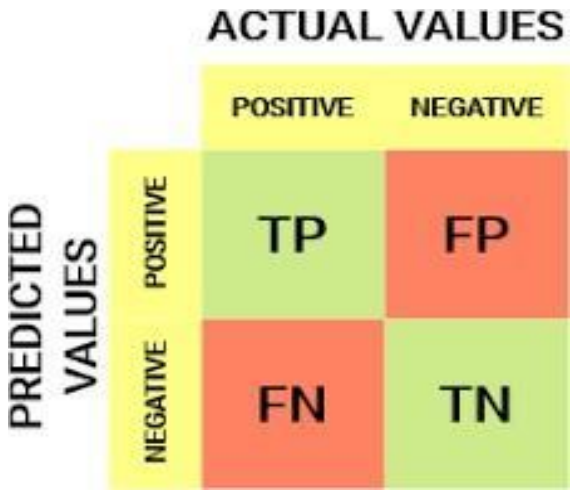
Important Model Evaluation Metrics

Evaluating a model is a core part of building an effective machine learning model. There are several evaluation metrics for classification problems like F-1, Precision, Recall, Accuracy, and ROC-AUC Scoring. Different evaluation metrics are used for different kinds of problems. Evaluation metrics explain the performance of a model. An important aspect of evaluation metrics is their capability to discriminate among model results.

To get a proper understanding of F-1, Precision, Recall, and Accuracy, it's very important to understand the concept of the **Confusion Matrix.**

6.1 Confusion Matrix

Definition: A confusion matrix is an N X N matrix, where N is the number of classes being predicted. For the problem in hand, we have N=2, and hence we get a 2 X 2 matrix. Here are a few terms, we need to remember for a confusion matrix:



Let's decipher the matrix:

- The target variable has two values: Positive or Negative.
- The columns represent the actual values of the target variable.
- The rows represent the predicted values of the target variable.

Understanding True Positive, True Negative, False Positive, and False Negative in a Confusion Matrix True.

Positive (TP):

- The predicted value matches the actual value.
- The actual value was positive, and the model predicted a positive value.

True Negative (TN)

- The predicted value matches the actual value.
- The actual value was negative and the model predicted a negative value.

False Positive (FP) – Type 1 error

- The predicted value was falsely predicted.
- The actual value was negative, but the model predicted a positive value

False Negative (FN) – Type 2 error

- The predicted value was falsely predicted.
- The actual value was positive, but the model predicted a negative value.

Need for Confusion Matrix

Most of the real-life datasets are imbalanced, and thus blindly relying on the most commonly used metrics which is Accuracy is not sufficient, as it might be misleading. Below are the few important terms we should know before deciding the evaluation metrics.

1. **Accuracy:** The accuracy of a machine learning classification algorithm is one way to measure how often the algorithm classifies a data point correctly. Accuracy is the number of correctly predicted data points out of all the data points. More formally, it is defined as the number of true positives and true negatives divided by the number of true positives, true negatives, false positives, and
   false negatives.

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

2. **Recall:** Recall tells us how many of the actual positive cases we were able to predict correctly with our model. To simplify this, when the actual value is positive how often his prediction corrects. Also known as Sensitivity.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

3. **Precision:** Precision tells us how many of the correctly predicted cases turned out to be positive. To understand this in simpler terms, when a positive value is predicted, how often is prediction correct.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

4. **F-1 Score:** In practice, when we try to increase the precision of our model, the recall goes down, and vice-versa. The F1-score captures both the trends in a single value. F1-score is a harmonic mean of Precision and Recall, and so it gives a combined idea about these two metrics. It is maximum when Precision is equal to Recall.

The interpretability of the F1-score is poor. This means that we don't know what our classifier is maximizing – precision or recall. So, we use it in combination with other evaluation metrics which gives us a complete picture of the result.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Chapter 7

Hyperparameter Tuning

7.1Need for hyperparameter tuning

Hyperparameters control the behaviour of the algorithm that is used for modelling. Hyperparameters are passed in the arguments during the initialization of the algorithm.

The hyperparameters that are mostly used for RandomForest are n estimators.

Hyperparameters help us limit overfitting or underfitting of the model. They also aid in reducing the time taken to execute.

The types of hyperparameters for boosting.

n_estimators: ***int, default=100***
The number of trees in the forest.

criterion ***{"gini", "entropy"}, default ="gini"***
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Note: this parameter is tree-specific.

max_depth: ***int, default=None***
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split: ***int or float, default=2***
The minimum number of samples required to split an internal node:

- If int, then consider min_samples_split as the minimum number.

- If float, then min_samples_split is a fraction and ceil (min_samples_split * n_samples) are the minimum number of samples for each split.

min_samples_leaf: ***int or float, default=1***

The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

- If int, then consider min_samples_leaf as the minimum number.
- If float, then min_samples_leaf is a fraction and ceil (min_samples_leaf * n_samples) are the minimum number of samples for each node.

max_features: ***{"auto", "sqrt", "log2"}, int or float, default=" auto"***

The number of features to consider when looking for the best split:

- If int, then consider max_features feature at each split.
- If float, then max_features is a fraction and round (max_features * n_features) features are considered at each split.
- If "auto", then max_features=sqrt(n_features).
- If "sqrt", then max_features=sqrt(n_features) (same as "auto").
- If "log2", then max_features=log2(n_features).
- If None, then max_features=n_features.

Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max_features features.

max_leaf_nodes: ***int, default=None***

Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

7.2 Selection of n_estimators

n_estimators is one of the most important hyperparameters for the extreme bagging model. n_estimators is the number of trees to be used in the forest. Since RandonForest is an ensemble method comprising of creating multiple decision trees, this parameter is used to control the number of trees to be used in the process.

It is important to choose the right number of trees/estimators that can learn about the data very well and thereby reducing the entropy and maximizing the information gain. As the number of trees increase, RandomForest algorithm works in a way that minimizes the error of the preceding tree.

Our approach is because GridSearchCV does not take into account Variance error while computing the best parameters for model tuning, therefore a manual looping is performed **with 400 to 700 n_estimators** to manually check the best bias accuracy and variance error trade-off.

7.3GridSearchCV

Grid search is the process of performing hyperparameter tuning to determine the optimal values for a given model. This is significant as the performance of the entire model is based on the hyperparameter values specified.

The estimator parameter of GridSearchCV requires the models that are used for the hyperparameter tuning process. For this example, let's use the kernel of the RandomForest algorithm. The param_grid parameter requires a list of parameters and the range of values for each parameter of the specified estimator. The most significant parameters required when working with the kernel of the RandomForest model are n_estimators. A list of values to choose from should be given to each hyperparameter of the model. You can change these values and experiment more to see which value ranges give better performance. A cross-validation process is performed in order to determine the hyperparameter value set which provides the best accuracy levels.

After tunning the model, we got the best **F1 score at n_estimator at 600.**

Chapter 8

Applying Tuned Model on the Training Data

Passing the training data

After selecting the optimum hyperparameters using GridSearchCV, the tuned model is ready to be subjected to the 50,000 records of training data. The training data comprises data with dynamic sentiments, this helps in training the model in different verticals to predict the sentiments.

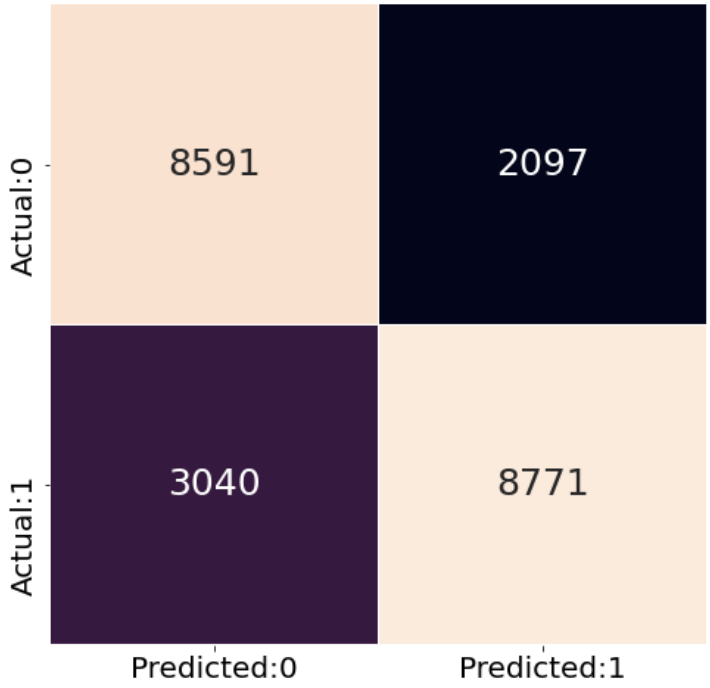
Let us look at the model's performance with the training data.

	precision	recall	f1-score	support
0	0.74	0.80	0.77	10688
1	0.81	0.74	0.77	11811
accuracy			0.77	22499
macro avg	0.77	0.77	0.77	22499
weighted avg	0.77	0.77	0.77	22499

```
from sklearn.metrics import precision_score , recall_score , f1_score
print('precision score:',precision_score(y_test,y_pred_lemma_cv))
print('recall score:',recall_score(y_test,y_pred_lemma_cv))
print('f1  score:',f1_score(y_test,y_pred_lemma_cv))
```

```
precision score: 0.8070482149429518
recall score: 0.7426128185589704
f1  score: 0.7734908946602583
```

After Hyperparameter tunning, these matrices help us to explain precision score, recall score and f1 score.



The above confusion matrix talks about our model's prediction and detection. The matrix says that 8591 are correctly classified as a positive, whereas 2097 are wrongly predicted. 8771 are correctly classified negative whereas 3040 are wrongly predicted. These wrong predictions are because reviews consist of meaningless words. These sentiments are hard for a machine to comprehend and therefore fail to classify them correctly based on a positive or negative review.

CHAPTER 9

Conclusion

Predicting the sentiments in review and ratings for the Sentiment Analysis of Drug Review by Patients which formed different classification models such as Logistical Regression, Decision Tree, Random Forest, Ada Boost, XGB Classifier, Gradient Boosting, Multinomial Naïve Bayes, f-1 scores for all the models are taken and compared.

After comparing all the models, RandomForest Classifier gave the best result in terms of F1_score. The scoring for RandomForest Classifier is then improved with the help of hyperparamter tuning.

A significant improvement was observed after hyperparameter tuning for the RandomForest Classifier Model in order to detect and predict sentiments of the patients. Different n_estimator values were checked. The optimal n_estimator value is 600 which gave the weighted recall and F-1 score as 74% and 77% respectively.

Business Recommendation

The objective is: “Sentiment Analysis of Drug Review by Patients”. Thus, it is inferred that False Negatives (FN) are much costlier than the False Positives (FP). FP only costs a message that is sent to the user whereas FN affects the Drug maker, clinician and patient as directly affecting the ratings.

Based on the patients reviews and ratings, positive and negative sentiments will help drug industry/drug maker to analyses the drugs having best result for specific conditions.