# Review Sentiment Analysis

Aryan Qusyairi – P2225182
Joseph Wong – P2225351
Nay Thuta - P2234191

# CONTENTS:

**1. Problem:**
- What is Sentiment Analysis
- ML Techniques

**2. Data Collection:**
- Our Data Source

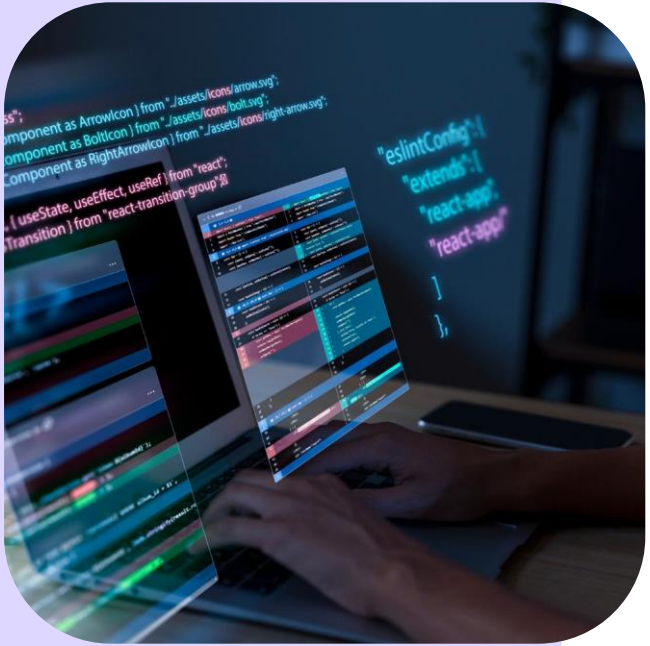
**3. Data Preparation:**
- Transformation, Cleaning and Correction

**4. Exploratory Data Analysis:**
- Our Chosen Approach
- Visualizations

**5. Data Pre-processing:**
- Further transformation, and processing

**6. Our Model:**
- Suitable ML Models
- Model Training

**7. Conclusion:**
- Business Applications
- PowerBI Dashboard




# 01

# Introduction

# INTRODUCTION

## What is Sentiment Analysis

A process of natural language processing and machine learning that involves identifying, extracting, and analyzing subjective information from text. It is used to determine, emotional tone, opinion, or attitude expressed in a piece of text and categorized as positive, negative or neutral.

# PROBLEM

## Why is it an Issue?

**Consumers frequently report their experiences** with drugs and their side effects in online reviews. However, these insights are often **scattered, unstructured, and difficult to quantify.** Without systematic analysis, critical information about adverse effects or patterns may go unnoticed, delaying interventions that could improve patient safety and satisfaction.

# PROBLEM

## Why is it Important?

### Trust and Transparency

Allows the demonstration of attentiveness to customer feedback, while **builds trust among patients and healthcare providers.**

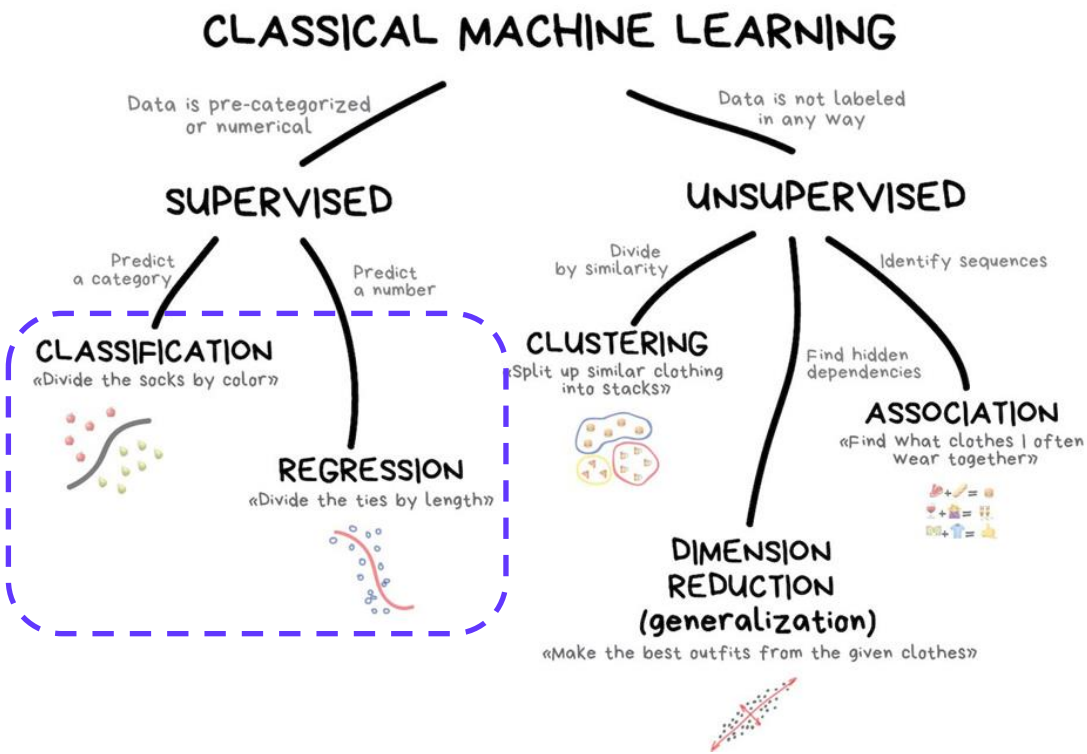
### Customer Support and Engagement

Sentiment Analysis helps businesses tailor customer support responses to **address specific concerns more effectively.**

### Improve Drug Safety

Helps identify adverse events or safety concerns reported, enabling **early detection and response** to potential health risks.

# CHOSEN ML TECHNIQUES



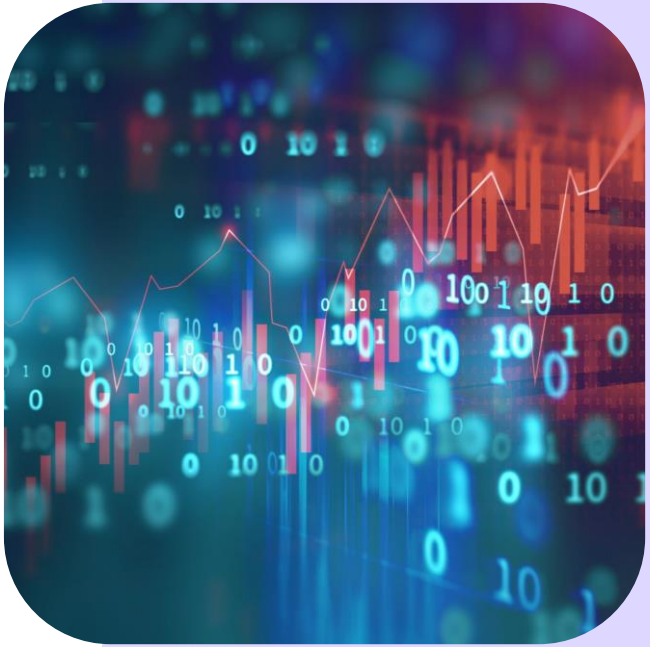
# OUR OBJECTIVE

## Sentiment Analysis in Drug Reviews

We aim to leverage sentiment analysis for identifying emotions behind drug reviews, with the goals of **fostering trust and transparency** by actively addressing patient concerns, **enhancing customer support**, and **improving drug safety** and responding to events and trends promptly.

# 02

# Data Collection



# OUR SOURCES


## Why did we choose Kaggle™?

**Diverse and Extensive Collection**

Offers datasets across a wide range of topics, which include healthcare reviews, social media and more.
Includes datasets for both structured and unstructured data.

**Accessibility**

Most datasets are freely available for download and usage, making it ideal for students like us.
Datasets are provided in common formats like CSV, JSON, or image files.

# CHOSEN DATASET

## Drug Review Dataset by Mohamed Abdelwahab Ali


# 03

# Data Preparation

# OVERVIEW

```
[10]: import pandas as pd

      df = pd.read_csv('drug_review.csv', sep=',')
      #dft = pd.read_csv('drugsComTest_raw.tsv', sep='\t')
```

Length of the review by word count

```
[11]: df.head()
```

Shows top 5 entries

[11]:

| | Unnamed: 0 | patient_id | drugName | condition | review | rating | date | usefulCount | review_length |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 191114 | Campral | alcohol dependence | "sober a year 8-25-11. god, aa and campral hav... | 10.0 | September 3, 2011 | 33 | 41 |
| 1 | 1 | 142693 | Levonorgestrel | birth control | "i've been on birth control for a while now du... | 4.0 | August 9, 2017 | 3 | 140 |
| 2 | 2 | 71561 | Vraylar | bipolar disorde | "hi, this is an updated experience. \r\r\n\r\r... | 8.0 | August 16, 2016 | 12 | 131 |
| 3 | 3 | 25765 | Ethinyl estradiol / norelgestromin | birth control | " have been on the ortho evra patch for just ... | 8.0 | September 15, 2013 | 16 | 138 |
| 4 | 4 | 1[illegible]43 | Etanercept | psoriasis | i have been on enbrel for 7 years and i have ... | 9.0 | August 5, 2010 | 9 | 65 |

Type of Drug

Patient's Condition

Patient's Review on Drugs

Patient's Ratings

Amount of People that find the review useful

# OVERVIEW

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27703 entries, 0 to 27702
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Unnamed: 0     27703 non-null  int64  
 1   patient_id     27703 non-null  int64  
 2   drugName       27703 non-null  object 
 3   condition      27703 non-null  object 
 4   review         27703 non-null  object 
 5   rating         27703 non-null  float64
 6   date           27703 non-null  object 
 7   usefulCount    27703 non-null  int64  
 8   review_length  27703 non-null  int64  
dtypes: float64(1), int64(4), object(4)
memory usage: 1.9+ MB
```

A mix of **Integers, Floats and Objects** within the columns, but a majority are **Integers and Objects**

# OVERVIEW

```
df.isnull().sum()
```

```
Unnamed: 0          0
patient_id          0
drugName            0
condition           0
review              0
rating              0
date                0
usefulCount         0
review_length       0
dtype: int64
```

```
print ("Rows     : " ,df.shape[0])
print ("Columns  : " ,df.shape[1])
print ("\nFeatures : \n" ,df.columns.tolist())
print ("\nMissing values :  ", df.isnull().sum().values.sum())
print ("\nUnique values :  \n",df
       .nunique())
```

```
Rows     :  27703
Columns  :  9

Features :
 ['Unnamed: 0', 'patient_id', 'drugName', 'condition', 'review', 'rating', 'date', 'usefulCount', 'review_length']

Missing values :   0

Unique values :
 Unnamed: 0       10000
patient_id        27703
drugName           2014
condition           579
review            26054
rating               10
date               3491
usefulCount         296
review_length       186
dtype: int64
```

Out of the chosen dataset, there are **no missing values, columns or rows.**

# CLEANING OF DATA

```
print ("Rows     : " ,df.shape[0])
print ("Columns  : " ,df.shape[1])
print ("\nFeatures : \n" ,df.columns.tolist())
print ("\nMissing values :  ", df.isnull().sum().values.sum())
print ("\nUnique values :  \n",df
       .nunique())
```

```
Rows     :  27703
Columns  :  9

Features :
 ['Unnamed: 0', 'patient_id', 'drugName', 'condition', 'review', 'rating', 'date', 'usefulCount', 'review_length']

Missing values :   0

Unique values :
 Unnamed: 0       10000
patient_id       27703
drugName          2014
condition          579
review           26054
rating              10
date              3491
usefulCount        296
review_length      186
dtype: int64
```

Cleans and processes Dataframe (df):

1. Removes unnecessary columns like Unnamed: 0
2. Eliminates Duplicate rows and rows with missing data
3. Removes rows where drugs are not defined.

# CLEANING OF DATA

```
print("Unique values for drugName: ",)
print(df['drugName'].unique().tolist())

print("\nUnique values for condition: ",)
print(df['condition'].unique().tolist())
```

Displays all unique drug names in the drugName column.

```
Unique values for drugName:
['Campral', 'Levonorgestrel', 'Vraylar', 'Ethinyl estradiol / norelgestromin', 'Etanercept', 'NuvaRing', 'Aubra', 'Acyclovir', 'Gabapentin', 'Ethinyl e
stradiol / norgestimate', 'Viibryd', 'Trazodone', 'Fentanyl', 'Saxenda', 'Oxymorphone', 'Depo-Provera', 'Montelukast', 'Metronidazole', 'Skyla', 'Adipe
x-P', 'Amitriptyline', 'Exemestane', 'Lamotrigine', 'Prednisone', 'Venlafaxine', 'Ketorolac', 'Valium', 'Etonogestrel', 'Ciprofloxacin / dexamethason
e', 'Twynsta', 'Risperidone', 'Pentosan polysulfate sodium', 'Modafinil', 'Nucynta', 'Atripla', 'Cytoxan', 'Nexplanon', 'Abilify', 'Ethinyl estradiol /
levonorgestrel', 'Nebivolol', 'Lyrica', 'Wellbutrin', 'Pristiq', 'Prozac', 'Alfuzosin', 'Linzess', 'ParaGard', 'Brethine', 'Lutera', 'Diclofenac', 'Neu
rontin', 'Suprep Bowel Prep Kit', 'Ivermectin', 'Tioconazole', 'Junel 1.5 / 30', 'Phentermine / topiramate', 'Soma', 'ella', 'Victoza', 'Bupropion', 'L
ithium', 'Zoloft', 'Strattera', 'Theo-24', 'Ethinyl estradiol / norethindrone', 'Sprintec', 'Lorcaserin', 'Liraglutide', 'Sofosbuvir', 'Etodolac', 'Ase
napine', 'Tri-Sprintec', 'Donepezil', 'Drospirenone / ethinyl estradiol', 'Eucrisa', 'Macrodantin', 'Aftera', 'Ambien CR', 'Benzoyl peroxide / clindamy
cin', 'Junel Fe 1 / 20', 'Amino acids', 'Diazepam', 'Copper', 'Microgestin Fe 1 / 20', 'Hydroxyzine', 'Aubagio', 'Monistat 7-Day Combination Pack', 'De
plin', 'Mirena', 'Belviq', 'Epclusa', 'Iloperidone', 'Amoxicillin', 'Paroxetine', 'Selenium sulfide', 'Invokana', 'Enbrel', 'Bontril Slow Release', 'Vy
vanse', 'Indomethacin', 'Implanon', 'Symbicort', 'Exenatide', 'Fluocinolone', 'Amphetamine / dextroamphetamine', 'Generess Fe', 'Lorazepam', 'Desvenlaf
axine', 'Trintellix', 'Dicyclomine', 'Trimethoprim', 'Imiquimod', 'Lortab', 'Ciprofloxacin', 'Yaz', 'Ortho Evra', 'Esomeprazole', 'Seasonique', 'Pregab
alin', 'Mesalamine', 'Valtrex', 'Tramadol', 'Chlordiazepoxide', 'Ezetimibe', 'Mirapex', 'Hypotears', 'Finacea', 'Mononessa', 'Reglan', 'Carbidopa / lev
odopa', 'Armour Thyroid', 'Azithromycin', 'Transderm-Scop', 'Buprenorphine', 'Tretinoin', 'Adapalene', 'Escitalopram', 'Testosterone', 'ProAir RespiCli
ck', 'Fluoxetine', 'Dexedrine', 'Aluminum chloride hexahydrate', 'Teriflunomide', 'Medroxyprogesterone', 'Mobic', 'Latuda', 'Vortioxetine', 'Natalizuma
b', 'Topiramate', 'Sulfamethoxazole / trimethoprim', 'Humira', 'Nitrofurantoin', 'Liletta', 'Risperdal', 'Beyaz', 'Rituximab', 'Biotin', 'Belbuca', 'Ce
fdinir', 'Plan B', 'Phentermine', 'Tirosint', 'Elmiron', 'Colchicine', 'Acetaminophen / hydrocodone', 'Alosetron', 'Nicotrol Inhaler', 'Acetaminophen /
```

# CLEANING OF DATA

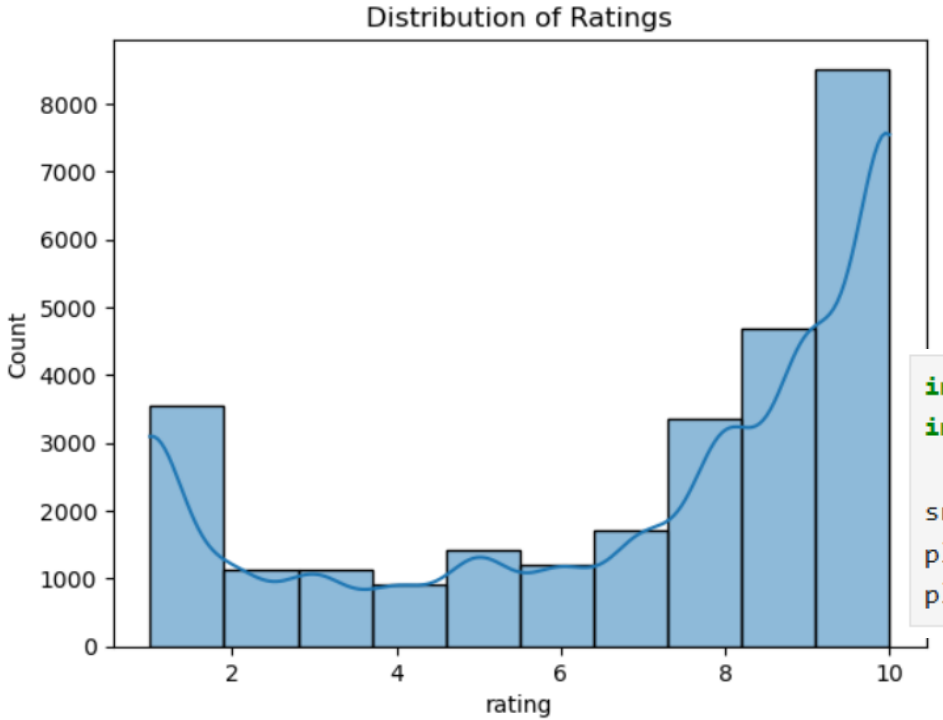
```
#print(df.loc[df['condition'] == '0</span> users found this comment helpful.'])
#df_new = df[df['condition'] != ("0</span> users found this comment helpful.") ]
#df_new = df[~df['condition'].isin(["0</span> users found this comment helpful.", "1</span> users found this comment helpful."])]
#df.drop(["0</span> users found this comment helpful."])
df.drop(df[df['condition'].str.contains("</span> users found this comment helpful.", na=False)].index, inplace=True)
#dft.drop(df[df['condition'].str.contains("</span> users found this comment helpful.", na=False)].index, inplace=True)

print ("Rows     : " ,df.shape[0])
print ("Columns  : " ,df.shape[1])
```

```
Rows     :  27551
Columns  :  9
```

Removes rows that has the "user found this comment helpful" string pattern.

# 04 Exploratory Data Analysis

# DISTRIBUTION GRAPHS

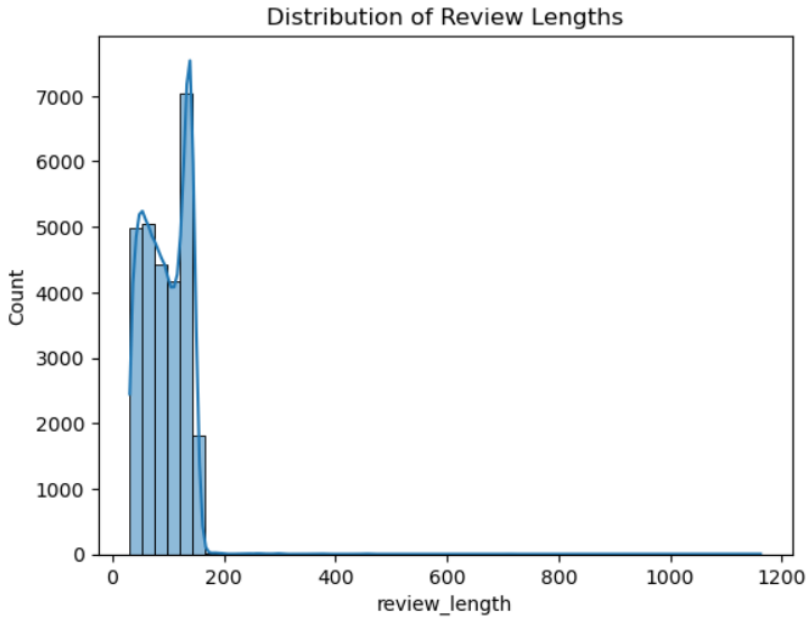



Significant number of Positive (7-10) ratings
Marginally lower number of Negative (0-3) ratings
Significantly low number of Neutral (4-6) ratings.

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.histplot(df['rating'], bins=10, kde=True)
plt.title('Distribution of Ratings')
plt.show()
```

# DISTRIBUTION GRAPHS

```
df['review_length'] = df['review'].apply(lambda x: len(str(x).split()))
sns.histplot(df['review_length'], bins=50, kde=True)
plt.title('Distribution of Review Lengths')
plt.show()
```




Most reviews are of 7000 characters, while a majority are below 5000 words.

# CORRELATION MATRIX

```
correlation_matrix = df[['rating','review_length']].corr()

print("Correlation Matrix: rating vs review_length")
print(correlation_matrix)

correlation_matrix = df[['rating','usefulCount']].corr()

print("\nCorrelation Matrix: rating vs usefulCount")
print(correlation_matrix)
```

```
Correlation Matrix: rating vs review_length
                rating  review_length
rating         1.000000       0.032928
review_length  0.032928       1.000000

Correlation Matrix: rating vs usefulCount
               rating  usefulCount
rating       1.000000     0.240712
usefulCount  0.240712     1.000000
```

A low correlation between ratings and review lengths

=

Longer reviews are loosely related with higher ratings

A weak positive correlation between ratings and useful count

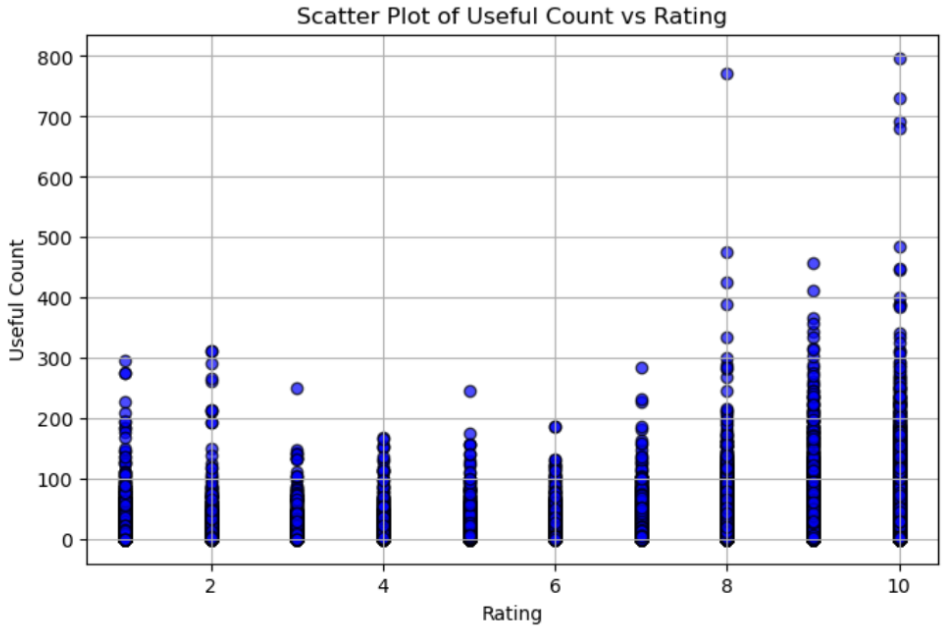
=

Useful reviews are slightly related with higher ratings

# DISTRIBUTION GRAPHS

Relates the rating of the reviews to the amount of people that found the review useful

```
plt.figure(figsize=(8, 5))
plt.scatter(df['rating'], df['usefulCount'], alpha=0.7, color='blue', edgecolors='k')

plt.title("Scatter Plot of Useful Count vs Rating")
plt.xlabel("Rating")
plt.ylabel("Useful Count")
plt.grid(True)
plt.show()
```



# RANDOM SAMPLING

```python
# Randomly sample 100 reviews for each rating
samples_per_rating = {}
for rating in range(1, 11):  # Assuming ratings are 1 to 10
    samples_per_rating[rating] = df[df['rating'] == rating]['review'].sample(100, random_state=100).tolist()

# Create a DataFrame for sampled reviews
sampled_reviews = pd.DataFrame(
    [(rating, review) for rating, reviews in samples_per_rating.items() for review in reviews],
    columns=['rating', 'review']
)
```

Random sampling of 100 reviews where they would then be stored in the sample_per_rating dictionary

# RANDOM SAMPLING

```python
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# Initialize the sentiment analyzer
analyzer = SentimentIntensityAnalyzer()

# Analyze sentiment for the sampled reviews
def analyze_sentiment(review):
    score = analyzer.polarity_scores(review)['compound']

    # Adjust thresholds for drug review context
    if score >= 0.05:
        return 'Positive'
    elif -0.05 < score < 0.05:
        return 'Neutral'
    else:
        return 'Negative'

# Apply sentiment analysis
sampled_reviews['sentiment'] = sampled_reviews['review'].apply(analyze_sentiment)
```

```
Sample reviews for rating = 1
   sentiment  \
0   Negative
1   Negative
2   Positive
3    Neutral
4   Negative
..       ...
95  Negative
96  Negative
97  Negative
98  Negative
99  Negative
```

Calculates compound sentiment score using VADER. Classifies the review as Positive, Neutral, or Negative based on thresholds

# RANDOM SAMPLING

```python
# Group by rating and sentiment
sentiment_summary = sampled_reviews.groupby(['rating', 'sentiment']).size().unstack(fill_value=0)

print("Sentiment Distribution by Rating:")
print(sentiment_summary)
```

```
Sentiment Distribution by Rating:
sentiment  Negative  Neutral  Positive
rating
1                79        3        18
2                71        4        25
3                71        0        29
4                63        4        33
5                59        5        36
6                49        2        49
7                44        5        51
8                46        4        50
9                34        2        64
10               33        1        66
```

Ratings of reviews correlate with the sentiment distribution closely

# K-MEANS CLUSTERING

```python
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Assuming sampled_reviews and sentiment_summary are already created
# Example sentiment_summary format:
# sentiment_summary = sampled_reviews.groupby(['rating', 'sentiment']).size().unstack(fill_value=0)

# Step 1: Prepare the data
# We'll use the 'Negative', 'Neutral', and 'Positive' columns as features for clustering
X = sentiment_summary[['Negative', 'Neutral', 'Positive']]

# Step 2: Normalize the data (Optional but recommended for K-means)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Apply K-means clustering
kmeans = KMeans(n_clusters=3, random_state=42)  # Assuming we want 3 clusters
sentiment_summary['Cluster'] = kmeans.fit_predict(X_scaled)

# Display the results
print("Sentiment Distribution with Clusters:")
print(sentiment_summary)

# Optionally, you can examine the centroids of the clusters
print("\nCluster Centroids:")
print(kmeans.cluster_centers_)
```

Data is standardized using StandardScaler. Allows K-means to work efficiently due to similar features.

Divided data into 3 clusters (positive, neutral, negative). Allows us to find patterns in sentiment distributions.

# K-MEANS CLUSTERING

Sentiment Distribution with Clusters:

| sentiment rating | Negative | Neutral | Positive | Cluster |
|---|---|---|---|---|
| 1 | 79 | 3 | 18 | 2 |
| 2 | 71 | 4 | 25 | 0 |
| 3 | 71 | 0 | 29 | 2 |
| 4 | 63 | 4 | 33 | 0 |
| 5 | 59 | 5 | 36 | 0 |
| 6 | 49 | 2 | 49 | 1 |
| 7 | 44 | 5 | 51 | 0 |
| 8 | 46 | 4 | 50 | 0 |
| 9 | 34 | 2 | 64 | 1 |
| 10 | 33 | 1 | 66 | 1 |

Cluster Centroids:

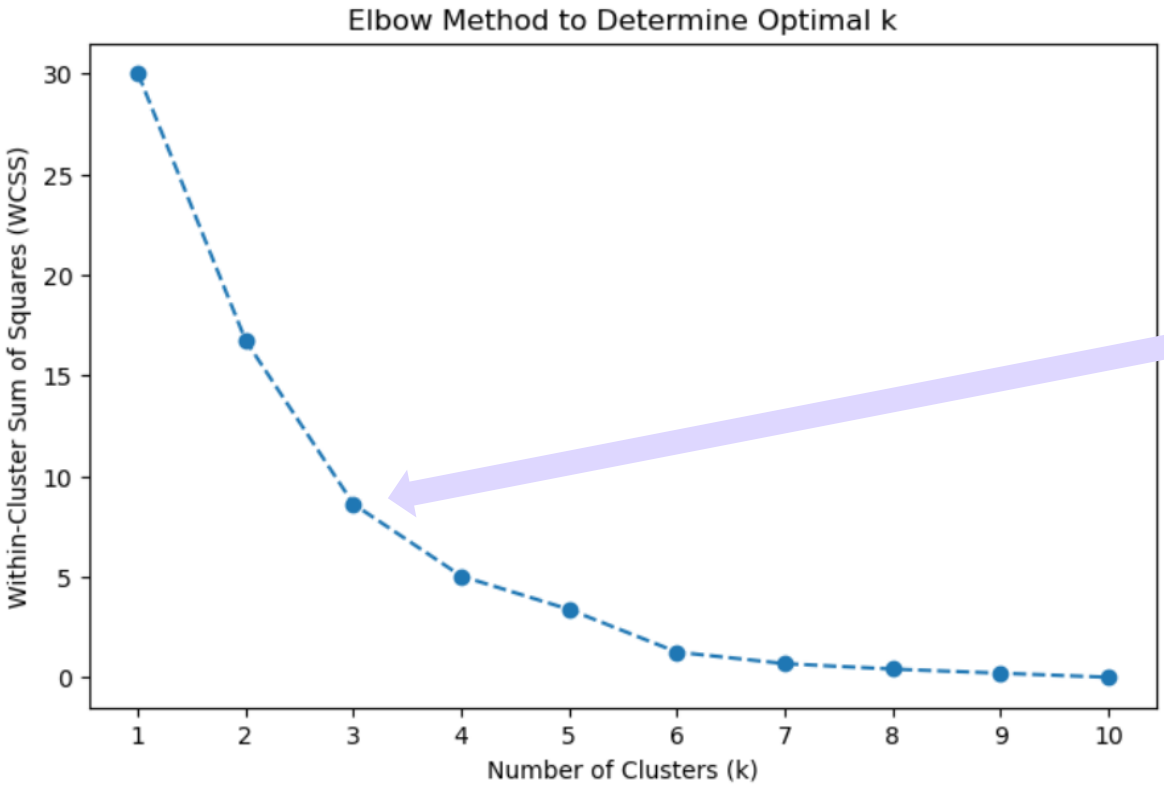
```
[[ 0.1113492   0.86824314 -0.19990018]
 [-1.06327569 -0.82689823  1.13276768]
 [ 1.31654054 -0.93026051 -1.19940107]]
```

# ELBOW METHOD

```python
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Step 1: Prepare a range of k values
wcss = []  # Store within-cluster sum of squares for each k
k_values = range(1, 11)  # Try k from 1 to 10

# Step 2: Compute WCSS for each k
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)  # Inertia is the WCSS

# Step 3: Plot the elbow curve
plt.figure(figsize=(8, 5))
plt.plot(k_values, wcss, marker='o', linestyle='--')
plt.title('Elbow Method to Determine Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.xticks(k_values)
plt.show()
```

Allows visual identification of the optimal number of clusters by finding the "elbow" in the curve.

# ELBOW METHOD




Optimal number of cluster is 3

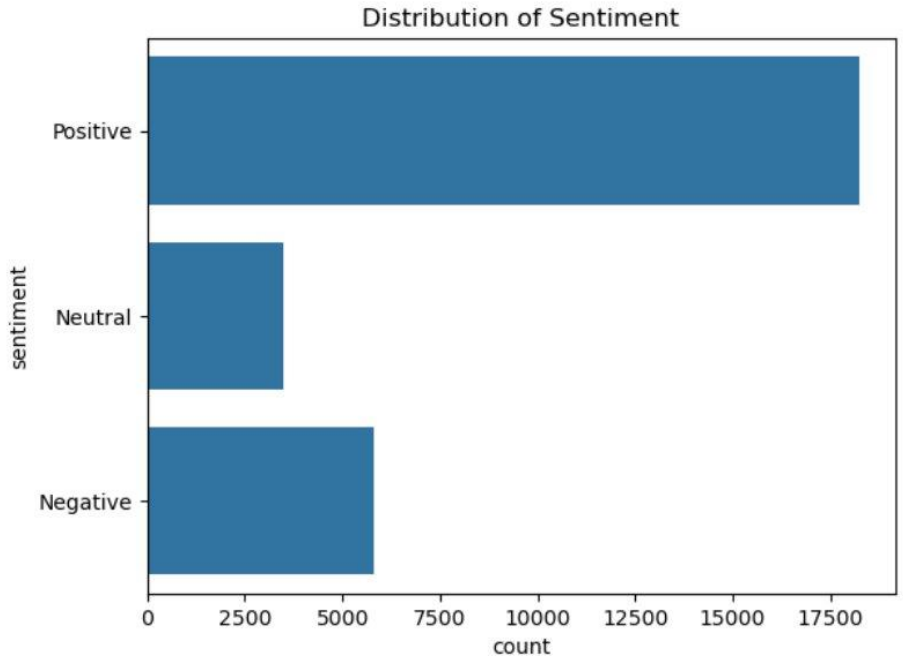
# FEATURE ENGINEERING

```python
def label_sentiment(rating):
    if rating >= 7:
        return 'Positive'
    elif rating >= 4:
        return 'Neutral'
    else:
        return 'Negative'

df['sentiment'] = df['rating'].apply(label_sentiment)
print(df['sentiment'].value_counts())

sns.countplot(df['sentiment'])
plt.title('Distribution of Sentiment')
plt.show()
```

Classification of numerical ratings into 3 sentiment categories.
Divides the ratings based on predefined thresholds, typically reflecting sentiment strength

# DISTRIBUTION GRAPHS

```
sentiment
Positive    18250
Negative     5799
Neutral      3502
Name: count, dtype: int64
```




The dataset shows a positive bias, emphasizing the need for a model that generalizes well despite the imbalance.

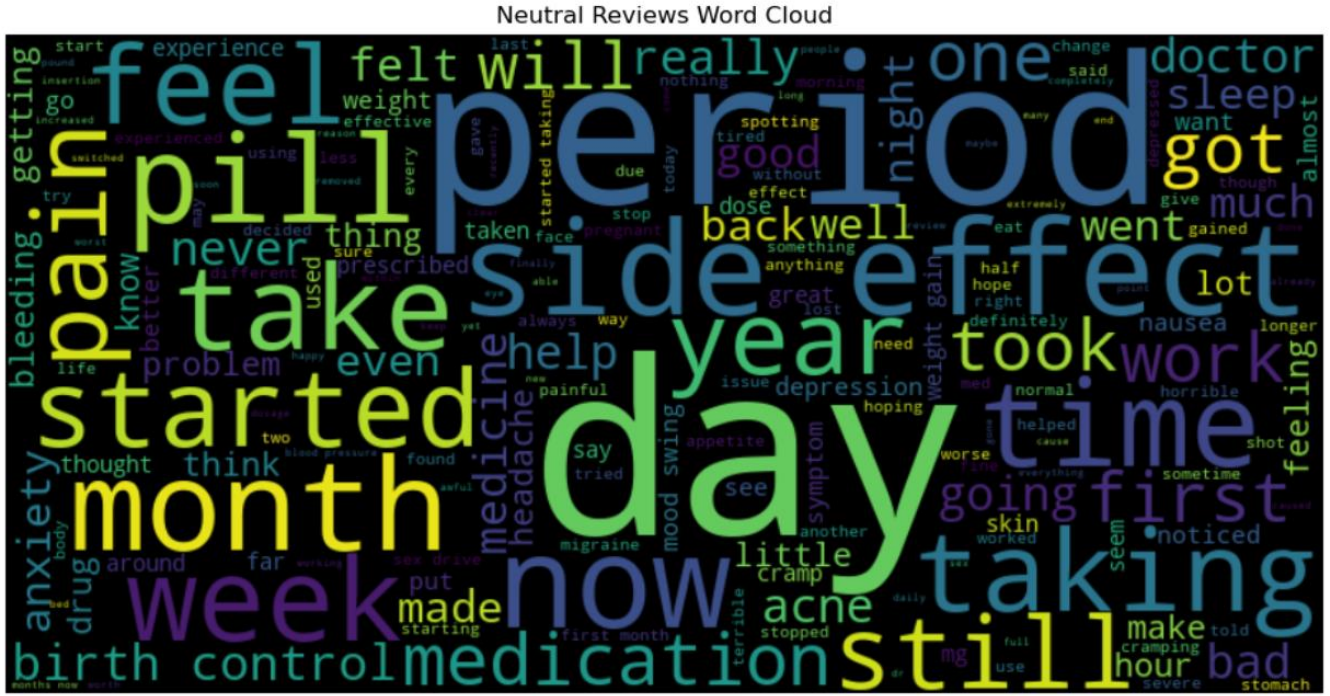
# WORD CLOUD GRAPH

```
from wordcloud import WordCloud

positive_reviews = ' '.join(df[df['sentiment'] == 'Positive']['review'].dropna())
neutral_reviews = ' '.join(df[df['sentiment'] == 'Neutral']['review'].dropna())
negative_reviews = ' '.join(df[df['sentiment'] == 'Negative']['review'].dropna())

wordcloud_pos = WordCloud(width=800, height=400).generate(positive_reviews)
wordcloud_neu = WordCloud(width=800, height=400).generate(neutral_reviews)
wordcloud_neg = WordCloud(width=800, height=400).generate(negative_reviews)

plt.figure(figsize=(12, 6))
plt.imshow(wordcloud_pos, interpolation='bilinear')
plt.title('Positive Reviews Word Cloud')
plt.axis('off')
plt.show()

plt.figure(figsize=(12, 6))
plt.imshow(wordcloud_neu, interpolation='bilinear')
plt.title('Neutral Reviews Word Cloud')
plt.axis('off')
plt.show()

plt.figure(figsize=(12, 6))
plt.imshow(wordcloud_neg, interpolation='bilinear')
plt.title('Negative Reviews Word Cloud')
plt.axis('off')
plt.show()
```

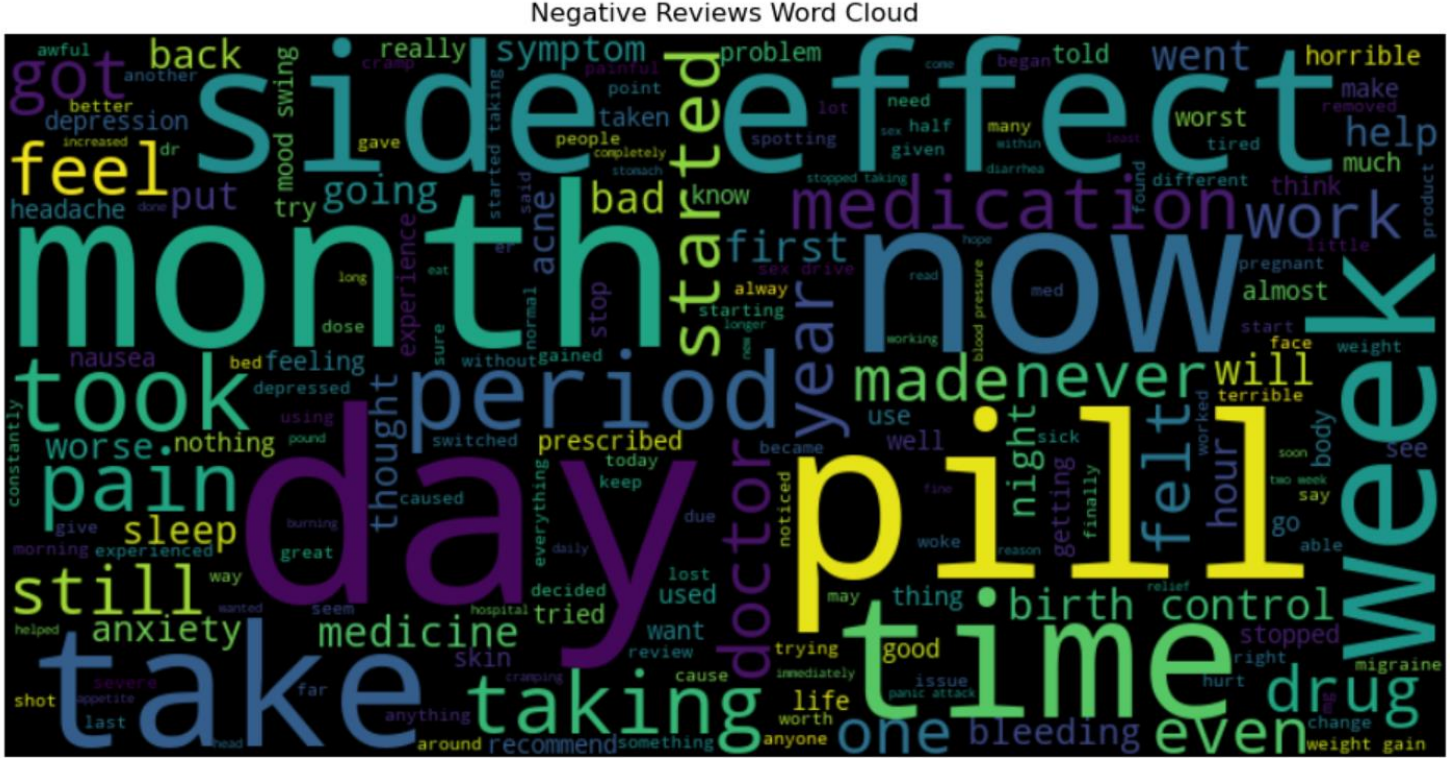
For each sentiment, the review column is accessed and any missing reviews are removed

# WORD CLOUD GRAPH

# WORD CLOUD GRAPH



# WORD CLOUD GRAPH



# WORD CLOUD GRAPH

Overall, the word clouds failed to provide meaningful insights into key terms, as it was dominated by neutral words such as "day," "now," "take," "month," "pill," and "side effect" across all three sentiment ratings. This highlights the need for **more thorough preprocessing** of the review data and the implementation of an effective model that applies **appropriate weights to sentiment-influencing words** to derive accurate sentiment insights.


# 05

# PRE-PROCESSING

# PRE-PROCESSING

```
# Clean text
stop_words = set(nltk.corpus.stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'\s+', ' ', text)  # Remove extra spaces
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    words = [lemmatizer.lemmatize(word) for word in text.split() if word not in stop_words]
    return ' '.join(words)
```

Defines English stop words and initializes lemmatizer to reduce the words to their base form.

Converts all texts to lowercase, and removes extra spaces and punctuation.

Removes stop words and executes the lemmatizer.

# PRE-PROCESSING

```python
# Categorize sentiment
def categorize_sentiment_from_rating(rating):
    if rating >= 7:
        return 'positive'
    elif rating >= 4:
        return 'neutral'
    else:
        return 'negative'

data['true_sentiment'] = data['rating'].apply(categorize_sentiment_from_rating)
# Generate a custom lexicon from dataset reviews
def generate_custom_lexicon(data, min_count=15):
    # Extract unigrams, bigrams, and trigrams
    reviews = data['review'].str.split()
    unigrams = chain.from_iterable(reviews)
    bigrams = chain.from_iterable(zip(review, review[1:]) for review in reviews)
    trigrams = chain.from_iterable(zip(review, review[1:], review[2:]) for review in reviews)
```

Assigns sentiment label based on ratings (Positive, Neutral, Negative).

Unigrams: Individual words
Bigrams: Consecutive pairs of words
Trigrams: Consecutive triplets of words.

# PRE-PROCESSING

```
# Count frequencies
unigram_counts = Counter(unigrams)
bigram_counts = Counter(bigrams)
trigram_counts = Counter(trigrams)

# Filter n-grams by frequency
filtered_unigrams = {word: count for word, count in unigram_counts.items() if count >= min_count}
filtered_bigrams = {" ".join(bigram): count for bigram, count in bigram_counts.items() if count >= min_count}
filtered_trigrams = {" ".join(trigram): count for trigram, count in trigram_counts.items() if count >= min_count}

# Combine all n-grams
all_phrases = {**filtered_unigrams, **filtered_bigrams, **filtered_trigrams}

# Compute average ratings for phrases
phrase_sentiment = {}
for phrase in all_phrases.keys():
    relevant_reviews = data[data['review'].str.contains(phrase)]
    avg_rating = relevant_reviews['rating'].mean() if not relevant_reviews.empty else None
    if avg_rating:
        sentiment_score = (avg_rating - 5) / 5  # Scale sentiment between -1 and +1
        phrase_sentiment[phrase] = sentiment_score

return phrase_sentiment
```

Count the occurrence of each unigram, bigram, and trigram.

Filter and merge all filtered unigrams, bigrams, and trigrams into a single dictionary.

For each phrase in all_phrase, find the associated reviews and calculate average rating.

```
# Create and save custom lexicon
custom_lexicon = generate_custom_lexicon(data)
with open("custom_vader_lexicon.txt", "w") as f:
    for phrase, score in custom_lexicon.items():
        f.write(f"{phrase}\t{score}\n")
```

# PRE-PROCESSING

```
# Add VADER sentiment scores
sia = SentimentIntensityAnalyzer()
sia.lexicon.update(custom_lexicon)

data['vader_compound'] = data['review'].apply(lambda x: sia.polarity_scores(x)['compound'])
data['vader_pos'] = data['review'].apply(lambda x: sia.polarity_scores(x)['pos'])
data['vader_neg'] = data['review'].apply(lambda x: sia.polarity_scores(x)['neg'])
data['vader_neu'] = data['review'].apply(lambda x: sia.polarity_scores(x)['neu'])

# Encode labels
label_encoder = LabelEncoder()
data['encoded_sentiment'] = label_encoder.fit_transform(data['true_sentiment'])

# Split data
train_data, test_data = train_test_split(data, test_size=0.2, stratify=data['encoded_sentiment'], random_state=42)
```

For each phrase in all_phrase, find the associated reviews and calculate average rating.

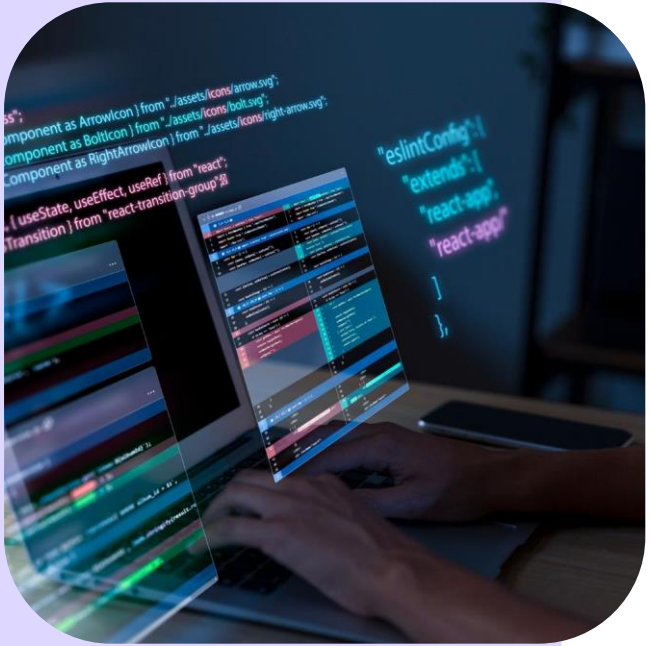
# PRE-PROCESSING

Converts text into numerical representation.
Limits number of words to 20000 most important, based on frequency.

```
# Feature extraction
tfidf = TfidfVectorizer(max_features=20000, ngram_range=(1, 2), stop_words='english')
train_tfidf = tfidf.fit_transform(train_data['review'])
test_tfidf = tfidf.transform(test_data['review'])

# Combine TF-IDF with VADER scores
train_features = hstack([train_tfidf, train_data[['vader_compound', 'vader_pos', 'vader_neg', 'vader_neu']]])
test_features = hstack([test_tfidf, test_data[['vader_compound', 'vader_pos', 'vader_neg', 'vader_neu']]])

# Oversampling with SMOTE
smote = SMOTE(random_state=42)
train_features_balanced, train_labels_balanced = smote.fit_resample(train_features, train_data['encoded_sentiment'])
```

Combines TF-IDF (Term Frequency-Inverse Document Frequency) matrix with VADER (Valence Aware Dictionary and Sentiment Reasoner) scores.

SMOTE (Synthetic Minority Oversampling Technique) ensures reproducibility with fixed random seed.




# 06

# OUR MODEL

# CHOSEN MODEL

```
param_grid_lr = {'C': [0.1, 1, 10], 'solver': ['liblinear']}
log_reg = LogisticRegression(max_iter=10000, class_weight='balanced')
grid_search_lr = GridSearchCV(estimator=log_reg, param_grid=param_grid_lr, cv=5, scoring='f1_weighted', verbose=1)
grid_search_lr.fit(X_train_tfidf, y_train)
best_log_reg = grid_search_lr.best_estimator_
print("Best Parameters (Logistic Regression):", grid_search_lr.best_params_)
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
Best Parameters (Logistic Regression): {'C': 1, 'solver': 'liblinear'}
```

Finding the optimal hyperparameters for a Logistic Regression model implementation

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import classification_report

param_grid_svm = {'C': [0.1, 1, 10], 'max_iter': [5000, 3000]}
svc = LinearSVC(class_weight='balanced')
grid_search_svm = GridSearchCV(estimator=svc, param_grid=param_grid_svm, cv=5, scoring='f1_weighted', verbose=1)
grid_search_svm.fit(X_train_tfidf, y_train)
best_svm = grid_search_svm.best_estimator_
print("Best Parameters (SVM):", grid_search_svm.best_params_)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

Finding the optimal hyperparameters for a Linear Support Vector Classification (Linear SVC)model implementation

# CHOSEN MODEL

```python
from sklearn.naive_bayes import MultinomialNB

# Set the parameter grid
param_grid = {
    'alpha': [0.1, 0.5, 1]  # Smoothing parameter
}

# Define the model
nb = MultinomialNB()

# Perform grid search
grid_search_nb = GridSearchCV(estimator=nb, param_grid=param_grid, cv=5, scoring='f1_weighted', verbose=1)
grid_search_nb.fit(X_train_tfidf, y_train)

# Get the best model
best_nb = grid_search_nb.best_estimator_
print("Best Parameters:", grid_search_nb.best_params_)
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
Best Parameters: {'alpha': 0.1}
```

Finding the optimal hyperparameters for a multinomial Naive Bayes model implementation

```python
from sklearn.ensemble import RandomForestClassifier
param_grid_rf = {'n_estimators': [100,200], 'max_depth': [20,30], 'min_samples_split': [2,3]}
rf = RandomForestClassifier(random_state=42, n_jobs=1, class_weight='balanced')
grid_search_rf = GridSearchCV(estimator=rf, param_grid=param_grid_rf, cv=5, scoring='f1_weighted', verbose=1)
grid_search_rf.fit(X_train_tfidf, y_train)
best_rf = grid_search_rf.best_estimator_
print("Best Parameters (Random Forest):", grid_search_rf.best_params_)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best Parameters (Random Forest): {'max_depth': 30, 'min_samples_split': 3, 'n_estimators': 200}
```

Finding the optimal hyperparameters for a Random Forest Classifier model implementation

# CHOSEN MODEL

```
from xgboost import XGBClassifier
majority_class_weight = len(y_train) / (len(classes) * y_train.value_counts().min())
param_grid_xgb = {'n_estimators': [200,100], 'learning_rate': [0.1], 'max_depth': [6,4]}
xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42, n_jobs=1, scale_pos_weight=majority_class_weight)
grid_search_xgb = GridSearchCV(estimator=xgb, param_grid=param_grid_xgb, cv=5, scoring='f1_weighted', verbose=1)
grid_search_xgb.fit(X_train_tfidf, y_train)
best_xgb = grid_search_xgb.best_estimator_
print("Best Parameters (XGBoost):", grid_search_xgb.best_params_)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Best Parameters (XGBoost): {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 200}
```

Finding the optimal hyperparameters for an Extreme Gradient Boosting Classifier (XGBoost) model implementation

# CHOSEN MODEL

Model: Logistic Regression
Predicted label distribution: Counter({2: 5375, 1: 2650, 0: 241})

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.10 | 0.02 | 0.04 | 1036 |
| 1 | 0.24 | 0.36 | 0.29 | 1733 |
| 2 | 0.68 | 0.67 | 0.67 | 5497 |
| accuracy | | | 0.52 | 8266 |
| macro avg | 0.34 | 0.35 | 0.33 | 8266 |
| weighted avg | 0.52 | 0.52 | 0.51 | 8266 |

Model: SVM
Predicted label distribution: Counter({2: 5350, 1: 2665, 0: 251})

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.11 | 0.03 | 0.04 | 1036 |
| 1 | 0.24 | 0.37 | 0.29 | 1733 |
| 2 | 0.68 | 0.66 | 0.67 | 5497 |
| accuracy | | | 0.52 | 8266 |
| macro avg | 0.34 | 0.35 | 0.33 | 8266 |
| weighted avg | 0.52 | 0.52 | 0.51 | 8266 |

Classification Reports for all tested models

Model: Naive Bayes
Predicted label distribution: Counter({2: 8004, 1: 252, 0: 10})

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.00 | 0.00 | 0.00 | 1036 |
| 1 | 0.30 | 0.04 | 0.08 | 1733 |
| 2 | 0.67 | 0.97 | 0.79 | 5497 |
| accuracy | | | 0.66 | 8266 |
| macro avg | 0.32 | 0.34 | 0.29 | 8266 |
| weighted avg | 0.51 | 0.66 | 0.54 | 8266 |

Model: Random Forest
Predicted label distribution: Counter({1: 6286, 2: 1701, 0: 279})

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.14 | 0.04 | 0.06 | 1036 |
| 1 | 0.22 | 0.81 | 0.35 | 1733 |
| 2 | 0.72 | 0.22 | 0.34 | 5497 |
| accuracy | | | 0.32 | 8266 |
| macro avg | 0.36 | 0.36 | 0.25 | 8266 |
| weighted avg | 0.54 | 0.32 | 0.31 | 8266 |

Model: XGBoost
Predicted label distribution: Counter({2: 7061, 1: 1102, 0: 103})

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.12 | 0.01 | 0.02 | 1036 |
| 1 | 0.27 | 0.17 | 0.21 | 1733 |
| 2 | 0.67 | 0.86 | 0.76 | 5497 |
| accuracy | | | 0.61 | 8266 |
| macro avg | 0.35 | 0.35 | 0.33 | 8266 |
| weighted avg | 0.52 | 0.61 | 0.55 | 8266 |

Achieves the highest accuracy (61%) and excels in identifying the positive majority class (2), with a strong f1-score (76%) due to its high precision (67%) and recall (86%).

Performs better than other models for minority neutral and negative classes (0 and 1) by leveraging gradient boosting to handle the imbalanced dataset effectively.

# HYPERPARAMETER TUNING

```python
# Define XGBoost classifier
xgb = XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='mlogloss')

# Hyperparameter tuning
param_distributions = {
    'n_estimators': [300, 500, 700],
    'max_depth': [5, 7, 3],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.8, 0.9],
    'colsample_bytree': [0.8, 0.9],
    'scale_pos_weight': [1, 3, 10],
    'gamma': [0, 0.1, 0.2],
    'min_child_weight': [1, 3, 5]
}

search = RandomizedSearchCV(
    xgb, param_distributions, n_iter=15, cv=StratifiedKFold(n_splits=3),
    scoring='f1_weighted', n_jobs=1, verbose=2, random_state=42
)

search.fit(train_features_balanced, train_labels_balanced)
```

Defines chosen XGBoost model

Expanded Grid of Hyperparameters for tuning the XGBoost Model for our data

Performs efficient hyperparameter tuning of the XGBoost model using randomized sampling and stratified cross-validation, optimizing for weighted F1-score on the imbalanced dataset. To find the best combination of hyperparameters

# MODEL EVALUATION

```python
# Evaluate the best model
best_model = search.best_estimator_
predictions = best_model.predict(test_features)
print("Best hyperparameters found:")
print(search.best_params_)

# Classification report
target_names = label_encoder.classes_
report = classification_report(test_data['encoded_sentiment'], predictions, target_names=target_names)
print("--- XGBoost (Improved) ---")
print(report)

# Save the vectorizer and model
joblib.dump(tfidf, "improved_tfidf_vectorizer.pkl")
joblib.dump(best_model, "improved_xgboost_model.pkl")

# Save the report
with open("improved_vader_xgboost_report.txt", "w") as f:
    f.write(report)
```

Retrieves the XGBoost with the optimal hyperparameters and the predicted class labels of the training data

Displays the precision, recall and f1-score of the best XGBoost model

Saves the XGBoost Model, Vectoriser and Classification Report as local files

```
Best hyperparameters found:
{'subsample': 0.8, 'scale_pos_weight': 3, 'n_estimators
ytree': 0.8}
--- XGBoost (Improved) ---
              precision    recall  f1-score   support

    negative       0.67      0.56      0.61      1167
     neutral       0.43      0.12      0.19       705
    positive       0.78      0.94      0.85      3669

    accuracy                           0.75      5541
   macro avg       0.63      0.54      0.55      5541
weighted avg       0.72      0.75      0.72      5541
```

# PREDICTING THE DATA

```
# Load the saved model and vectorizer
model_path = "improved_xgboost_model.pkl"
vectorizer_path = "improved_tfidf_vectorizer.pkl"
model = joblib.load(model_path)
tfidf = joblib.load(vectorizer_path)

# Load the new dataset
file_path = "new_drug_reviews.tsv"  # Update with the correct path to your dataset
new_data = pd.read_csv(file_path, sep='\t', on_bad_lines='skip')
new_data.drop("Unnamed: 0", axis=1)
new_data.drop_duplicates()
new_data.drop(new_data[new_data['condition'].str.contains("</span> users found this comment helpful.", na=False)].index, inplace=True)
```

Imports trained XGBoost model, word vectoriser and new dataset for prediction

Cleans and processes new Dataframe (new_data):

1. Removes unnecessary columns like Unnamed: 0
2. Eliminates Duplicate rows and
3. Removes rows where drugs are not defined.

# PREDICTING THE DATA

```
# Preprocessing the new dataset
lemmatizer = WordNetLemmatizer()
stop_words = set(nltk.corpus.stopwords.words('english'))

def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'\s+', ' ', text)  # Remove extra spaces
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    words = [lemmatizer.lemmatize(word) for word in text.split() if word not in stop_words]
    return ' '.join(words)

new_data['review'] = new_data['review'].apply(clean_text)
```

Defines English stop words and initializes lemmatizer to reduce the words to their base form.

Converts all texts to lowercase, and removes extra spaces and punctuation.

Removes stop words and executes the lemmatizer.

# PREDICTING THE DATA

```python
# Categorize sentiment
def categorize_sentiment_from_rating(rating):
    if rating >= 7:
        return 'positive'
    elif rating >= 4:
        return 'neutral'
    else:
        return 'negative'

new_data['true_sentiment'] = new_data['rating'].apply(categorize_sentiment_from_rating)
# Add VADER sentiment scores
def generate_custom_lexicon(data, min_count=15):
    # Extract unigrams, bigrams, and trigrams
    reviews = data['review'].str.split()
    unigrams = chain.from_iterable(reviews)
    bigrams = chain.from_iterable(zip(review, review[1:]) for review in reviews)
    trigrams = chain.from_iterable(zip(review, review[1:], review[2:]) for review in reviews)
```

Assigns sentiment label based on ratings (Positive, Neutral, Negative).

Unigrams: Individual words
Bigrams: Consecutive pairs of words
Trigrams: Consecutive triplets of words.

# PREDICTING THE DATA

```
# Count frequencies
unigram_counts = Counter(unigrams)
bigram_counts = Counter(bigrams)
trigram_counts = Counter(trigrams)

# Filter n-grams by frequency
filtered_unigrams = {word: count for word, count in unigram_counts.items() if count >= min_count}
filtered_bigrams = {" ".join(bigram): count for bigram, count in bigram_counts.items() if count >= min_count}
filtered_trigrams = {" ".join(trigram): count for trigram, count in trigram_counts.items() if count >= min_count}

# Combine all n-grams
all_phrases = {**filtered_unigrams, **filtered_bigrams, **filtered_trigrams}

# Compute average ratings for phrases
phrase_sentiment = {}
for phrase in all_phrases.keys():
    relevant_reviews = data[data['review'].str.contains(phrase)]
    avg_rating = relevant_reviews['rating'].mean() if not relevant_reviews.empty else None
    if avg_rating:
        sentiment_score = (avg_rating - 5) / 5  # Scale sentiment between -1 and +1
        phrase_sentiment[phrase] = sentiment_score

return phrase_sentiment
```

Count the occurrence of each unigram, bigram, and trigram.

Filter and merge all filtered unigrams, bigrams, and trigrams into a single dictionary.

For each phrase in all_phrase, find the associated reviews and calculate average rating.

```
# Create and save custom lexicon
custom_lexicons = generate_custom_lexicon(new_data)
with open("custom_vader_lexicons.txt", "w") as f:
    for phrase, score in custom_lexicons.items():
        f.write(f"{phrase}\t{score}\n")
```

# PREDICTING THE DATA

```
# Add VADER sentiment scores
sia = SentimentIntensityAnalyzer()
sia.lexicon.update(custom_lexicons)
new_data['vader_compound'] = new_data['review'].apply(lambda x: sia.polarity_scores(x)['compound'])
new_data['vader_pos'] = new_data['review'].apply(lambda x: sia.polarity_scores(x)['pos'])
new_data['vader_neg'] = new_data['review'].apply(lambda x: sia.polarity_scores(x)['neg'])
new_data['vader_neu'] = new_data['review'].apply(lambda x: sia.polarity_scores(x)['neu'])
```

For each phrase in all_phrase, find the associated reviews and calculate average rating.

# PREDICTING THE DATA

```
# Transform the text data using the loaded vectorizer
new_tfidf = tfidf.transform(new_data['review'])

# Combine TF-IDF features with VADER sentiment scores
new_features = hstack([new_tfidf, new_data[['vader_compound', 'vader_pos', 'vader_neg', 'vader_neu']]])

# Predict the sentiment using the loaded model
predictions = model.predict(new_features)

# Map numeric predictions back to sentiment labels

label_encoder = LabelEncoder()
new_data['encoded_sentiment'] = label_encoder.fit_transform(new_data['true_sentiment'])
joblib.dump(label_encoder, "improved_label_encoder.pkl")
label_encoder_path = "improved_label_encoder.pkl"  # Save and load label encoder
label_encoder = joblib.load(label_encoder_path)
predicted_sentiments = label_encoder.inverse_transform(predictions)

# Add the predicted sentiment to the new dataset
new_data['predicted_sentiment'] = predicted_sentiments

# Save the results
output_path = "predicted_sentiments.csv"
new_data.to_csv(output_path, index=False)

print("Sentiment predictions saved to:", output_path)
```

Applies the vectoriser developed from the training data onto the new dataset

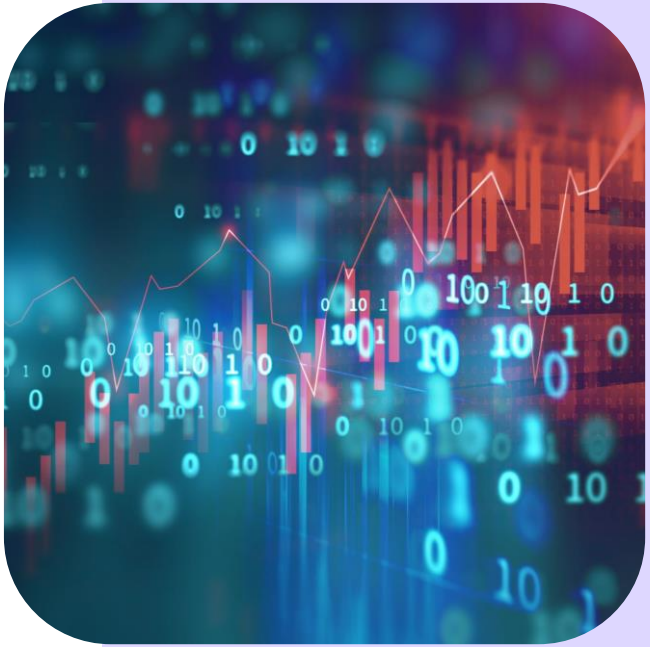
Combines TF-IDF (Term Frequency-Inverse Document Frequency) matrix with VADER (Valence Aware Dictionary and Sentiment Reasoner) scores.

Converts the predicted rating number of the review to the Positive, Neutral and Negative Labels

Outputs the processed dataset for further business analysis in external software

# 07

# CONCLUSION



# BUSINESS APPLICATIONS

**Customer Feedback Analysis:**
Understand sentiments behind product reviews and patient feedback

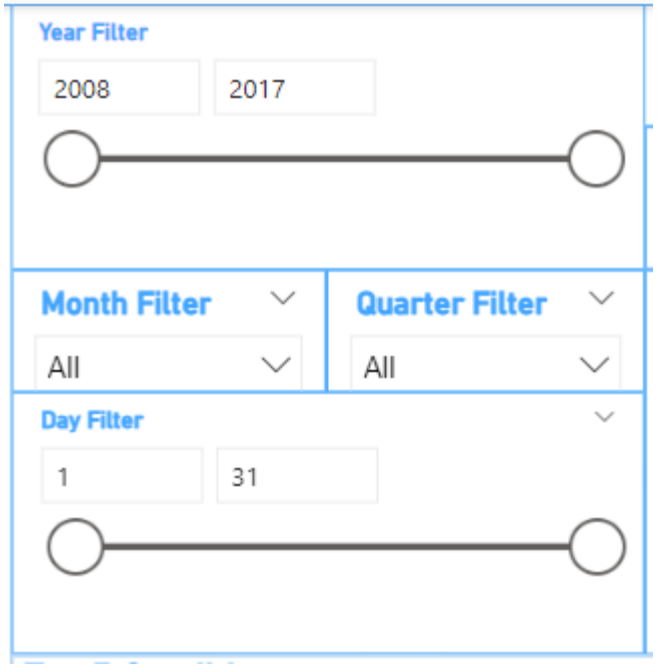
**Brand Reputation Management:**
Monitor and respond to public perceptions of drug products

**Product Improvement:**
Identify features customers like or dislike for targeted enhancements

# CUSTOMER FEEDBACK ANALYSIS

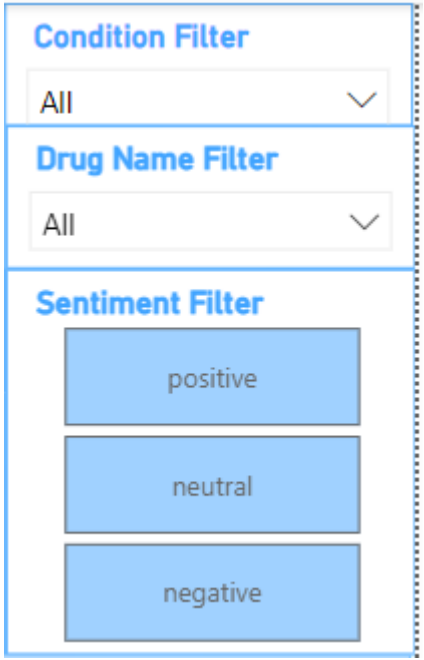

Overall sentiment trends are derived from sentiment value analysis, where **each positive or negative review is assigned a score of 1 or –1**. These trends are visually represented in a bar chart, showing sentiment fluctuations over time.

# CUSTOMER FEEDBACK ANALYSIS



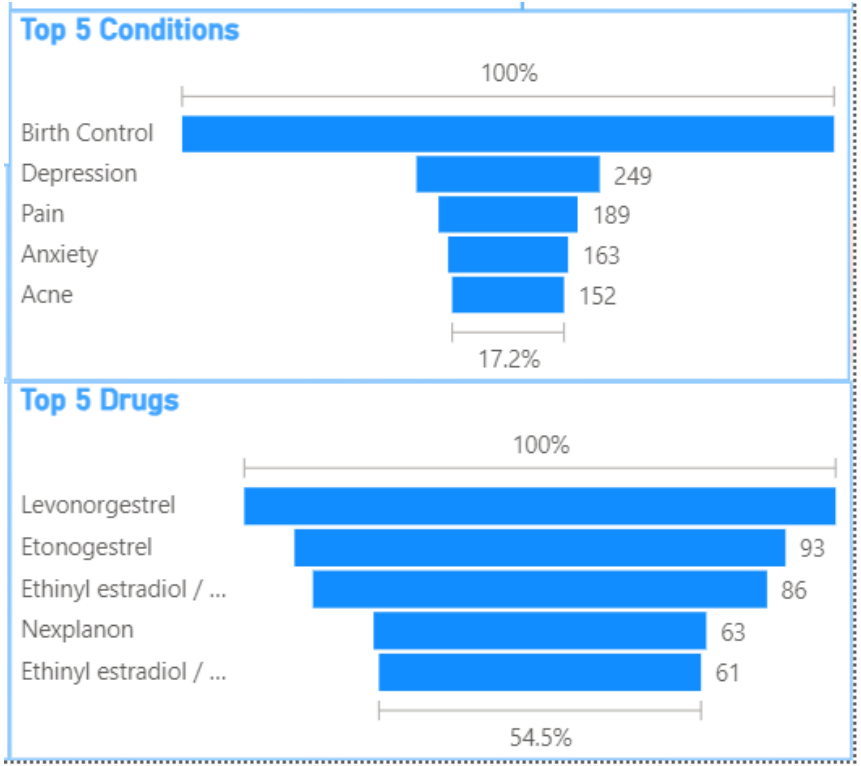

The dashboard features a **time period filter**, allowing users to refine sentiment trends by **year, month, quarter, or day** for a more detailed analysis of temporal patterns and shifts in customer feedback.

# CUSTOMER FEEDBACK ANALYSIS




The dashboard offers advanced filters for **drug name, medical condition, and sentiment classification (positive, negative, or neutral)**, enabling precise data segmentation. Users can isolate sentiment trends, refine datasets dynamically, and extract actionable insights to enhance decision-making.

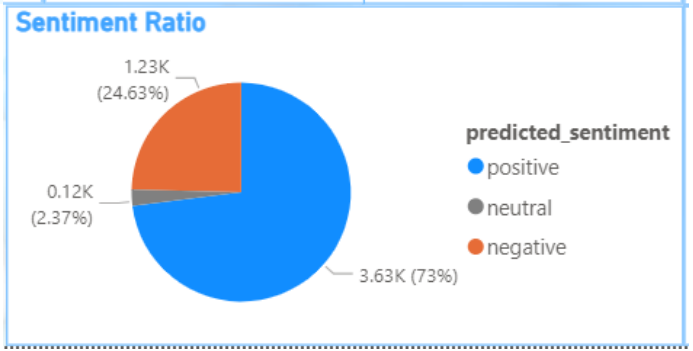
# CUSTOMER FEEDBACK ANALYSIS




The dashboard features **tree charts** that visualize **the top five conditions and drugs** based on sentiment analysis. These charts provide a **hierarchical representation of data**, allowing users to quickly identify the most discussed medications and medical conditions. By analyzing these insights, stakeholders can recognize key trends, assess product reception, and make data-driven improvements to optimize patient and customer experiences.

# CUSTOMER FEEDBACK ANALYSIS

| 6.86 | 0.73 | 0.25 | 0.02 |
|---|---|---|---|
| Average of rating | Positive_Ratio | Negative_Ratio | Neutral_Ratio |

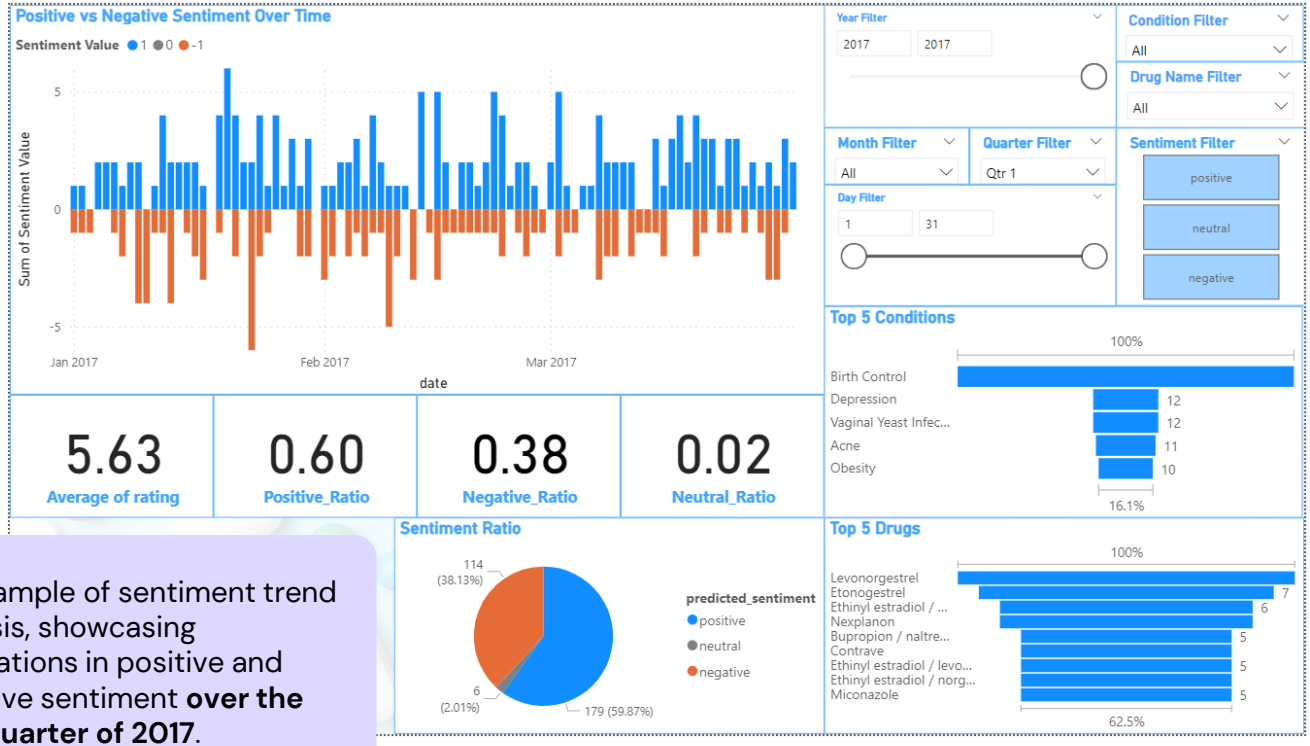



These visuals enhance the dashboard experience by providing a clear summary of the **average rating of the reviews** and **the proportions of sentiment classes**—positive, negative, and neutral—**filtered by user-specified criteria**. By visually breaking down sentiment distribution and overall satisfaction, users can quickly identify key insights from the data.

# EXAMPLE USE CASES


An example of sentiment trend analysis, showcasing daily fluctuations in positive and negative sentiment **during May 2015**.

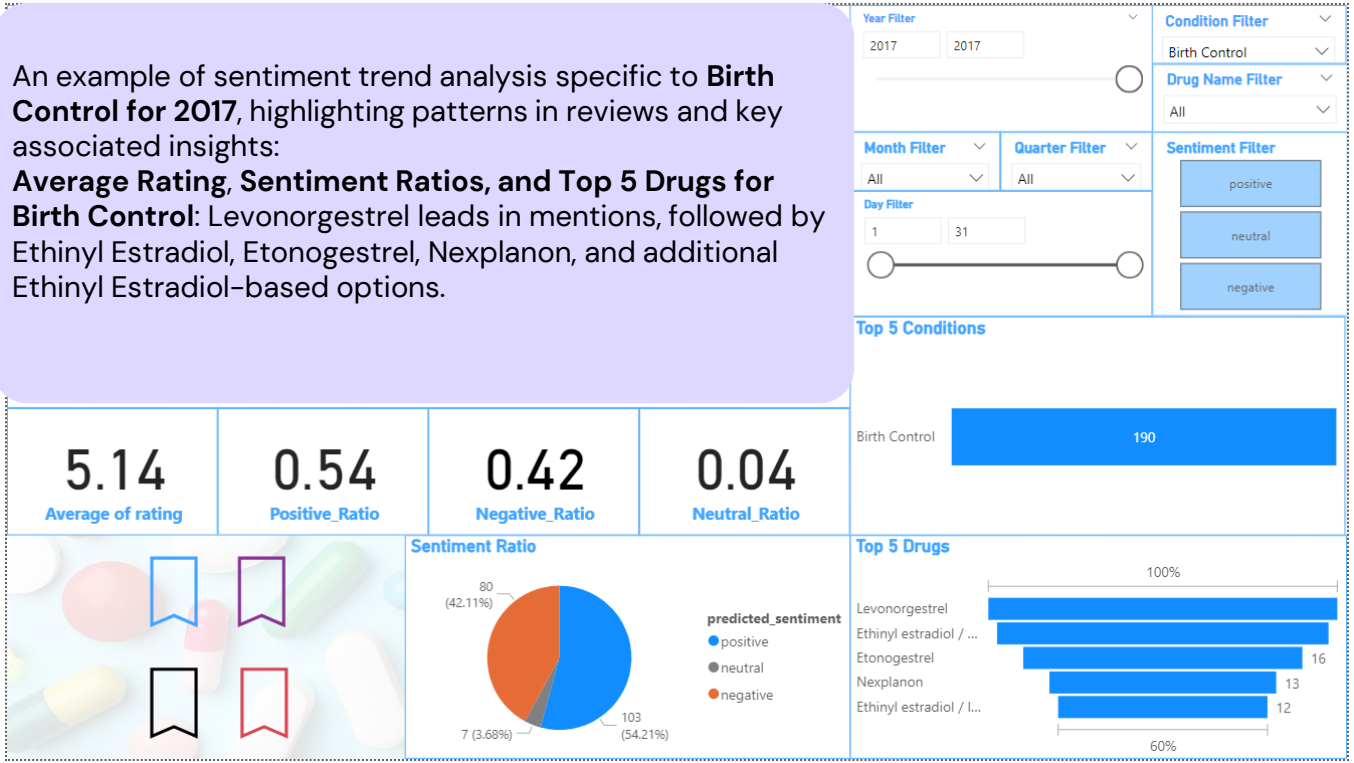
# EXAMPLE USE CASES




An example of sentiment trend analysis, showcasing fluctuations in positive and negative sentiment **over the first quarter of 2017**.

# EXAMPLE USE CASES

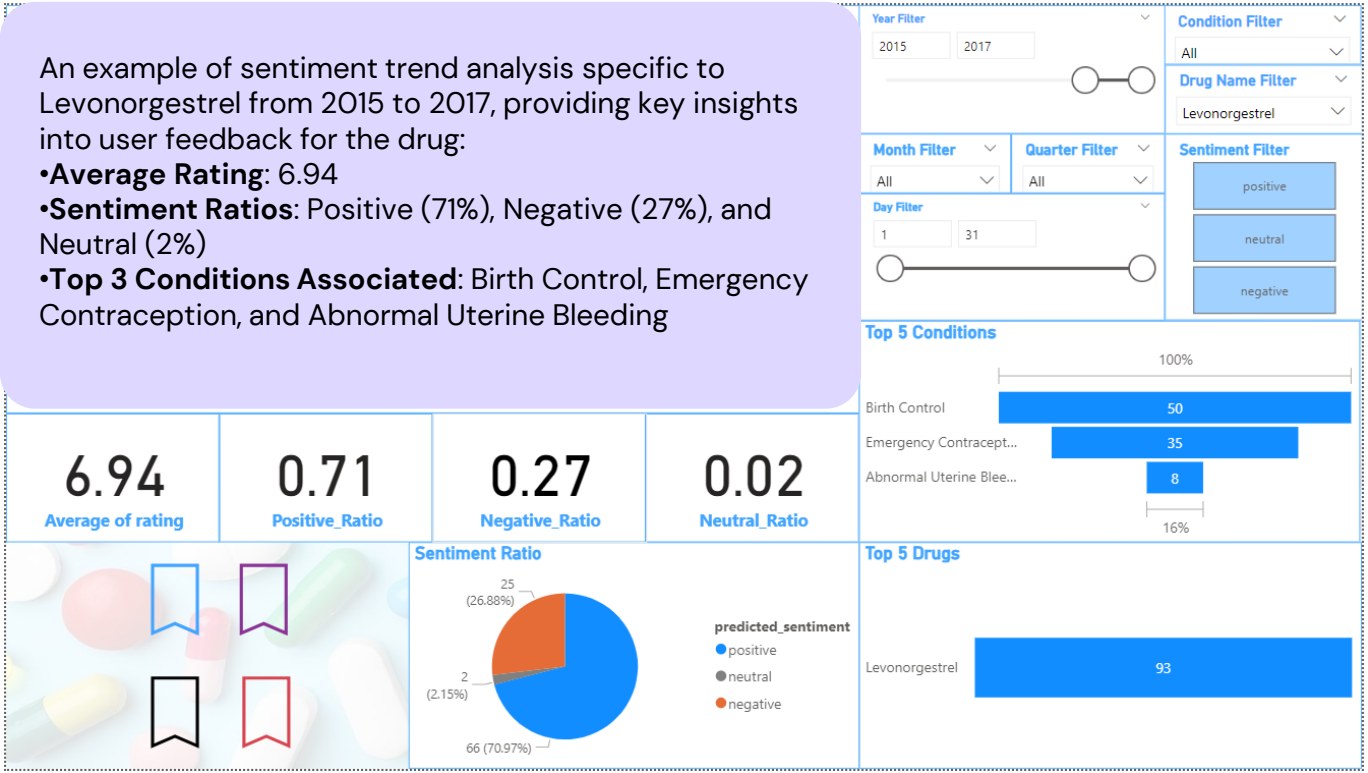
An example of sentiment trend analysis specific to **Birth Control for 2017**, highlighting patterns in reviews and key associated insights:
**Average Rating**, **Sentiment Ratios, and Top 5 Drugs for Birth Control**: Levonorgestrel leads in mentions, followed by Ethinyl Estradiol, Etonogestrel, Nexplanon, and additional Ethinyl Estradiol-based options.



# EXAMPLE USE CASES

An example of sentiment trend analysis specific to Levonorgestrel from 2015 to 2017, providing key insights into user feedback for the drug:

- **Average Rating**: 6.94
- **Sentiment Ratios**: Positive (71%), Negative (27%), and Neutral (2%)
- **Top 3 Conditions Associated**: Birth Control, Emergency Contraception, and Abnormal Uterine Bleeding



# CONCLUSION

The sentiment analysis model effectively combines **VADER sentiment scoring**, **custom lexicons**, **TF-IDF vectorization**, and **XGBoost classification** to analyze textual reviews. By cross-referencing sentiment predictions with existing user ratings, it ensures accuracy and reliability. Key enhancements, such as **SMOTE oversampling** and hyperparameter tuning, address data imbalances and optimize performance.

The model demonstrates strong predictive capabilities across sentiment categories and offers actionable insights for applications like **customer feedback analysis**, **brand management**, and **product improvement**. This project showcases a scalable solution for **real-world sentiment analysis**, with potential for **further optimization** and **expanded use**.

# References


- https://www.youtube.com/watch?v=QpzMWQvxXWk
- https://www.youtube.com/watch?v=ilblwDAQo4Q
- https://www.analyticsvidhya.com/blog/2022/07/sentiment-analysis-using-python/
- https://www.kaggle.com/datasets/bittlingmayer/amazonreviews
- https://github.com/jmachado100/NLP-Spotify-Reviews-Text-Classification/blob/main/NLP Spotify Reviews Text Classification
- https://www.nltk.org/
- https://www.geeksforgeeks.org/python-sentiment-analysis-using-vader/
- https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.fit

# THANKS!