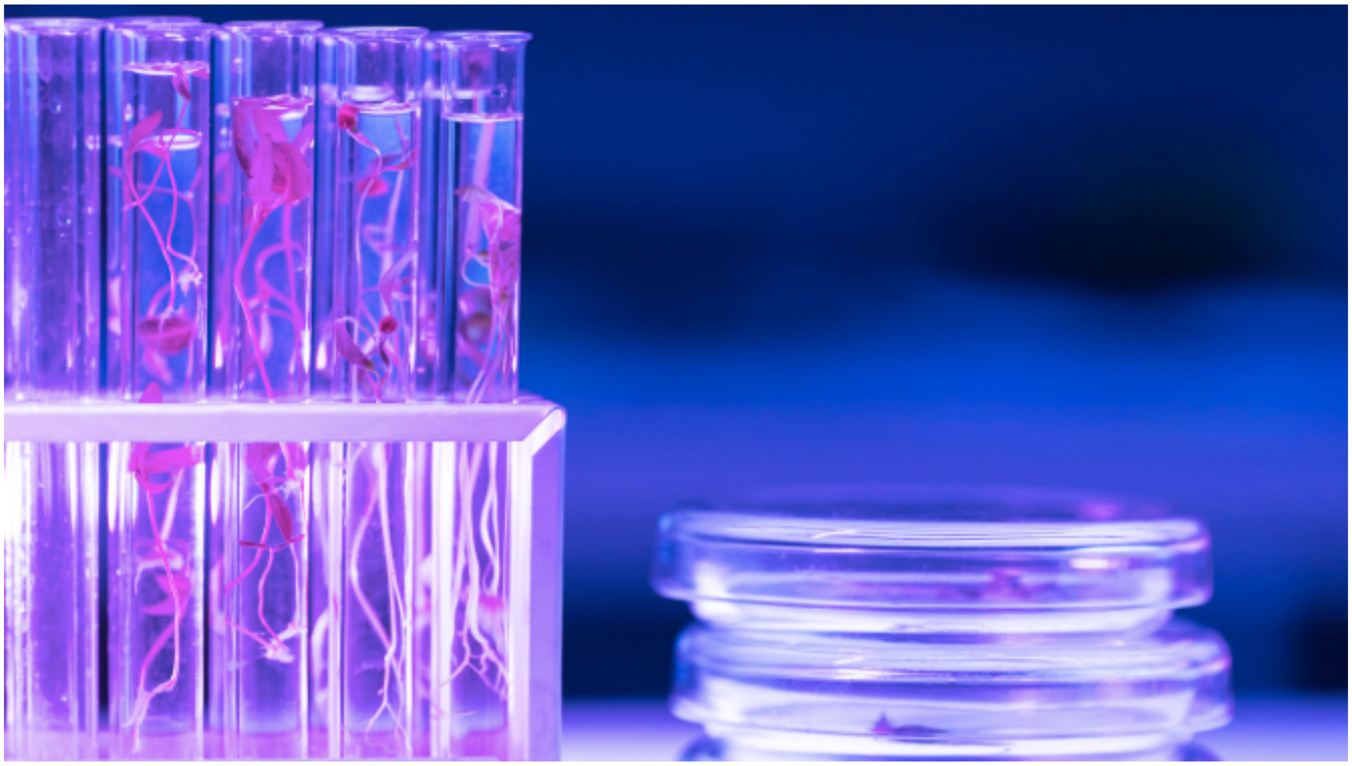
FRACTAL DATA

Home ⌄
About ⌄

Evaluating Psychedelic Efficacy: A Machine Learning & NLP Approach

Written in March 5, 2023 by **FD** Uncategorized

Contents:



Introduction: Why & How

This blog post is going to get quite technical, but first: an introduction to why the topic of psychedelic medicine is important to me and how I believe machine learning (with NLP, natural language processing) can help us grow our understanding of psychedelics' impacts.

If you have ever felt anxious, depressed, or afraid for your emotional well being, chances are you've searched online to see how others manages challenges like those you face. Maybe you don't have access to a psychiatrist or therapist, or perhaps you've given it a try and found yourself wanting additional resources. And even if you've received great support from a mental health practitioner, they have legal limitations dictating the suggestions they make.

While professional ethics are important, the criminal-legal system often gets things wrong, and this is certainly the case when it comes to the war on drugs. Criminalization of drugs purposefully and disproportionately targets Black people, poor people, and other marginalized communities, and it is harmful for everybody, regardless of identity. People need to be able to feel we have lives worth living, and we benefit from having access to all available knowledge and resources for supporting basic wellness and personal development. One resource comes in the form of medicinal drugs, many of which are criminalized. Criminalization not only limits availability of particular substances but also decreases access to research-based information people need in order to use safely.

Psychedelic medicine is not without its proponents and research studies. Much can be found on, for example, maps.org. There are just fewer formal sources of data from clinical studies than we have for prescription psych meds. There is, however, a boon of information about psychedelics in the form of informal reports posted on forums including Reddit, PsychonautWiki, or Erowid. Since these consist of people simply sharing their experiences in a narrative format, they often are anonymous and usually do not come along with much quantitative data.

Personally, I like my doctor, and I trust most of what she tells me. But I'm a curious person, and I also value people's subjective accounts of unconventional approaches to self care. I love reading vivid details that people share about their experiences on forums, but I also want numeric data that can help me decide whether or not to try any given medicinal option. My doctor can give me hard facts, but not about psychedelics; strangers on the internet can tell me about psychedelics but don't share much in the way of hard facts.

This is where machine learning and natural language processing come in. The contents of narrative psychedelic experience reports can be analyzed by a computer and quantified in a variety of ways. In this project, I have developed one approach to summarize numerically a wealth of narrative information about psychedelics shared across decades by thousands of people.

I hope my findings can be used by individuals, nonprofits, or startups who are working to better understand the ins and outs of how psychedelics affect people. Spoiler alert: my research points to the possibility that people might prefer psychedelics over currently available prescription psych meds. And what better indication of the efficacy of a medicine used to treat emotional symptoms than user-reported emotional effects? Now is a great time for anybody who cares about healthcare to continue to develop psychedelic medicines and advocate for their decriminalization.

One note before I start building toward the more technical side of things: I'm enthusiastic about psychedelics, so my project is about psychedelics. Nobody has my consent, however, to spread myths about psychedelic drug users being wiser, safer, more moral, etc. than users of any criminalized substances. Nobody should be punished for putting things in their body in an attempt to relieve suffering or feel well, whether or not they succeed.

The Data Science Pipeline

The "data science pipeline" is just data scientists' term for the process we use to move from messy, unstructured data such as a narrative texts gathered from disparate sources, to quantitative, actionable insights derived via careful analysis and machine learning techniques. This section will include code for how I completed each step of the pipeline, but code will be prefaced by a plain-language summary, so you won't miss any key details if you skip the technical bits.

Every decent research project starts with a decently compelling question or two. Here's mine: How well might psychedelic drugs work to treat mental illness, compared with prescription psych meds? What insights can be drawn from anonymously-submitted psychedelic experience reports? In this document, I outlined some of my project goals before getting started.

The data science pipeline for this project consists of the following stages:

- Wrangling
- Parsing & Cleaning Text
- Feature Engineering
- Exploratory Data Analysis
- Modeling
- Deployment

If you're following along with the code, here's a list of imports:

```
# For wrangling
import requests
from bs4 import BeautifulSoup
from tqdm import tqdm # I just use this to show a progress bar for some of
the potentially slower loops.
import pandas as pd
import json
# For NLP/EDA
import unicodedata # for function remove_accented_chars
import re
import numpy as np
import contractions
import string
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
from textstat import flesch_kincaid_grade
import spacy
from spellchecker import SpellChecker
from textblob import TextBlob
from nltk.tokenize import word_tokenize
from nltk.util import ngrams
# For modeling
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.metrics import make_scorer, balanced_accuracy_score, log_loss,
f1_score
from sklearn.metrics import top_k_accuracy_score, recall_score,
roc_auc_score, confusion_matrix
from sklearn.dummy import DummyClassifier
from sklearn.naive_bayes import ComplementNB, MultinomialNB
from xgboost import XGBClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import BaggingClassifier
# For deployment
import pickle
```

Wrangling

The next step in the pipeline is "wrangling" data. My data came from two types of sources: 1. informal psychedelic experience reports and 2. formal medical studies.

1: Scrape Psychedelic Experience Reports

We can gather the text of a website by "scraping" its contents into a file of our own that we can then manipulate. So far, I have only scraped psychedelic experience reports from erowid.org. I'd like to expand to psychonautwiki and some psychedelic subreddits someday, or maybe you'll beat me to it? Please let me know if you do!

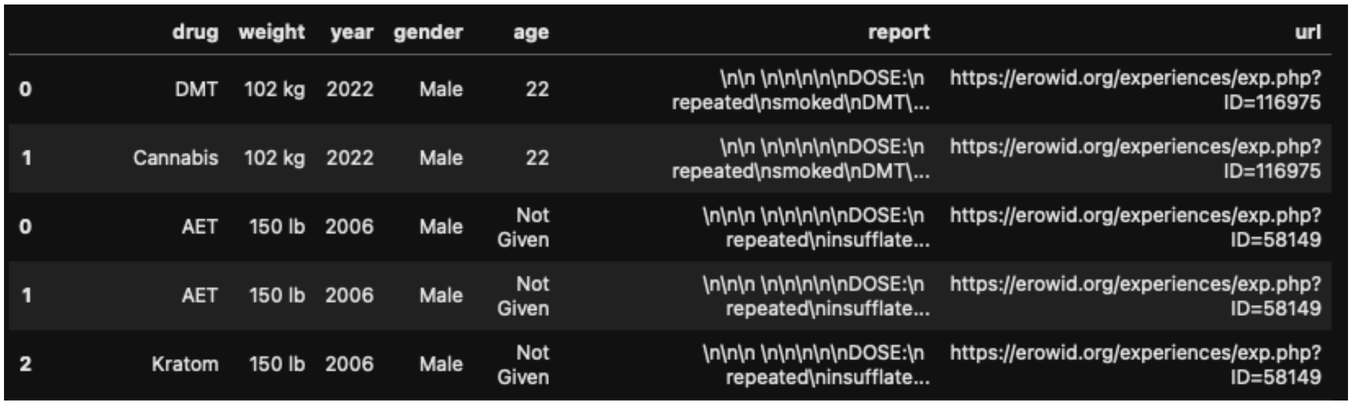
I used code to fetch text from 4,562 reports about people's experiences with 191 psychedelic substances that can be categorized as tryptamines (i.e. LSD, Psilocybin Mushrooms, DMT), phenethylamines (i.e. MDMA, Mescaline, 2C-X), arylcyclohexylamines (i.e. Ketamine, PCP, PCE), or "other entheogens" (entheogen = substance that supports compassionate connection with oneself, others, and the universe). I dumped these into a csv file that could be turned into a dataframe (tabular representation of data like a spreadsheet with rows and columns).

Some of my code for scraping Erowid:

```
# Start with a list of just drugs of interest, psychedelic drugs:
psychedelic_drugs = ['AET', 'AL-LAD', 'ALD-52', 'ALEPH', 'Aleph-4',
'Allylescaline', 'AMT', 'Arylcyclohexylamines', 'Ayahuasca', 'Banisteriopsis
caapi', 'BOD', 'BOH-2C-B', 'Bufotenin', 'Cacti - Mescaline-containing',
'DALT', 'Deschloroketamine', 'DET', 'DiPT', 'DMT', 'DMT-Containing', 'DMXE',
'DOB', 'DOC', 'DOET', 'DOF', 'DOI', 'DOIP', 'DOM', 'DON', 'DOPR', 'DPT',
'EIPLA', 'EPT', 'Escaline', 'ETH-LAD', 'Fluorexetamine', 'H.B. Woodrose',
'Harmaline', 'Harmine', 'Herbal Ecstasy', 'HOT-17', 'HOT-2', 'HOT-7', 'Huasca
Brew', 'Huasca Brew Group', 'Huasca Combo', 'Huasca Group', 'HXE', 'Iboga
Alkaloid Group', 'Ibogaine', 'Isoproscaline', 'Ketamine', 'LSA', 'LSD',
'LSM-775', 'LSZ', 'MALT', 'MDA', 'MDAI', 'MDE', 'MDMA', 'MEM', 'Mescaline',
'MET', 'Methallylescaline', 'Methoxetamine', 'Methoxpropamine', 'Mimosa
ophthalmocentra', 'Mimosa spp.', 'Mimosa tenuiflora', 'MIPLA', 'MIPT',
'MMDA', 'MMDA-3a', 'MPT', 'Mushrooms', 'Mushrooms - G. spectabilis',
'Mushrooms - P. atlantis', 'Mushrooms - P. azurescens', 'Mushrooms - P.
cubensis', 'Mushrooms - P. cyanescens', 'Mushrooms - P. mexicana', 'Mushrooms
- P. semilanceata', 'Mushrooms - P. subaeruginosa', 'Mushrooms - P.
tampanensis', 'Mushrooms - P. weilii', 'Mushrooms - Panaeolus cyanescens',
'MXiPr', 'PCE', 'PCP', 'Peyote', 'Phenethylamine', 'Phenethylamines',
'Phenethylamines - Other', 'PIPT', 'Proscaline', 'Psilocin', 'Psilocybin',
'S-Ketamine', 'Tabernanthe iboga', 'TCB-2', 'Tetrahydroharmine', 'TMA',
'TMA-2', 'TMA-6', 'Tryptamines - Substituted', '1B-LSD', '1cP-AL-LAD', '1cP-
LSD', '1F-LSD', '1P-ETH-LAD', '1P-LSD', '1V-LSD', "2'-Oxo-PCE", '2-
Fluorodeschloroketamine', '2-Me-DMT', '2C-B', '2C-B-Fly', '2C-C', '2C-CN',
'2C-D', '2C-E', '2C-EF', '2C-G-N', '2C-H', '2C-I', '2C-IP', '2C-N', '2C-P',
'2C-T', '2C-T-13', '2C-T-2', '2C-T-21', '2C-T-4', '2C-T-7', '2C-TFM', '3,4-
MD-PCP', '3-Cl-PCP', '3-HO-PCE', '3-HO-PCP', '3-Me-PCE', '3-Me-PCPy', '3-MEO-
PCE', '3-MeO-PCMo', '3-MeO-PCP', '3-Methyl-PCP', '3C-E', '3C-P', '3F-PCP',
'4-AcO-DALT', '4-AcO-DET', '4-AcO-DiPT', '4-AcO-DMT', '4-AcO-DPT', '4-AcO-
EIPT', '4-AcO-EPT', '4-AcO-MALT', '4-AcO-MET', '4-AcO-MiPT', '4-AcO-MPT', '4-
HO-DET', '4-HO-DiPT', '4-HO-DPT', '4-HO-EPT', '4-HO-MALT', '4-HO-MCPT', '4-
HO-MET', '4-HO-MiPT', '4-HO-MPT', '4-HO-PIPT', '4-MeO-DMT', '4-MeO-MiPT', '4-
MeO-PCP', '4-MTA', '4-PrO-DMT', '4C-D', '5-Chloro-AMT', '5-MeO-AET', '5-MeO-
AMT', '5-MeO-DALT', '5-MeO-DET', '5-MeO-DiPT', '5-MeO-DMT', '5-MeO-DPT', '5-
MeO-EIPT', '5-MeO-MALT', '5-MeO-MET', '5-MeO-MIPT', '5-MeO-PIPT', '5-MeO-
TMT', '5-Methoxy-Tryptamine']
# Create a list of links to pages associated with these drugs.
# Dashes and periods need to be deleted from drug names before they're
inserted into a url; spaces need to be replaced with underscores.
drug_names_for_links = []
```

```
for drug in psychedelic_drugs:
    no_dash = drug.replace('-', '')
    no_period = no_dash.replace('.', '')
    no_double_space = no_period.replace('  ', '_')
    no_space = no_double_space.replace(' ', '_')
    drug_names_for_links.append(no_space)
# Create strings for urls from the template erowid uses with each drug name
inserted into its own url.
drug_urls = []
for drug in drug_names_for_links:
    drug_urls.append('https://erowid.org/experiences/subs/exp_' + drug +
'.shtml')
# Now I have a url to each drug's homepage. Each drug homepage is filled with
links to multiple experience reports about that drug. Navigate to each page
and gather the link to "Show All" experience reports.
# Collect all hrefs
vault_hrefs = []
bad_urls = []
for url in tqdm(drug_urls):
    # Some urls could be wrong if I didn't change the drug names properly.
    try:
        drug_page = requests.get(url)
        drug_soup = BeautifulSoup(drug_page.content, "html.parser")
        href = drug_soup.find('img', attrs={'alt':'Show All
Reports'}).parent['href']
        vault_hrefs.append(href)
    except: bad_urls.append(url)
# Turn hrefs into proper urls
vault_urls = []
for href in vault_hrefs:
    vault_urls.append('https://erowid.org' + href)
# On each report page there is a table, but most of the information is not in
a table. There may be more efficient ways, but I'll start out by just pulling
various elements separately with a function and then joining them together
into a dataframe.
def page_to_df_or_dict(url):
    # Many pages may not have the same table structure as the page I tried.
    try:
        # extract information from the tables.
        mini_df = pd.read_html(url)
        drugs_reviewed = mini_df[2][3].to_list()
        weight = mini_df[3][1].to_list()
        remainiing_relevant_info = mini_df[4][0][0:3]
        year = remainiing_relevant_info[0]
        gender = remainiing_relevant_info[1]
        age = remainiing_relevant_info[2]
        # Create a dataframe from the table information.
        mini_df = pd.DataFrame({'drug':drugs_reviewed, 'weight':weight[0],
'year':year.replace('Exp Year: ', ''), 'gender':gender.replace('Gender: ',
''), 'age':age.replace('Age at time of experience: ', '')})
        # Access, extract, and add to the dataframe the narrative text.
        this_page = requests.get(url)
        this_soup = BeautifulSoup(this_page.content, "html.parser")
        this_text = this_soup.find('div', attrs={'class':'report-text-
surround'}).get_text()
        mini_df['report'] = this_text
        # Add the url to the dataframe
        mini_df['url'] = url
        return mini_df
    # If I can't get all the table information, maybe I can at least get just
the text.
    except:
        text_dict = {}
        this_page = requests.get(url)
        this_soup = BeautifulSoup(this_page.content, "html.parser")
        this_text = this_soup.find('div', attrs={'class':'report-text-
surround'}).get_text()
        text_dict[url] = this_text
        return text_dict
# Since some urls pointing to experience reports may be bad, account for that
but otherwise used text scraped with the function above to build out a
dataframe:
for url in tqdm(report_urls):
    try:
        new_rows = page_to_df_or_dict(url)
        df = pd.concat([df, new_rows])
    except:
        try:
            page_text_dict = page_to_df_or_dict(url)
            just_texts.update(page_text_dict)
        except: invalid_urls.append(url)
```

See the full notebook for this step of the process here. That all resulted in a dataframe that looked like this, except with thousands of rows:



	drug	weight	year	gender	age	report	url
0	DMT	102 kg	2022	Male	22	\n\n \n\n\n\n\nDOSE:\n repeated\nsmoked\nDMT\...	https://erowid.org/experiences/exp.php?ID=116975
1	Cannabis	102 kg	2022	Male	22	\n\n \n\n\n\n\nDOSE:\n repeated\nsmoked\nDMT\...	https://erowid.org/experiences/exp.php?ID=116975
0	AET	150 lb	2006	Male	Not Given	\n\n\n \n\n\n\n\nDOSE:\n repeated\ninsufflate...	https://erowid.org/experiences/exp.php?ID=58149
1	AET	150 lb	2006	Male	Not Given	\n\n\n \n\n\n\n\nDOSE:\n repeated\ninsufflate...	https://erowid.org/experiences/exp.php?ID=58149
2	Kratom	150 lb	2006	Male	Not Given	\n\n\n \n\n\n\n\nDOSE:\n repeated\ninsufflate...	https://erowid.org/experiences/exp.php?ID=58149

Note: You can see here that the "report" column aka psychedelic experience narrative — the key data point of interest for me — has a lot of junk in it like traces of html formatting (i.e. \n\n\n\n\n\n\n which is just computer code for 7 new lines where somebody hit the enter key) that were later removed during the text cleaning process.

2. Prescription Psych Med Studies

The other source of data I downloaded as a csv file already in tabular format. This came from a few different medical studies: druglib, drugs.com, psytar.

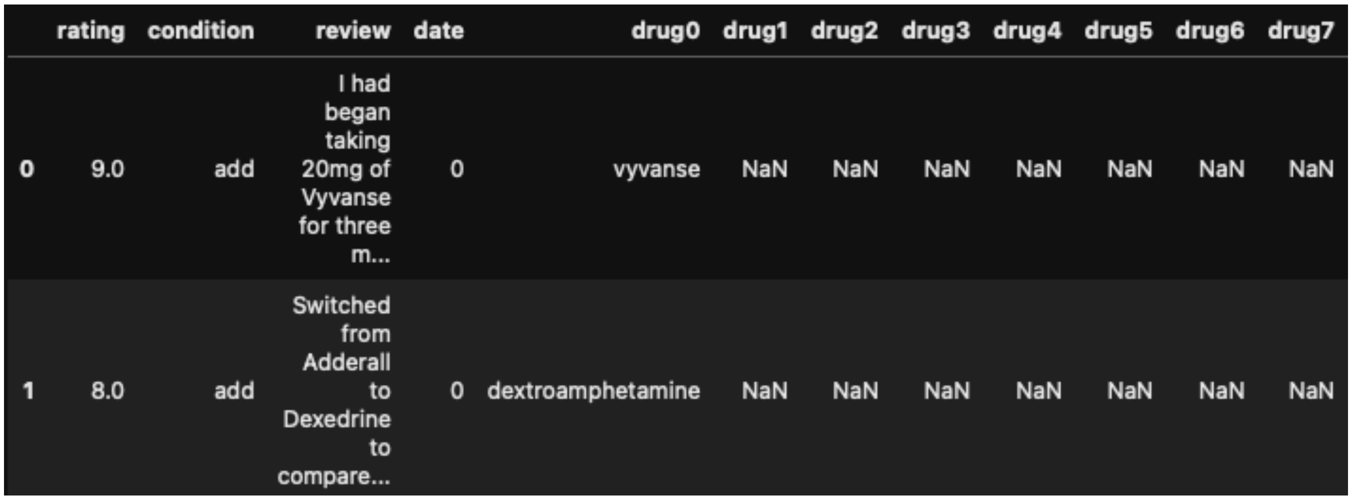
Why would I need this data about prescription psych meds when my project is all about psychedelics? My aim is to quantify some highly subjective narrative texts. One great thing about these formal medical studies is that they contain not only narrative reviews but lots of other data about each drug, as well, including, importantly, a rating on a scale of 1-5 provided by the patient reviewing each given drug.

My major task, therefore, has been to reverse engineer ratings from raw review text. This is labeled data, which means each drug review comes labeled with a rating. Text features associated with high or low ratings of each drug can be "learned" by an AI algorithm thanks to the labels. This learning can then be used make predictions or assign assumed ratings to texts where such labels do not exist, specifically, the scraped psychedelic experience reports.

An example: Imagine a hypothetical medical study participant is in love with their Xanax prescription and somewhere within the body of their review they type ":) 🙂:)". They also give Xanax a rating of 5/5. The machine learning algorithm might pick up on the fact that the emoticon ":)" should be associated with high ratings. A psychedelic experience report from Erowid might subsequently be more likely to be assigned a higher predicted rating if it contains ":)" than if it does not. But the AI can only know to do this if it has been trained using the text+label combinations from the medical study of prescription psych meds.

Once I downloaded all the data from these essential medical studies, I merged the three datasets together into one dataframe and deleted plenty of data that was irrelevant for my purposes using the pandas library. (A library is a set of coding tools that help any programmers complete important common tasks, and pandas is one of the most widely-used libraries for data scientists.)

These were fairly boilerplate tasks, so I won't include the code here, but it's linked above if you'd like to take a look at some of the munging challenges I faced. The resulting dataframe after basic cleaning looked like this (some people reviewed multiple drugs in the course of a single "review" text , thus the multiple drug columns):



	rating	condition	review	date	drug0	drug1	drug2	drug3	drug4	drug5	drug6	drug7
0	9.0	add	I had began taking 20mg of Vyvanse for three m...	0	vyvanse	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	8.0	add	Switched from Adderall to Dexedrine to compare...	0	dextroamphetamine	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Parsing & Cleaning Text (NLP)

Machine learning algorithms require numeric inputs to work, so text needs to be quantified somehow. It's also better to have text that is standardized with as few unnecessary features as possible. Extraneous words and symbols not essential to the core meaning create "noise" that distracts the AI from picking up on key patterns that are associated with high or low drug ratings. Therefore, extensive transformation (NLP) needs to be conducted on raw text.

For example, as stated above, if a text contains emoji such as ":)", this could be an important feature for the machine learning model to incorporate into its predictive process. But simple parentheses around words (like this) should be removed. We'd much rather have the model recognize the words "like" and "this" than "(like" and "this)". If a text contains "(like", that word will not be treated the same ways as "like", which is a problem, especially if, for example, somebody writes, "I really like this drug." Dirty text would make it so that this important sentence could get ignored rather than used as points in favor of a high predicted rating.

I built several functions to identify and remove most extraneous symbols from the text of both psych med reviews and psychedelic experience reports:

```
def remove_accented_chars(text):
    """Turns accented characters i.e. é, ä into unaccented characters ie e,
a, in a string
    Args:
      text: string containing any characters
    Returns: same string, with any accented characters replaced
    """
    text = unicodedata.normalize('NFKD', text).encode('ascii',
'ignore').decode('utf-8', 'ignore')
    return text
```

```
def strip_most_punc(df, column):
    """Deletes most common symbols from a string.
    Does not delete the following strings ' : ; () ! ? # %
    Comes with progress bar built in.
    Args:
      df: name of dataframe
      column: name of a column from the dataframe where strings are that
should be stripped
    Returns: same string in the same position within the df, with listed
characters removed and replaced with a space
    """
    punc = str.maketrans('', '', '-[]{}",<>./@^&*_~')
    for row in tqdm(range(len(df))):
        df.loc[row,column] = df.loc[row,column].translate(punc)
def strip_apostrophe(df, column):
    """Deletes apostrophes wherever they appear in strings in a dataframe
column.
    Args:
      df: name of dataframe
      column: name of a column from the dataframe where strings are that
should be stripped
    Returns: same string in the same position within the df, with apostrophes
removed
    """
    punc = str.maketrans('', '', "'")
    for row in tqdm(range(len(df))):
        df.loc[row,column] = df.loc[row,column].translate(punc)
def strip_non_emoji_emoji_symbol(df, column):
    """Deletes the symbols :;() where they appear next to letters.
        Replaces these symbols with a space: ' '. Leaves them anytime they
don't appear adjacent to letters.
    Args:
        df = dataframe with the strings to be stripped
        column: name of a column from the dataframe where strings are that
should be stripped

    Returns: same string in the same position within the df, with ;:()
removed in the fashion described above.
    """
    import re
    chars_to_remove = ['(', ')', ':', ';']

    # Isolate 1 narrative at a time
    for row in tqdm(range(len(df))):
        string_to_strip = df.loc[row,column]
        # Match the ( where it appears before or after a number.
        parenth_open_matches = re.findall('\([A-Za-z]', string_to_strip) +
re.findall(
            '[A-Za-z]\(', string_to_strip)
        # Match remaining symbols
        parenth_close_matches = re.findall('\)[A-Za-z]', string_to_strip) +
re.findall(
            '[A-Za-z]\)', string_to_strip)
        colon_matches = re.findall(':[A-Za-z]', string_to_strip) +
re.findall(
            '[A-Za-z]:', string_to_strip)
        semicolon_matches = re.findall(';[A-Za-z]', string_to_strip) +
re.findall(
            '[A-Za-z];', string_to_strip)
        # combine all lists to capture all matched strings where any symbols
should be replaced
        all_matches = list(parenth_open_matches + parenth_close_matches +
colon_matches +
                           semicolon_matches)
        # Now replace just the symbols, not the numbers, in these matched
strings
        for match in all_matches:
            for char in chars_to_remove:
                string_to_strip = string_to_strip.replace(match,
match.replace(char, ''))
        df.loc[row,column] = string_to_strip
def strip_emoji_like_if_spaces(df, column):
    """Deletes the symbols :;() where they appear surrounted entirely by
spaces i.e. ' ( '.
        Replaces these symbols with nothing: ''. Leaves them anytime they
don't appear adjacent to numbers.
    Args:
        df = dataframe with the strings to be stripped
        column: name of a column from the dataframe where strings are that
should be stripped

    Returns: same string in the same position within the df, with ;:()
removed in the fashion described above.
    """
    remove_if_spaces = [' ( ', ' ) ', ' : ', ' ; ']
    for char in remove_if_spaces:
    # Replace each character surrounded by spaces with just a single space
        df[column] = df[column].str.replace(char, '', regex=False)
def strip_emoji_sym_adjacent_number(df, column):
    """Deletes the symbols :;() where they appear next to numbers.
        Replaces these symbols with nothing: ''. Leaves them anytime they
don't appear adjacent to numbers.
    Args:
        df = dataframe with the strings to be stripped
        column: name of a column from the dataframe where strings are that
should be stripped

    Returns: same string in the same position within the df, with ;:()
removed in the fashion described above.
    """
    chars_to_remove = ['(', ')', ':', ';']
    # Isolate 1 narrative at a time
    for row in tqdm(range(len(df))):
        string_to_strip = df.loc[row,column]
        # Match the ( where it appears before or after a number.
        parenth_open_matches = re.findall('\([0-9]', string_to_strip) +
re.findall(
            '[0-9]\(', string_to_strip)
        # Match remaining symbols
        parenth_close_matches = re.findall('\)[0-9]', string_to_strip) +
re.findall(
            '[0-9]\)', string_to_strip)
        colon_matches = re.findall(':[0-9]', string_to_strip) + re.findall(
            '[0-9]:', string_to_strip)
        semicolon_matches = re.findall(';[0-9]', string_to_strip) +
re.findall(
            '[0-9];', string_to_strip)
        # combine all lists to capture all matched strings where any symbols
should be replaced
        all_matches = list(parenth_open_matches + parenth_close_matches +
colon_matches +
                           semicolon_matches)
        # Now replace just the symbols, not the numbers, in these matched
strings
        for match in all_matches:
            for char in chars_to_remove:
                string_to_strip = string_to_strip.replace(match,
match.replace(char, ''))
        df.loc[row,column] = string_to_strip
```

Some symbols were removed individually rather than in batches with the functions above because I needed to do a bit more exploration into whether they were likely important to keep or not. I already operated under the assumption that emoji and punctuation like !!! would be good to keep but other symbols like simple periods and commas would be good to remove. But what about = or +? I could imagine somebody giving a drug an A+ or writing “this drug = perfection.” Yet did people actually write things like this, or are these symbols just more noise that should be removed? After exploring several of the reviews where each symbol in question appeared, I decided on the following:

- Keep numbers
- Keep ! $ + = ? %
- Delte #
- Delete if surrounded by spaces or adjacent to numbers rather than adjacent to another symbol: ( ) : ;

After removing symbols, I was ready to standardize words in the text. This way, even if psychedelic experience reports did not contain the exact same words as the psych med reviews, the texts would still be comparable. One key step here was correcting spelling of every text as well as possible.

```
# First, find a misspelling to compare before and after correction
spell = SpellChecker()
misspelled = spell.unknown(df.loc[0,'review'].split(' '))
misspelled
# Output:
{'',
 '20mg',
 '30mg',
 'moodswings',
 'nausiea',
 'noticably',
 'perscribed',
 'shortterm',
 'stabalize',
 'vyvanse'}
# And try the spell-checking on a subset of data, since it takes a long time
mini_df = df.iloc[0:3,:].copy()
mini_df['spell_corr'] = [str(TextBlob(text).correct()) for text in
mini_df['review']]
# How well did this deal with misspellings?
spell = SpellChecker()
misspelled = spell.unknown(mini_df.loc[0,'spell_corr'].split(' '))
misspelled
# Output:
{'', '20mg', '30mg', 'moodswings', 'stabalize', 'vyvanse'}
# Improvement!
```

Next, the text was made lowercase and lemmatized. Lemmatization is similar to taking the root or stem of each word, except that every word remains a real word. For example, the root of “excitement” might be “excite,” and the root of “excitable” might be “excit,” but the lemma of both “excitement” and “excitable” could be excite. After lemmatization, sentences sound funny but are comprehensible, and the total number of unique lemmas in a text will be smaller than the total number of unique pre-lemmatized words, which reduces noise and improves AI model performance.

One final way of reducing noise in a text: remove stopwords. Stopwords are the most common words in any given language; some English stopwords are “the,” “it,” or “no.” For my purposes, it would not have been wise to remove all stopwords because some have a lot to do with whether somebody is speaking favorably or unfavorably about their drug experience. I used a library called spacy to simultaneously lemmatize text and remove stopwords, after revising the default stopword list.

```
nlp = spacy.load('en_core_web_sm')
```

```
stopwords = spacy.lang.en.stop_words.STOP_WORDS
len(nlp.Defaults.stop_words)
# Output: 326
nlp.Defaults.stop_words -= {'against', 'all', 'alone', 'always', 'any',
'anyone', 'anything', 'cannot', 'each', 'empty', 'ever', 'every', 'everyone',
'everything', 'few', 'first', 'least', 'less', 'many', 'most', 'mostly',
'much', 'never', 'no', 'not', 'nothing', 'nowhere', 'not', 'really',
'serious', 'whatever'}
len(nlp.Defaults.stop_words)
# Output: 296
# Now working with appropriate stopword list. Ready for lemmatization with
stopword removal.
df['no_stops_lemm'] = df.spell_corr.apply(lambda text: " ".join(token.lemma_
for token in nlp(text) if not token.is_stop))
# Make everything lowercase
df['no_stop_cap_lemm'] = df.no_stops_lemm.str.lower()
```

Here's a snippet of what one of the reviews looked like in the beginning:

```
"I had began taking 20mg of Vyvanse for three months and was
surprised to find that. . . ."
```

And what it looked like after all the natural language processing described above:

```
"begin take 20 mg Vyvanse month surprised find. . . .”
```

Similar text cleaning techniques were used with both datasets in order to compare like with like when it comes time to use a model trained on psych med reviews to predict ratings for psychedelic experience reports.

Feature Engineering

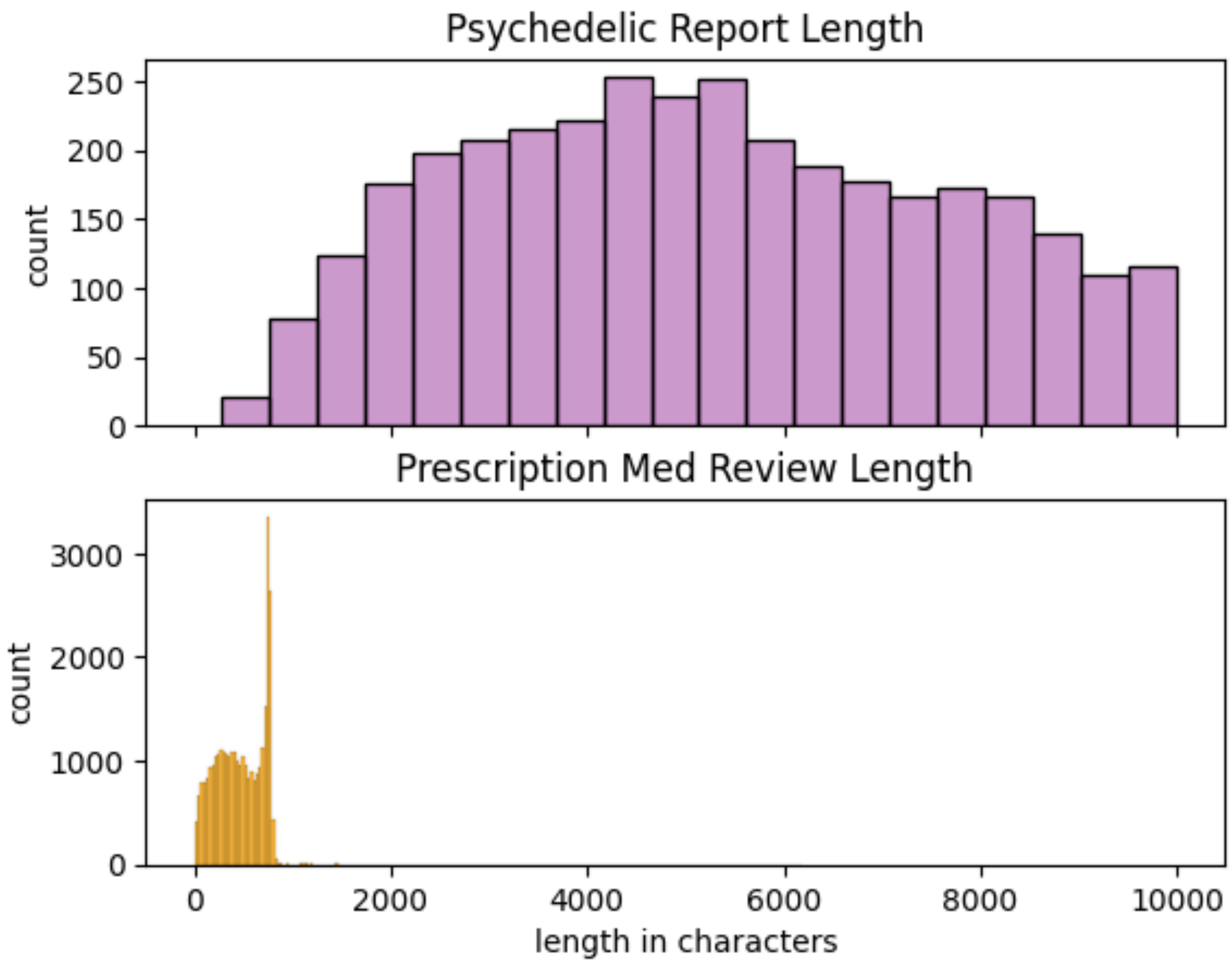
Feature engineering is probably a data scientist's most powerful tool for optimizing models and deriving the most relevant insights possible from our entire process. Sometimes, important information exists within our data already, but not in a format that a machine learning algorithm will pick up on.

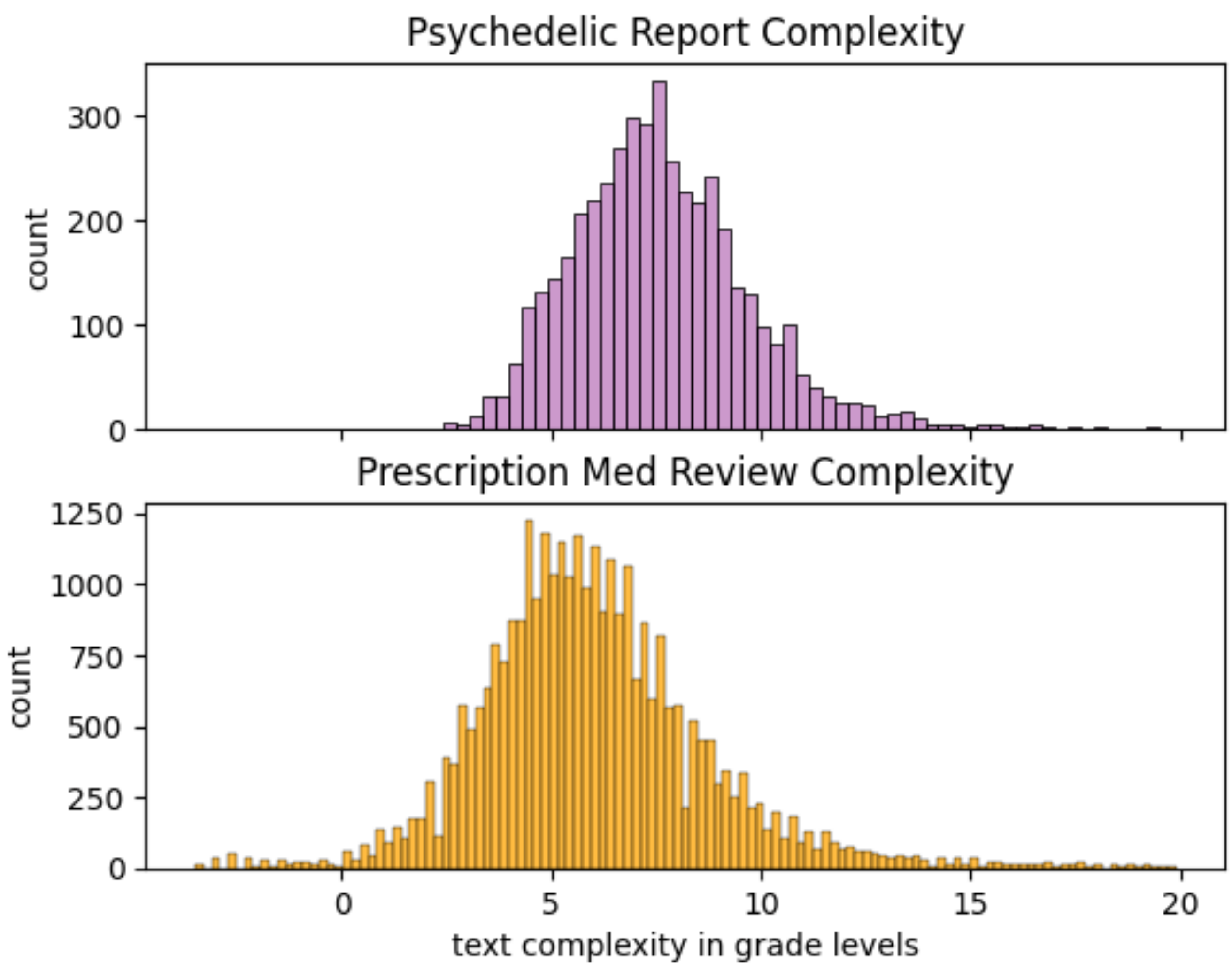
For example, every text in my data has a character length and a complexity based on average word and sentence lengths that can be measured and assigned a “grade level.” But the computer doesn't know this unless I add a column “length” where I count the number of characters in each text and a column “complexity” where I use a pre-existing algorithm to assign a reading level to each text.

A glimpse at how I created these features:

```
# Add length column.
df['review_len'] = [len(df.loc[row,'review']) for row in range(len(df))]
# Add a complexity column.
df['complexity'] = df['review'].apply(lambda x : flesch_kincaid_grade(x))
df.head()
```

Visualizing the results of having engineered these features for each dataset:



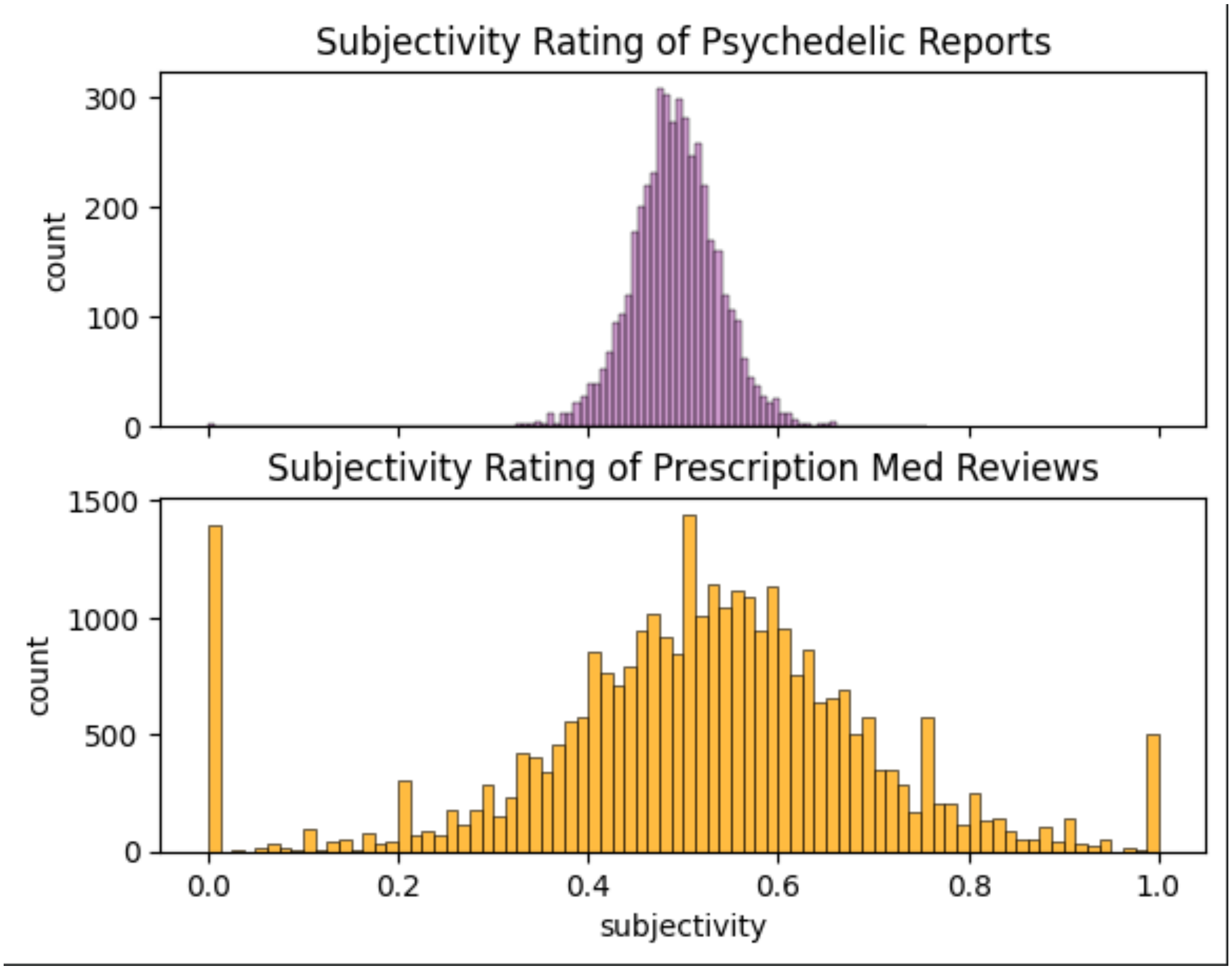


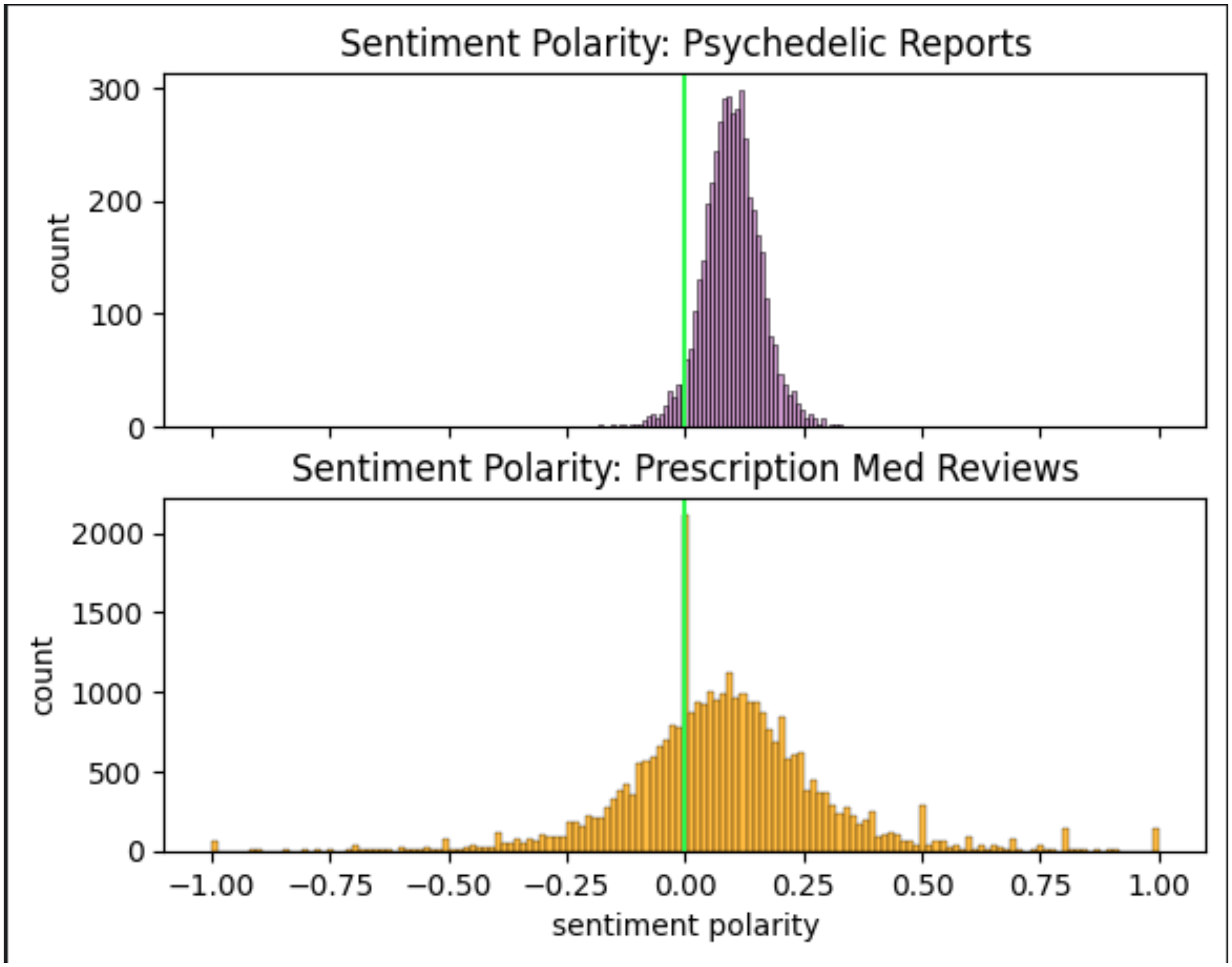
Another couple of key features I created have to do with what is called "subjectivity," how objective vs subjective a text is based on the emotional charge of words used, and "sentiment polarity," or the positive or negative tone of a text. Fortunately, these features were simple to create with the help of a library called TextBlob, which has pre-trained algorithms which can analyze a text and assign it a subjectivity and polarity score.

The code:

```
def get_subjectivity(text):
    return TextBlob(text).sentiment.subjectivity
def get_polarity(text):
    return TextBlob(text).sentiment.polarity
def analyze_sentiment(df, column):
    """Engineers subjectivity and polarity features for each string in a
column.
    Args:
        df: dataframe that contains a column of strings
        column: name of column full of strings
    Returns: a new column added to the dataframe that contains a subjectivity
score between 0 and 1
        and a column that contains polarity score between -1 and 1 (negative
or positive sentiment)
        associated with the text of each string.
    """
    df['subjectivity'] = df[column].apply(get_subjectivity)
    df['polarity'] = df[column].apply(get_polarity)
```

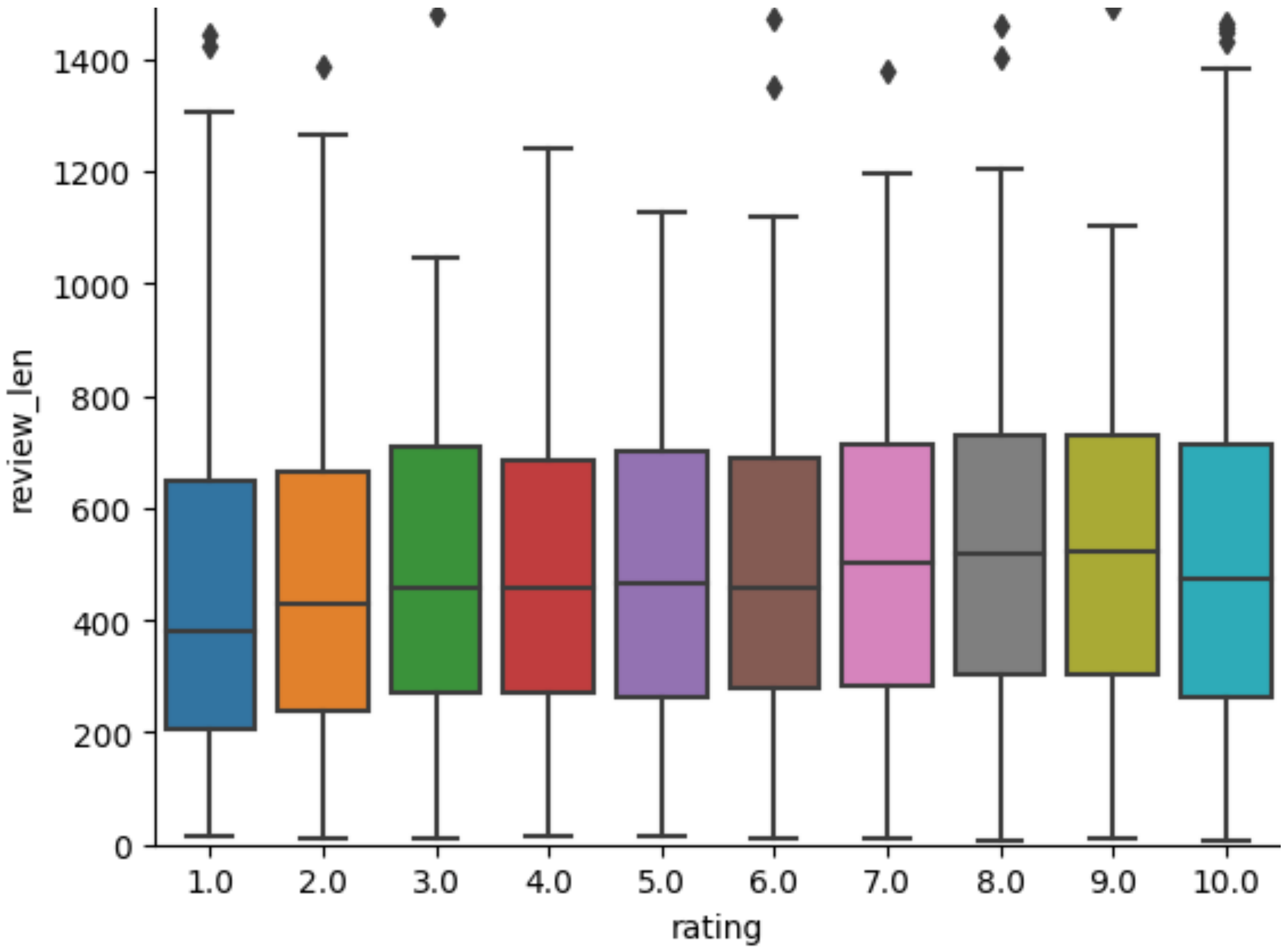
Subjectivity and sentiment polarity vary across the two datasets:



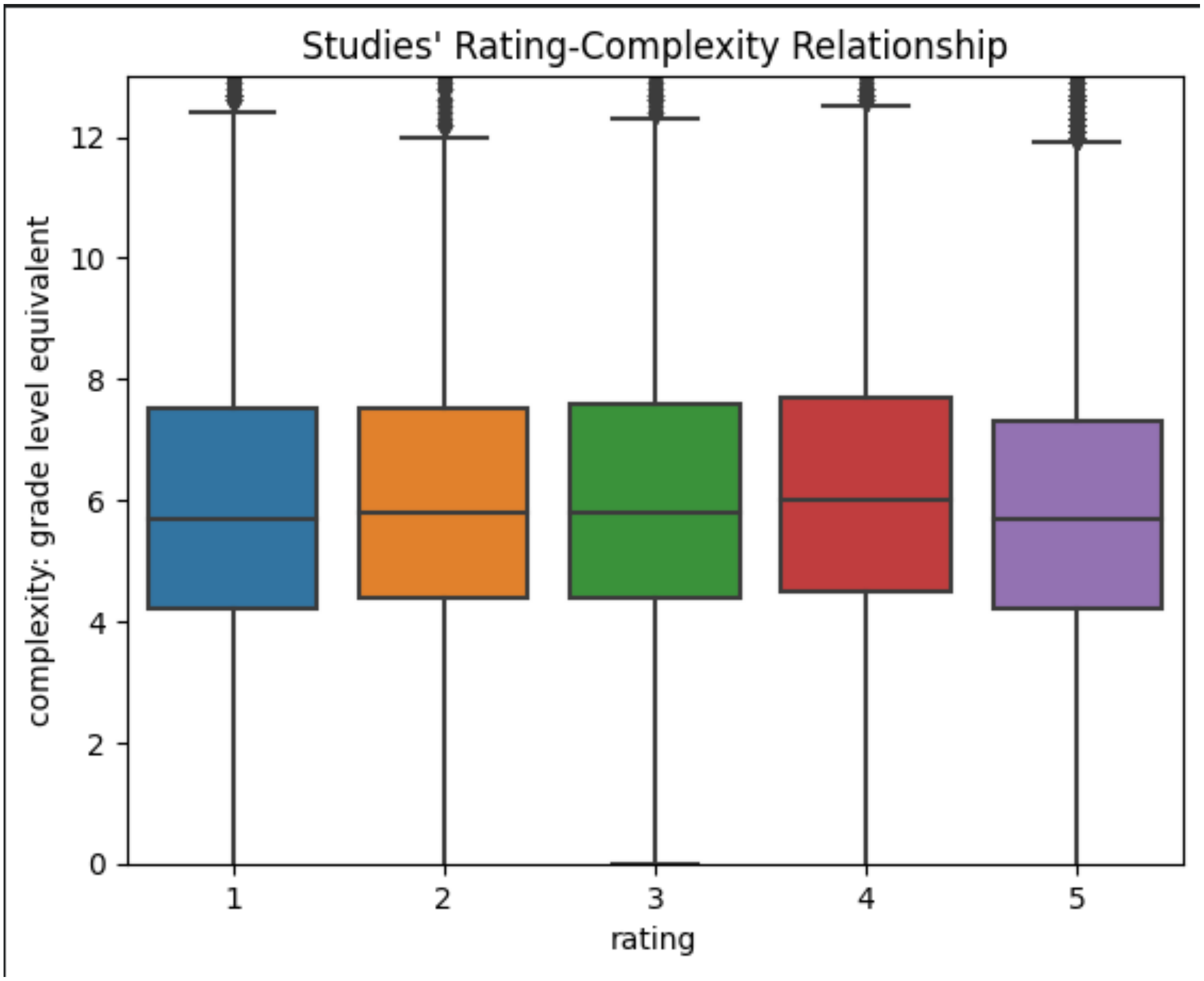


Not all features are equally important. A feature's correlation with the target variable "rating" from the psych med review data will determine how heavily the ultimate machine learning model will depend upon that variable when making its predictions. As seen in the plots below, sentiment polarity is, predictably, the most highly correlated with rating and, therefore, the most valuable feature.

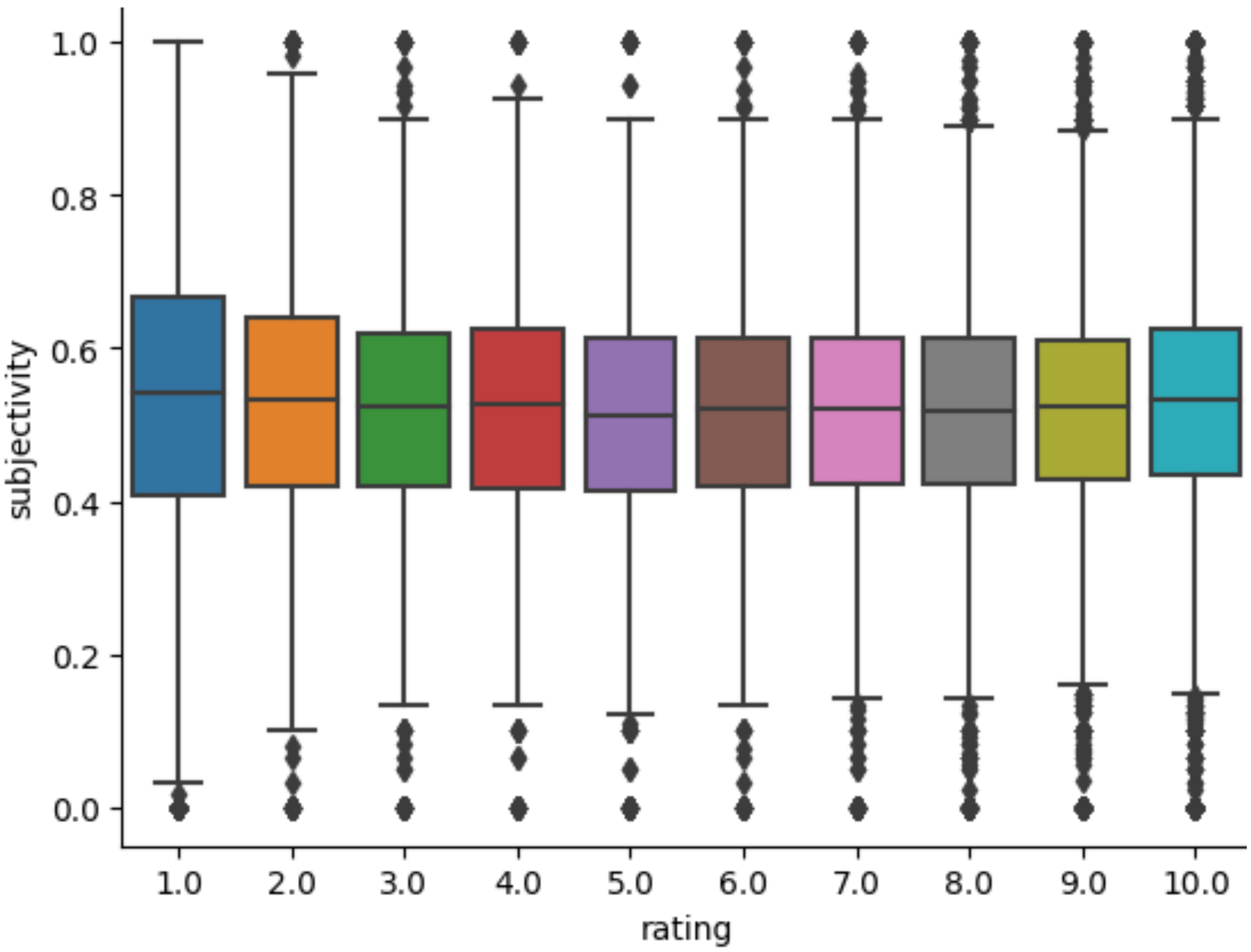
Shorter reviews are very slightly more likely to be associated with bad ratings.



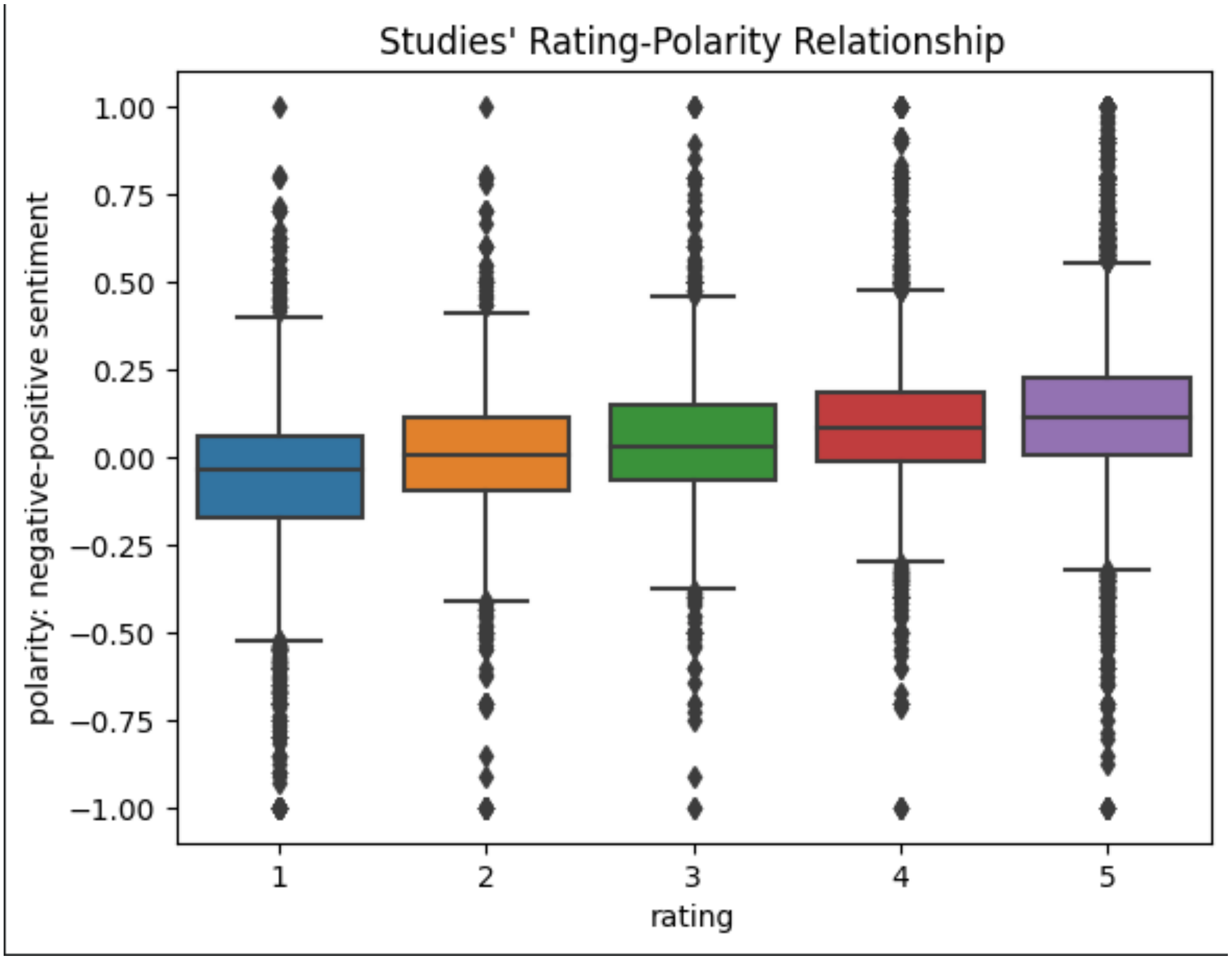
A less sophisticated level of text complexity is slightly more likely to be associated with high ratings.



Subjectivity varies slightly more widely among negative reviews, and more extremely positive or negative reviews are very slightly less objective in tone.



An increasingly positive tone or sentiment is clearly associated with higher ratings.



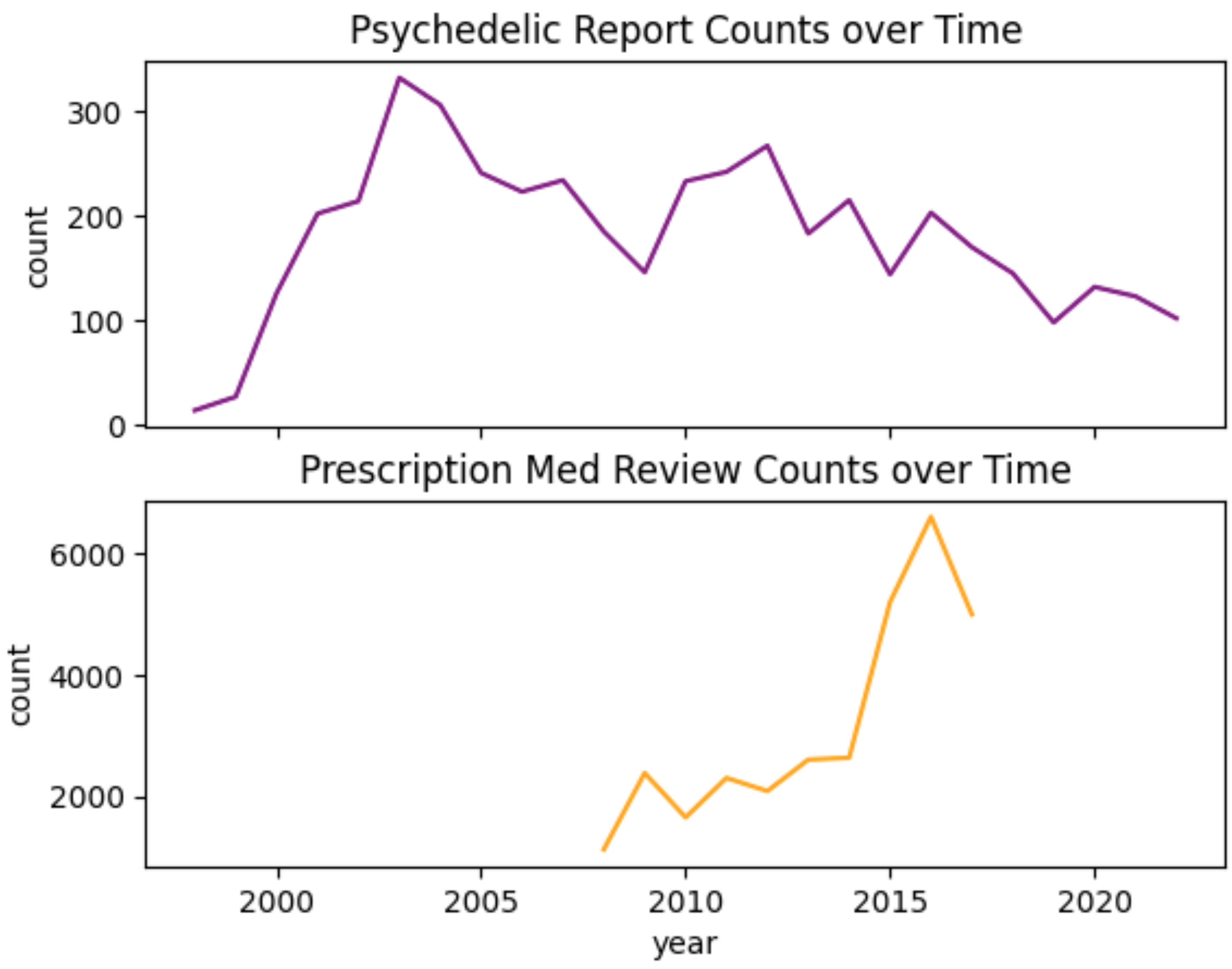
Finally, the full text itself needs to be made comprehensible to a computer. This can be done in a number of ways. One simple way is to simply count up the number of each word in each text and pass that as a matrix to the machine learning algorithm. For this I used a tool called CountVectorizer. I included this step in the creation of the machine learning model, so I'll share code for it further below.

Exploratory Data Analysis (EDA)

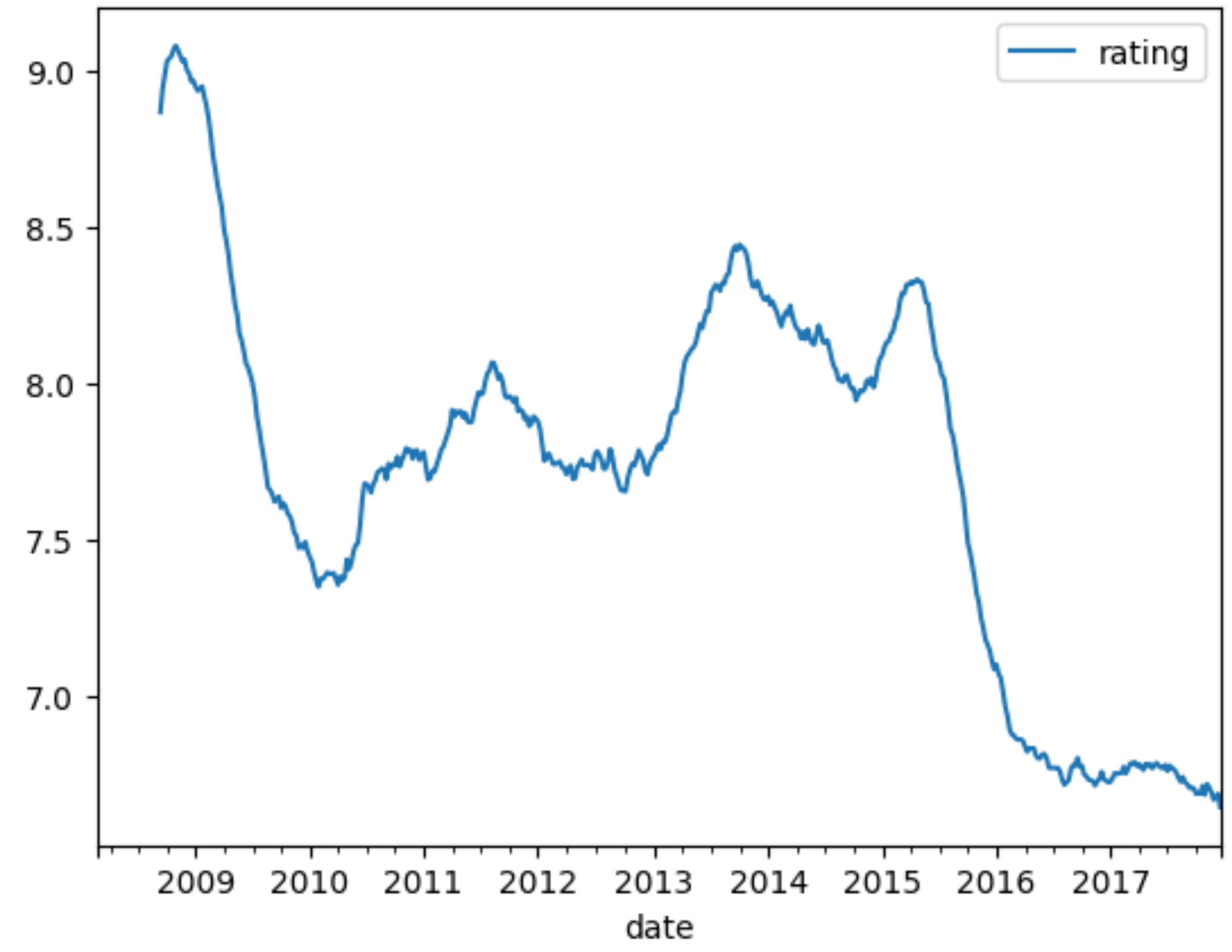
EDA for this project was tons of fun! This is where I dove in to learn as much as possible about all the data I gathered and make the most of it. It appears in this blog after feature engineering because it lasted right up until I was about to create my machine learning model, but I started conducting EDA immediately upon initially wrangling the first dataset. Some EDA I already shared in the feature engineering section above. Other investigations weren't particularly relevant to my ultimate goal of assigning ratings to psychedelic experience reports, but everything I learned along the way was fascinating, nevertheless.

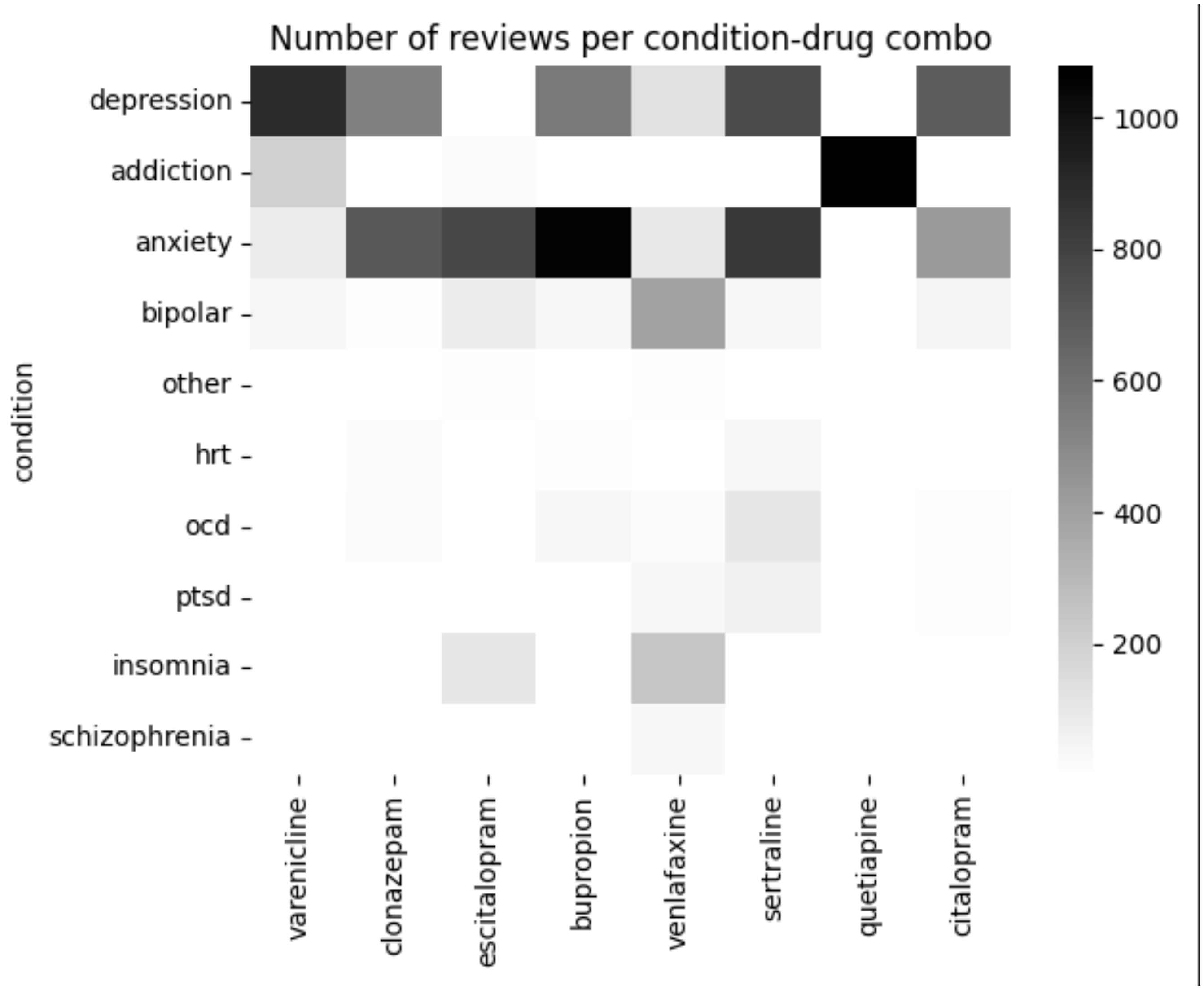
Here are some visuals representing interesting findings. Do you see anything that has you asking more questions than I answer here? There are some great ideas for future projects waiting to happen.

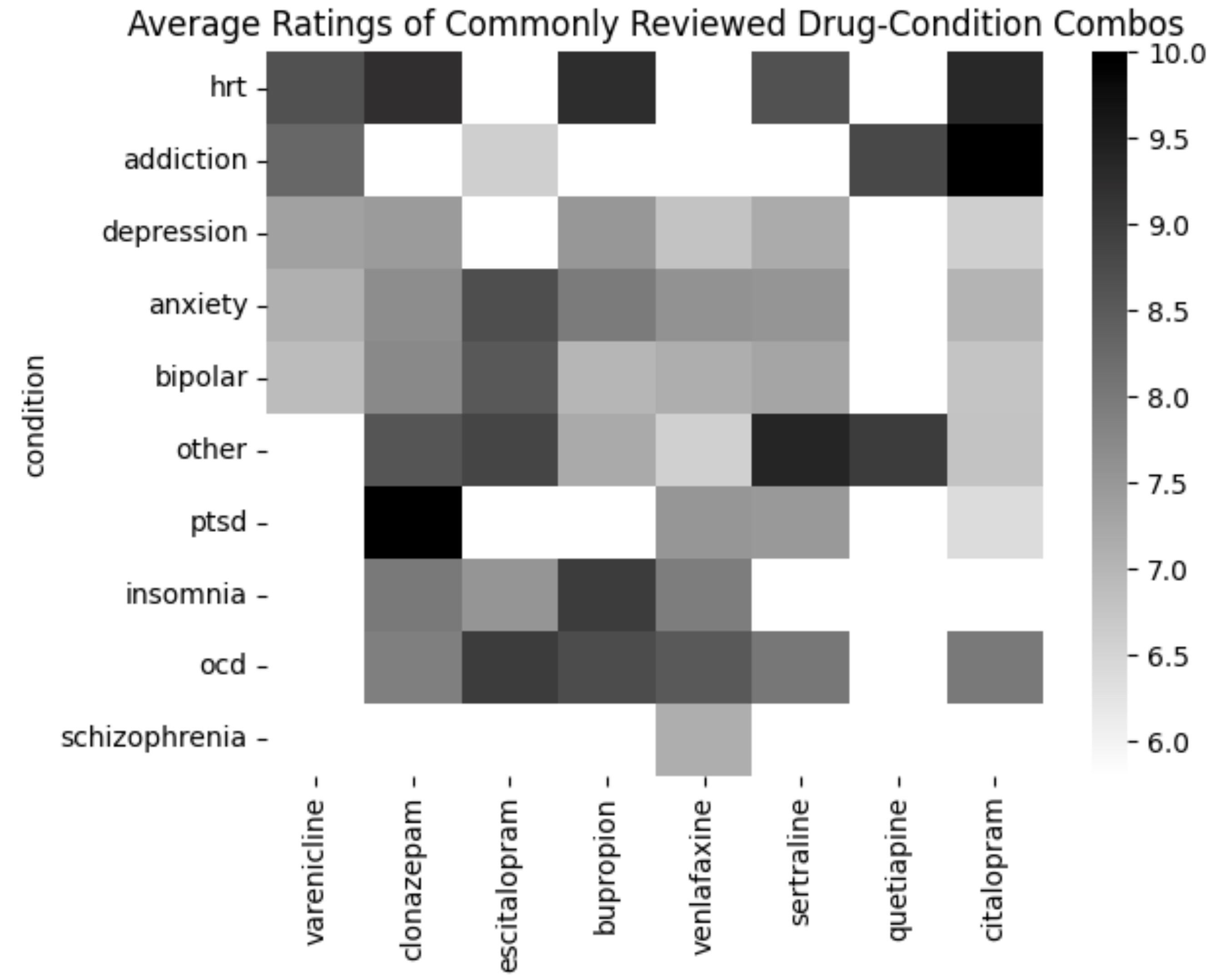
This chart just shows some context for the two datasets and when the reviews/reports were submitted for each:



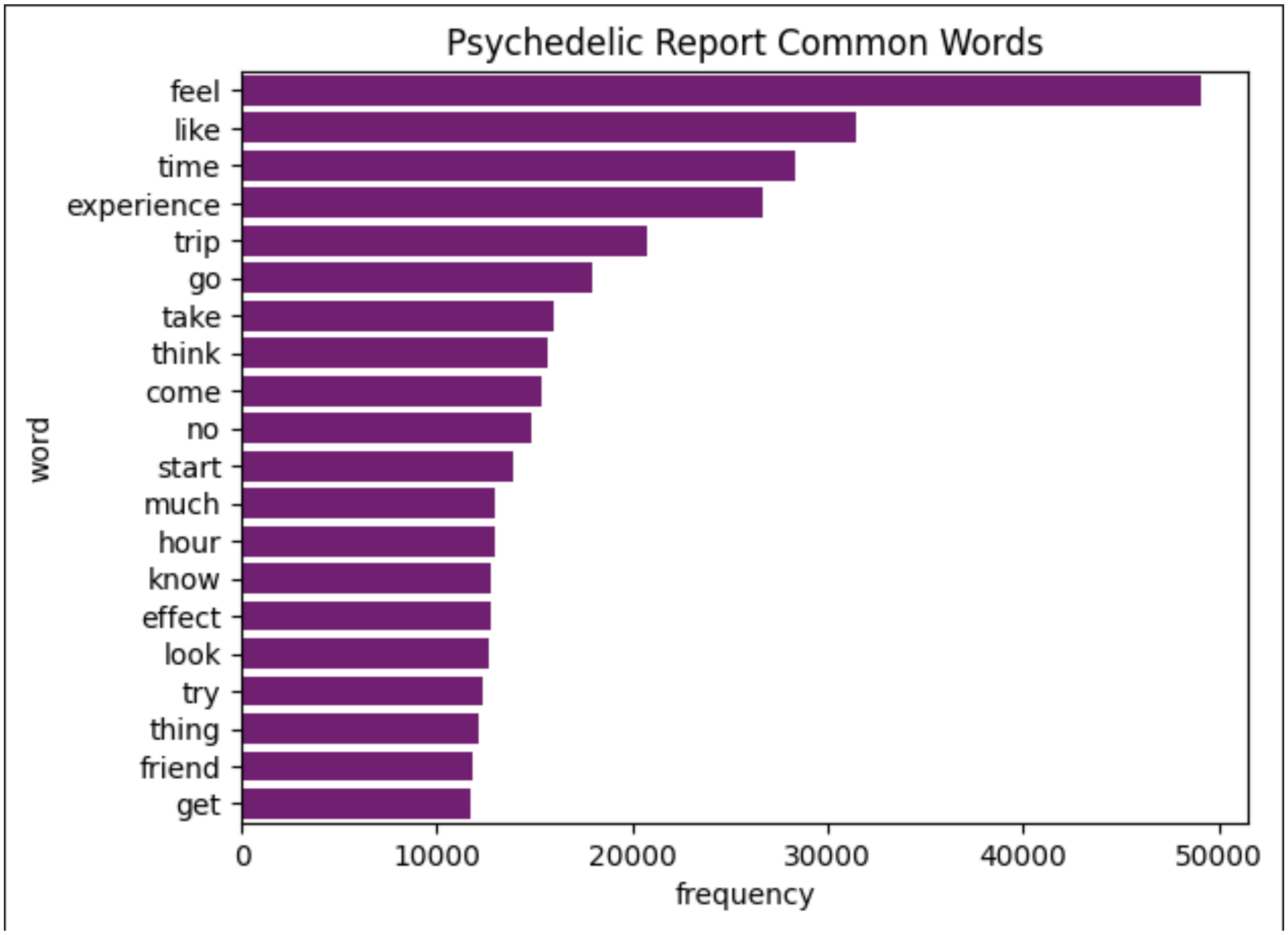
Some extraneous but interesting info that emerged from the original psych med review data:

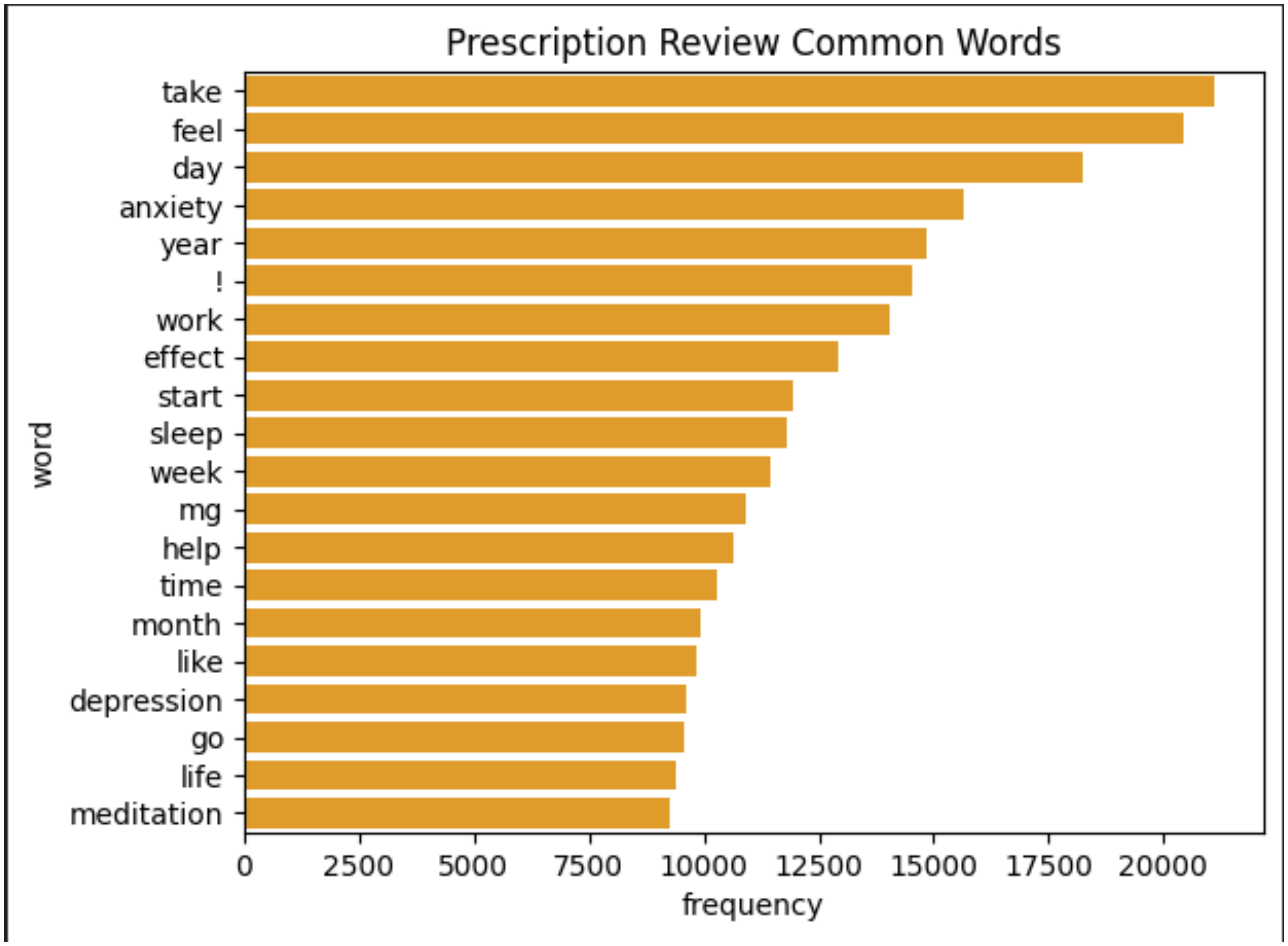

rating
9.0
8.5
8.0
7.5
7.0
2009
2010
2011
2012
2013
2014
2015
2016
2017
date

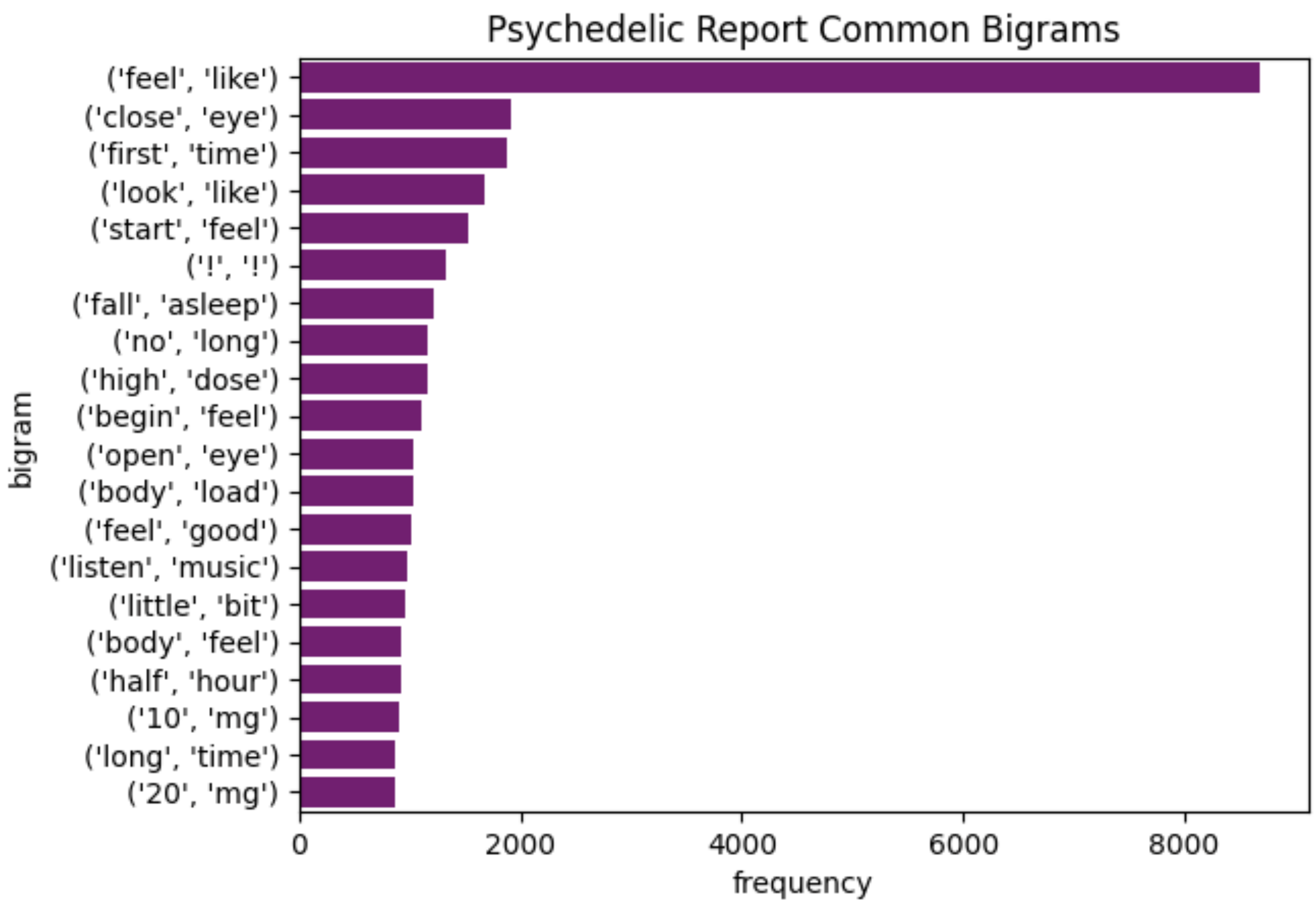

Number of reviews per condition-drug combo
condition
depression
addiction
anxiety
bipolar
other
hrt
ocd
ptsd
insomnia
schizophrenia
varenicline
clonazepam
escitalopram
bupropion
venlafaxine
sertraline
quetiapine
citalopram
1000
800
600
400
200

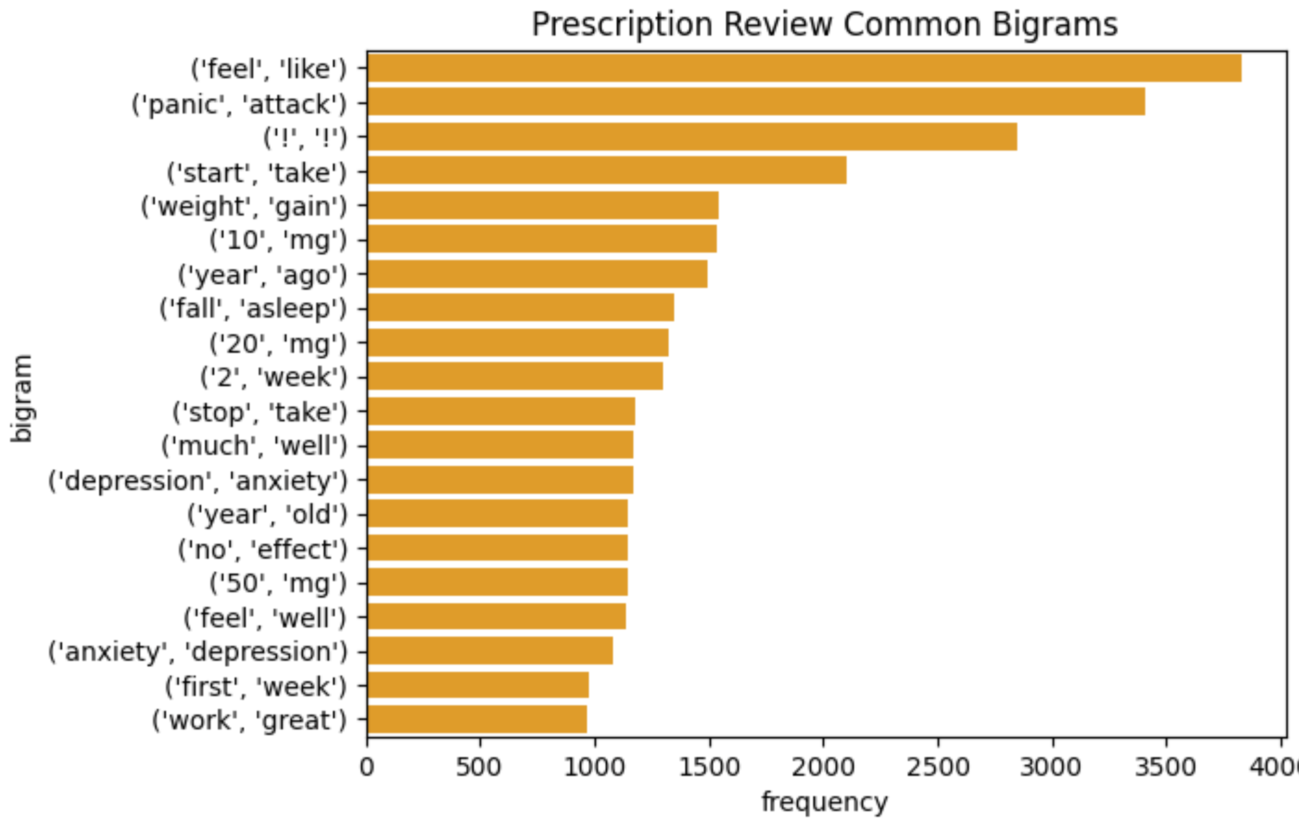


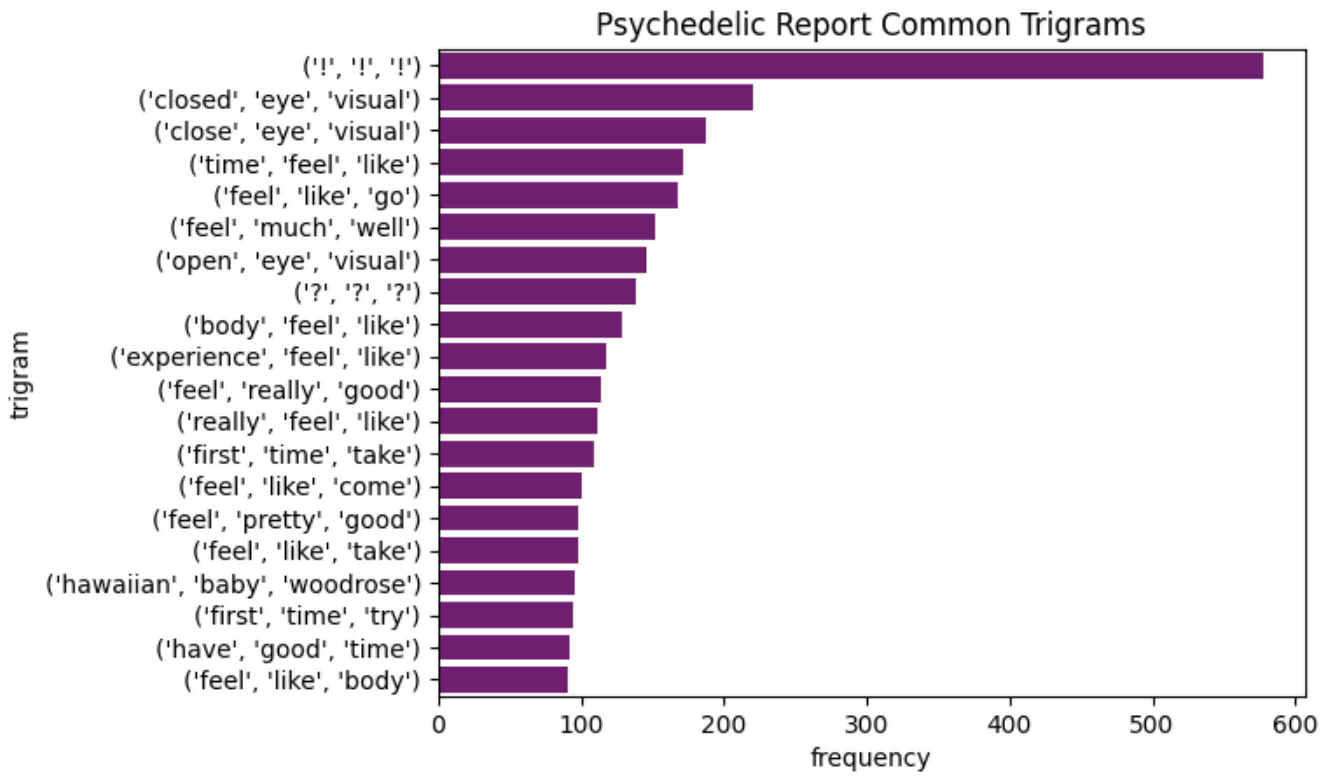
N-grams are a term to describe phrases containing n number of words. It’s pretty interesting comparing the most common words and 2- or 3-word phrases from the two datasets.

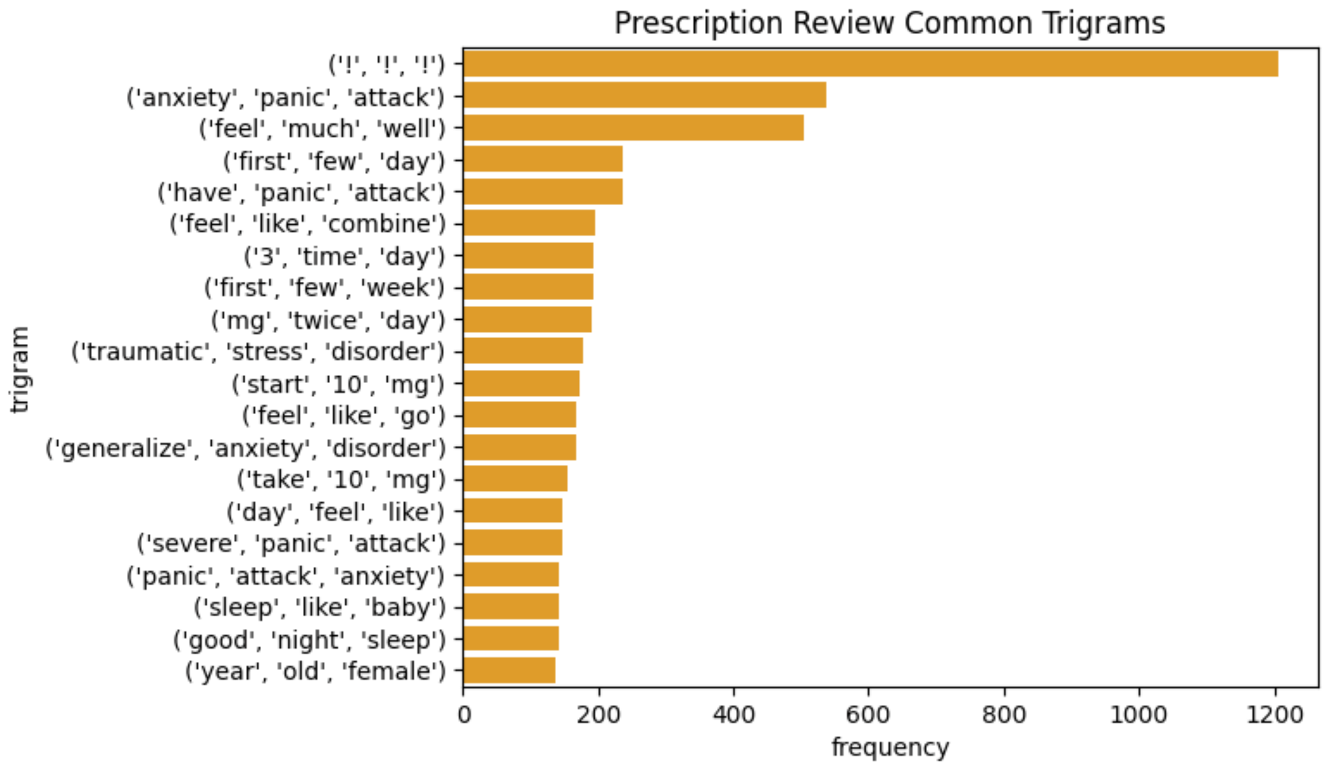

Psychedelic Report Common Words
word
frequency
feel
like
time
experience
trip
go
take
think
come
no
start
much
hour
know
effect
look
try
thing
friend
get
0
10000
20000
30000
40000
50000


Prescription Review Common Words
take
feel
day
anxiety
year
!
work
effect
start
sleep
week
mg
help
time
month
like
depression
go
life
meditation
word
0
2500
5000
7500
10000
12500
15000
17500
20000
frequency


Psychedelic Report Common Bigrams
('feel', 'like')
('close', 'eye')
('first', 'time')
('look', 'like')
('start', 'feel')
('!', '!')
('fall', 'asleep')
('no', 'long')
('high', 'dose')
('begin', 'feel')
('open', 'eye')
('body', 'load')
('feel', 'good')
('listen', 'music')
('little', 'bit')
('body', 'feel')
('half', 'hour')
('10', 'mg')
('long', 'time')
('20', 'mg')
bigram
0
2000
4000
6000
8000
frequency


Prescription Review Common Bigrams
('feel', 'like')
('panic', 'attack')
('!', '!')
('start', 'take')
('weight', 'gain')
('10', 'mg')
('year', 'ago')
('fall', 'asleep')
('20', 'mg')
('2', 'week')
('stop', 'take')
('much', 'well')
('depression', 'anxiety')
('year', 'old')
('no', 'effect')
('50', 'mg')
('feel', 'well')
('anxiety', 'depression')
('first', 'week')
('work', 'great')
bigram
0
500
1000
1500
2000
2500
3000
3500
4000
frequency





Modeling

This is where all the hard work paid off and I was able to train the model that I'd eventually use to predict ratings that people might have assigned to their psychedelic drugs given the contents of their narrative reports.

Before diving into modeling, I needed to decide whether or not to keep all the features I engineered or if some of them were co-correlated/redundant and therefore noise that would reduce model performance. I chose to keep all the columns intact and not perform any sort of feature reduction.

There are many machine learning algorithms available. Each has strengths and weaknesses for working with different types of data, and this was another choice I needed to make. The code I used to make these decisions is here:

```
# Make variables for types of columns & instantiate what to do with each
review_col = 'review'
numeric_cols = ['review_len', 'complexity', 'subjectivity',
'original_polarity', 'similarity_w_10']
num_trans = Pipeline(steps=[('pca', PCA(random_state=17)), ('mms',
MinMaxScaler()), ('si', SimpleImputer())])
text_trans = Pipeline(steps=[('cv', CountVectorizer(lowercase=False)), ('si',
SimpleImputer())])
# Build column transformer
cols = ColumnTransformer([('num', num_trans, numeric_cols),
                          ('text', text_trans, review_col)])
# Create parameters for PCA, CountVectorizer
pca_cv = {'cols__num__pca__n_components':[1,2,3,5],
              'cols__text__cv__ngram_range':[(1,1), (1,2), (1,3)],
              'cols__text__cv__analyzer':['word', 'char']}
# List potential parameters for each classifier
# Use SVC with OneVsRest and bagging to speed it up. kernel=linear for speed
just for now.
svc_params = {'clf':(OneVsRestClassifier(estimator=BaggingClassifier(
    estimator=SVC(probability=True, random_state=17, kernel='linear'),
    max_samples=0.05, bootstrap=False, n_jobs=-1, n_estimators=1)),),
              'clf__estimator__estimator__C':list(np.arange(1,11)),
              'clf__estimator__estimator__degree':list(np.arange(1,11)),
              'clf__estimator__estimator__gamma':['scale', 'auto'],
              'clf__estimator__estimator__coef0':list(np.arange(0,4,0.5)),
              'clf__estimator__estimator__shrinking':[True,False],
              'clf__estimator__estimator__probability':[True,False],
              'clf__estimator__estimator__class_weight':[None,'balanced']}
svc_params.update(pca_cv)
# ComplementNB is a beysian classifier I read is good for text & imbalanced
classes
cnb_params = {'clf':(ComplementNB(),), 'clf__alpha':[0.01, 0.1, 0.5, 1, 2, 5,
10]}
cnb_params.update(pca_cv)
# Also try MultinomialNB, often used for text classification
mnb_params = {'clf':(MultinomialNB(),), 'clf__alpha':[0.01, 0.1, 0.5, 1, 2,
5, 10], 'clf__fit_prior':[True,False]}
```

```
mnb_params.update(pca_cv)
# Only select XGB hyperparameters that I've read minimize overfitting, a
problem w/ trees
xgb_params = {'clf':(XGBClassifier(),),
'clf__colsample_bytree':list(np.arange(0.1, 0.6, 0.1)),
              'clf__subsample':list(np.arange(0, 0.6, 0.1)),
              'clf__max_depth':list(np.arange(1,5)),
'clf__gamma':list(np.arange(4,11,1)),
              'clf__eta':list(np.arange(0, 0.6, 0.1)),
              'clf__min_child_weight':[5,20,50,100,200], 'clf__alpha':
[5,10,20,50,100], 'clf__n_estimators':[5,10,20,50]}
xgb_params.update(pca_cv)
# Prepare scorers
top_k = make_scorer(score_func=top_k_accuracy_score, k=1, needs_proba=True)
scorers = {'top_k':top_k, 'f1':'f1_micro', 'neg_log_loss':'neg_log_loss',
           'roc_auc':'roc_auc_ovr'}
# Build final pipeline, param grid & instantiate gridsearch
pipe = Pipeline(steps=[('cols', cols), ('clf', DummyClassifier())])
param_grid = [cnb_params, mnb_params, xgb_params, svc_params]
rgs = RandomizedSearchCV(estimator=pipe, param_distributions=param_grid,
scoring=scorers, refit='roc_auc', error_score='raise', random_state=4)
# Run the gridsearch
#rgs.fit(X_train, y_train)
#print(rgs.best_params_)
#print(rgs.best_score_)
# Output:
{'cols__text__cv__ngram_range': (1, 1), 'cols__text__cv__analyzer': 'word',
'cols__num__pca__n_components': 5, 'clf__alpha': 1, 'clf':
ComplementNB(alpha=1)} 0.7257045188386937
```

The “best learner” was a machine learning algorithm called ComplementNB. I carried on to train this algorithm and create the final model. The entire data science process is called a pipeline, but we also use pipelines in code to take the training of a machine learning model step by step. My modeling pipeline steps were as follows:

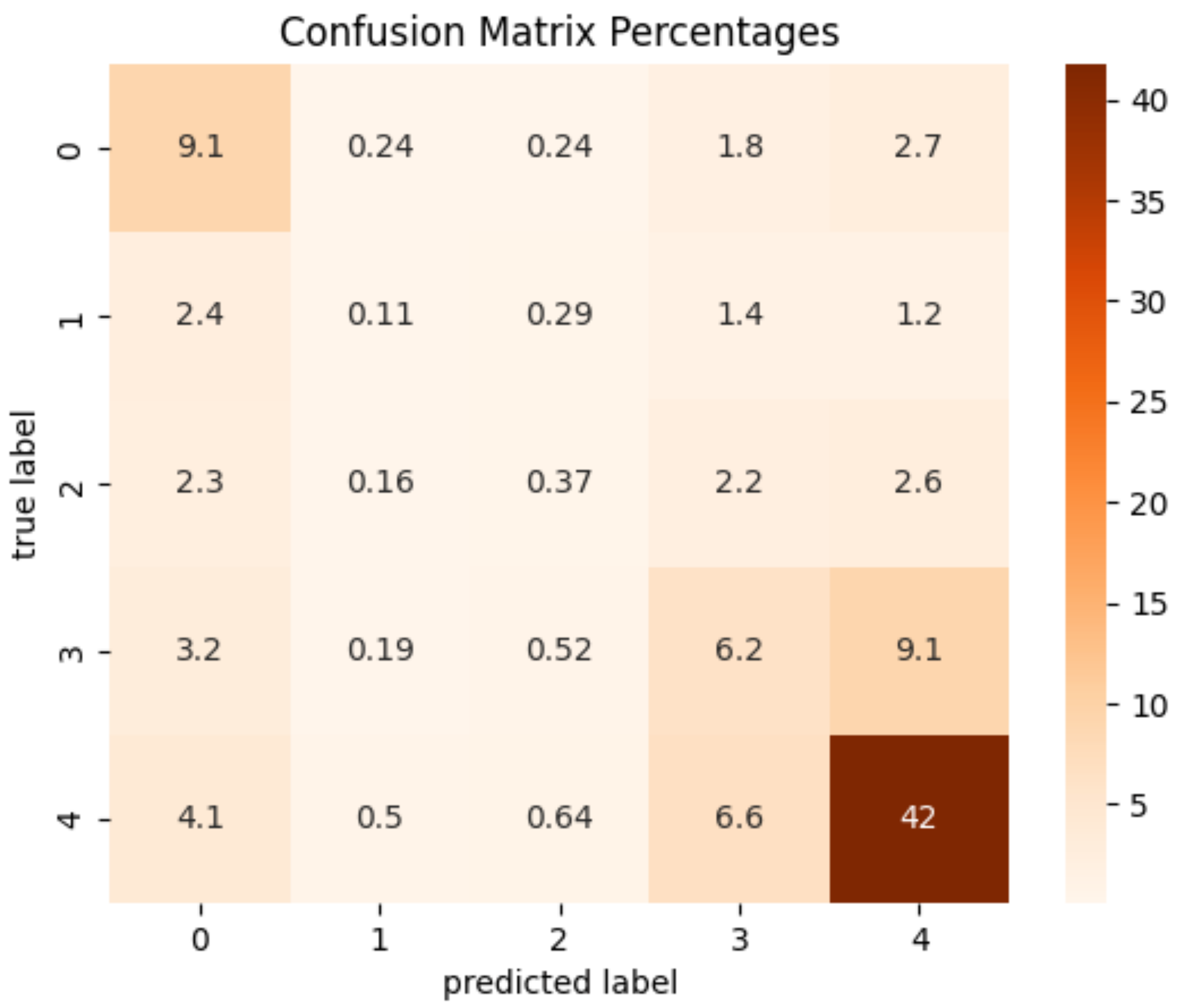
1. Normalize/standardize numeric columns with MinMaxScaler. Machine learning algorithms can mis-assign importance to a feature if its scale is out of whack with other features. For example, in my data, the character length of each review ranged from 3 to thousands. But the polarity of every single review was between -1 and 1. MinMaxScaler shifted all the numbers to fall between 0 and 1.
2. Turn the text into a sparse matrix of word counts with CountVectorizer.
3. Train the algorithm to decide how to assign ratings based on the data provided in steps 1 & 2. Do this with a “train set,” which is all the data except some which is set aside in order to test how well the model does with making predictions on unseen data.
4. Evaluate the efficacy of the model by seeing how accurately it assigned ratings to reviews in the “test set.” I used several metrics to evaluate the quality of the model, many of which are not so simple to explain in plan language. But I’ll share a bit below

about how well I think my model makes predictions.

Code for this modeling pipeline:

```
review_col = 'review'
numeric_cols = ['review_len', 'complexity', 'subjectivity',
'original_polarity',
                'similarity_w_10']
num_trans = MinMaxScaler()
text_trans = CountVectorizer(lowercase=False)
cols = ColumnTransformer([('num', num_trans, numeric_cols), ('text',
text_trans, review_col)])
clf = ComplementNB()
pipe = Pipeline(steps=[('cols', cols), ('clf', clf)])
pipe.fit(X_train, y_train)
proba = pipe.predict_proba(X_test)
pred = pipe.predict(X_test)
print('log loss: ', log_loss(y_test, proba))
print('roc_auc: ', roc_auc_score(y_test, proba, average='weighted',
multi_class='ovr'))
print('f1 (or accuracy best k w/ k=1): ‘, top_k_accuracy_score(y_test, proba,
k=1))
print('best k w/ k=2: ', top_k_accuracy_score(y_test, proba, k=2))
print(confusion_matrix(y_test, pred))
# Output:
log loss:  1.2726915117196869
roc_auc:  0.7518495541863638
f1 (or accuracy best k w/ k=1):  0.5754119138149556
best k w/ k=2:  0.7316223067173637
# Pickle the trained model for later use
with open('../models/trained_model.pkl','wb') as f:
    pickle.dump(pipe,f)
```

A confusion matrix shows the results of modeling. What did the model predict, and how does that compare with the actual rating the review’s author gave any given drug?



The labels here run from 0-4 instead of from 1-5, because that's how the computer needs to make its predictions. We can see that most people rated their drugs highly, and the model accurately predicted that most of the reviews should be given a high rating. 42% of the ratings were accurately predicted to be 4/4. 6% were accurately predicted to be 3/4. Less than 1% each were accurately predicted to be 2/4 and 1/4. 9% were accurately predicted to be 0/4.

One of my evaluation metrics was "best k with k=2." This means that the model made a prediction for what rating to assign a review, and it tells us what it thinks is its next-best guess. 73% of the time, the model makes an accurate prediction on its first or second try.

There is no single number defining accuracy that makes the model "good enough." There is definitely room for improvement if approximately 25% of the time the model doesn't accurately predict a review's rating even after 2 tries. Rather than disregard this model's usefulness, however, we can take its predictions with a grain of salt and put this accuracy score in a more meaningful context. We can compare these results to those we would get if we just randomly assigned ratings to reviews. Such a "naive model" would only be correct 20% of the time, whereas my model is correct on its first guess 58% of the time.

Deployment

Once I trained the model, it was ready to be applied with all the scraped, processed psychedelic experience reports. I saved the trained model to a special file type called a "pickle," and then I told it to make predictions based on then new text's features. The code for this is actually quite simple:

```
with open("../models/trained_model.pkl", "rb") as f:
model = pickle.load(f)
# Assign ratings
ratings = model.predict(X_unseen)
```

Summary of Findings

The average of all the psychedelic experience reports' predicted ratings was 4.49. This is compared to an actual average rating of 3.93 for prescription psych meds.

That number 4.49 could be off base given the imperfect performance of my model and the fact that not all words were the same between the two datasets. Nevertheless, all this research and analysis does suggest that people may have rated their psychedelic drugs more highly if they had left quantitative ratings along with their experience reports on Erowid.

Resources

See all my code in my repo here.

Or a 3-page technical summary report.

Or a cute powerpoint presentation intended for a general audience.

I used the following resources to support my work throughout this project:

- https://towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72
- https://medium.com/plotly/nlp-visualisations-for-clear-immediate-insights-into-text-data-and-outputs-9ebfab168d5b
- https://www.kaggle.com/code/sainathkrothapalli/nlp-visualisation-guide
- https://www.kaggle.com/code/abhishek/approaching-almost-any-nlp-problem-on-kaggle/notebook

Tags learning, project

Leave a Reply

Enter your comment here...

FRACTAL DATA