Sunny Zhang

501195903

Dr. Ceni Babaoglu

November 3, 2025

CIND 820 - Data Preparation Section

November 10, 2025

1 CIND 820 - Deliverable 3 - Data Preparation

1.1 Data Preparation Section

The purpose of this section of the notebook is to setup our data such that it becomes ready for data analysis and ML model adoption. Since this project is largely text-based, we must engineer some features first before we can begin our analysis. As such, breaking convention, our project will start with the *Data Preparation Section* first before moving into the *Data Analysis Section*.

1. First we need to fix the 'stars' column. It's formatted as Float64 and we want integers for a cleaner grouping later.
2. Second, we need to prepare our 'text' column such that it is formatted in a way that is acceptable for both our VADER-library and Countvectorizer library for the purposes of feature engineering. We need to create 2 versions of the 'text' column': One for the VADER library to read and generate sentiment scores, and another for the Countvectorizer library to create additional Bag-of-Words vectors for our model training. We require two different columns because the level of data pre-processing required for each are different.
3. Third, we'll need to employ stratified sampling to balance out our star-ratings. We will also further reduce the size of our dataset for the purposes of model training.
4. Fourth, we'll engineer VADER-based sentiment scores using the VADER library
5. Then, we'll create Bag-of-Words vectors, using the Countvectorizer library, which will be used as additional features for our model.

The completion of these 5 steps will mark the end of the *Data Preparation Section* as the data will be ready for analysis and model training.

1.1.1 Setting up the Notebook Environment

[2]:
```
import os
os.chdir('C:/Users/Sunora/iCloudDrive/Documents/Learning Data Analytics/TMU␣
↪Certificate copy/CIND 820/Yelp Dataset')
```

[3]:
```
#Importing libraries and setting up environment
import pandas as pd # Used for Data Manipulation
import numpy as np # Used for Numerical Operations
import re # Used for Regular Expressions
import nltk # Natural Language Toolkit
```

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer # Used for␣
↪Sentiment Analysis (VADER)
# nltk.download('vader_lexicon') # Downloading VADER Lexicon
import spacy # Used for Advanced NLP tasks (tokenization, lemmatization, etc.)
nlp = spacy.load('en_core_web_sm') # Loading SpaCy English Model

#Defining working directory and reading data
filepath = r'C:\Users\Sunora\iCloudDrive\Documents\Learning Data Analytics\TMU␣
↪Certificate copy\CIND 820\Yelp Dataset\yelp_review_sample_data.csv'
originalData = pd.read_csv(filepath)

# Previewing the Datatype Info
originalData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 349514 entries, 0 to 349513
Data columns (total 5 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   review_id    349514 non-null  object
 1   user_id      349514 non-null  object
 2   business_id  349514 non-null  object
 3   stars        349514 non-null  float64
 4   text         349514 non-null  object
dtypes: float64(1), object(4)
memory usage: 13.3+ MB
```

1.1.2 1. Fixing Stars Column

[4]:
```
# Step 1. Converting 'stars' column from Float64 to Integer
originalData['stars'] = originalData['stars'].astype(int)
```

1.1.3 2. Text Pre-processing for VADER and Bag-of-Words Libraries

[ ]:
```
# Step 2. Cleaning 'text' data column and and performing text preprocessing

# General cleaning of 'text' column
originalData['text'] = originalData['text'].fillna('').astype(str) # Fill NaN␣
↪values with empty strings and ensure all entries are strings

# Creating a function to perform basic text cleaning - applicable to both VADER␣
↪and Bag-of-Words (BoW)
def text_clean_general(text):
    text = re.sub(r'<.*?>', '', text)  # Remove HTML tags (if any)
    text = re.sub(r'\s+', ' ', text)  # Replace multiple spaces with a single␣
↪space (meaning double-spaces, new lines, tabs, etc.)
    return text.strip() # Remove leading/trailing spaces
```

```
# Applying the function defined above to create a new text column that is␣
 ↪pre-processed for VADER analysis
originalData['text_VADER'] = originalData['text'].apply(text_clean_general) #␣
 ↪Creating a new column 'text_VADER' for VADER analysis

# Loop for cleaning the 'text' column for BoW - to be used on the 'text_VADER'␣
 ↪column as BoW needs more aggressive cleaning
corpus = originalData['text_VADER'].str.lower().tolist() # Converting the␣
 ↪'text_VADER' column to a list for processing; using VADER cleaned text as␣
 ↪base
outputs = [] # List to store cleaned text and token counts - for easier␣
 ↪dataframe conversion
for doc in nlp.pipe(corpus, batch_size=1000, n_process=8): # Using nlp.pipe for␣
 ↪efficient processing (due to size of dataset)
    tokens = [token.lemma_ for token in doc if not token.is_stop and token.
 ↪is_alpha] # Lemmatization, removing stop words and non-alphabetic tokens
    cleaned_text = ' '.join(tokens) # Joining tokens back into a single string
    token_count = len(tokens) # Counting number of tokens after cleaning - an␣
 ↪additional feature
    outputs.append((cleaned_text, token_count)) # Appending cleaned text and␣
 ↪token count as a tuple to outputs list

# Gathering outputs from the loop (above) into the original dataframe as new␣
 ↪columns
originalData[['text_BoW', 'num_Tokens']] = pd.DataFrame(outputs, index =␣
 ↪originalData.index) # text_BoW for Bag of Words cleaned text, num_Tokens for␣
 ↪number of tokens
```

1.1.4 3. Stratified Sampling for Star Rating Balancing

[6]:
```
# Using balancedData to store the new balanced originalDataset based on␣
 ↪stratified sampling - 5000 samples per star rating
balancedData = (
    originalData[originalData['num_Tokens'] > 0]  # Filtering out rows with␣
 ↪zero tokens after cleaning
    .groupby('stars', group_keys=False)
    .apply(lambda x: x.sample(n = 5000, random_state = 2025))
    .reset_index(drop=True)
)
```

```
C:\Users\Sunora\AppData\Local\Temp\ipykernel_20768\1940115590.py:5:
FutureWarning: DataFrameGroupBy.apply operated on the grouping columns. This
behavior is deprecated, and in a future version of pandas the grouping columns
will be excluded from the operation. Either pass `include_groups=False` to
exclude the groupings or explicitly select the grouping columns after groupby to
silence this warning.
```

```
  .apply(lambda x: x.sample(n = 5000, random_state = 2025))
```

[7]:
```
balancedData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   review_id    25000 non-null  object
 1   user_id      25000 non-null  object
 2   business_id  25000 non-null  object
 3   stars        25000 non-null  int64
 4   text         25000 non-null  object
 5   text_VADER   25000 non-null  object
 6   text_TFIDF   25000 non-null  object
 7   num_Tokens   25000 non-null  int64
dtypes: int64(2), object(6)
memory usage: 1.5+ MB
```

1.1.5 4. Feature Engineering VADER-Sentiment Scores

[8]:
```
# Generate VADER sentiment scores
sia = SentimentIntensityAnalyzer() # Initializing VADER Sentiment Intensity␣
 ↪Analyzer
balancedData['vader_scores'] = balancedData['text_VADER'].apply(sia.
 ↪polarity_scores) # Applying VADER to the 'text_VADER' column and storing␣
 ↪results in a new column 'vader_scores'
balancedData = pd.concat([balancedData.drop(['vader_scores'], axis=1),␣
 ↪balancedData['vader_scores'].apply(pd.Series)], axis=1) # Expanding the␣
 ↪'vader_scores' dictionary into separate columns

balancedData.head() # Previewing the final balanced dataset with VADER scores
```

[8]:
```
               review_id                 user_id             business_id  \
0 CoCim4CRm-WCoU-CFfWpLw  McdCFYocB1hFIiDQBRQ7YA  P_nqb7lULOtx3pAJbKfFXA
1 8s6Eejmy24XUhgNkR2uIUA  X67DbQdqHeZ-F2UVUOhn1g  WNjrsnJVPPnv_FtHHdjklA
2 2GdPCXF_5fR4_od5DJTD8Q  -VPeYf78MNJAB0iR7d9-zg  QboMIy08NLnBbLXEsmnDHg
3 vYSCzz-jM7ibdoIUCRLysw  IOVt1g8iKOD_cxXkJyXb0A  INz7vujcHs0AggsV__pXYQ
4 mLokfOcquwIP57pcOkHBZQ  TV3p-bv5yh8RgdJ3WxM7Ug  eh8WfQqPa2ZWtbXe9_wHgQ

   stars                                               text  \
0      1  Santa Fe used to be my favorite restaurant. I …
1      1  I have never experienced this level of incompe…
2      1  I've noticed a "Rising sun" flag displayed in …
3      1  Sold me a part that was wrong size and wouldn'…
4      1  I brought my car to Hyundai Service for a chec…
```

```
                                           text_VADER  \
0  Santa Fe used to be my favorite restaurant. I …
1  I have never experienced this level of incompe…
2  I've noticed a "Rising sun" flag displayed in …
3  Sold me a part that was wrong size and wouldn'…
4  I brought my car to Hyundai Service for a chec…

                                           text_TFIDF  num_Tokens    neg  \
0  santa fe favorite restaurant enjoy cancun taco…          48  0.119
1  experience level incompetence customer service…          84  0.110
2  notice rise sun flag display restaurant symbol…          27  0.276
3                  sell wrong size exchange get rob            6  0.222
4  bring car hyundai service check engine light c…          44  0.055

     neu    pos  compound
0  0.779  0.102   -0.0209
1  0.819  0.071   -0.6697
2  0.690  0.034   -0.9612
3  0.778  0.000   -0.6449
4  0.919  0.026   -0.3201
```

1.1.6 5. Creating Bag-of-Words Features

[ ]:
```
# Bag-of-Words Feature Engineering
import sklearn
from sklearn.feature_extraction.text import CountVectorizer

balancedDataBoW = balancedData.copy() # Creating a copy of balancedData for␣
 ↪Bag-of-Words processing

# Creating Bag-of-Words Matrix
bow_vectorizer = CountVectorizer(
    max_features = 5000, # Limiting to top 5000 features to manage␣
 ↪dimensionality
    ngram_range= (1, 1), # Using unigrams only,
    # min_df = 0.03, # Minimum document frequency of 5%
    # max_df = 0.75 # Maximum document frequency of 75%
    )

# Creating Bag-of-Words matrix
bow_matrix = bow_vectorizer.fit_transform(balancedDataBoW['text_TFIDF']) #␣
 ↪Fitting and transforming the Bag-of-Words matrix

# Adding Bag-of-Words features to balancedDataBoW DataFrame
bow_df = pd.DataFrame(
    bow_matrix.toarray(),
    index=balancedDataBoW.index,
    columns=bow_vectorizer.get_feature_names_out()
)
balancedDataBoW = pd.concat([balancedDataBoW.reset_index(drop=True), bow_df.
 ↪reset_index(drop=True)], axis=1) # Concatenating the Bag-of-Words features␣
 ↪to the balancedDataBoW DataFrame
balancedDataBoW.drop(columns=['text', 'text_VADER', 'text_BoW'], inplace=True)␣
 ↪# Dropping text columns to reduce file size
```

[15]:
```
# Checking the shape of the Bag-of-Words DataFrame
balancedDataBoW.shape
```

[15]:
```
(25000, 5008)
```

1.1.7 Saving Cleaned Dataset as separate .CSV file

[16]:
```
# Save the balancedDataBoW data to a new CSV file to prevent future re-runs of␣
 ↪the data cleaning step.
balancedDataBoW.to_csv('balancedDataBoW.csv', index=False)
```

1.2 End of Data Preparation Section

This marks the end of the *Data Preparation Section*. In summary, we've cleaned our original dataset, balanced our star-ratings through stratified sampling, feature engineered sentiment scores using VADER and using a Bag-of-Words matrix. We then dropped the 'text', 'text_VADER', and 'text_TFIDF' columns which are no longer required as we've already created vectorized features from the text. The final cleaned dataset is then saved as balancedDataBoW (for Bag-of-Words) / balancedDataTFIDF (for TF-IDF) csv files.

1.2.1 Next Steps

In the next phase, the *Data Analysis Section* we will further examine and explore this balancedData csv file.

CIND 820 - Data Analysis Section

November 10, 2025

1 CIND 820 - Deliverable 3 - Data Analysis

1.1 Data Analysis Section

In this section of the notebook, we will examine the finalized dataset we created from the previous *Data Preparation Section*. We will start this *Data Analysis Section* by providing a summary of our key dataset features (descriptive statistics portion) and then we will visualize any potential patterns or trends (exploratory data analysis portion). Throughout each chunk we will leave notes to briefly discuss our intial observations.

1.2 Retrieving our balancedDataBoW

Note: We've decided to utilize the Bag-of-Words vectorization method given its ability to capture more domain-agnostic words.

[1]:
```
import pandas as pd

# Windows Version of Filepath
filepath = r'C:\Users\Sunora\iCloudDrive\Documents\Learning Data Analytics\TMU␣
 ↪Certificate copy\CIND 820\Yelp Dataset\balancedDataBoW.csv'

## MacOS Version of Filepath
#filepathMac = r"/Users/sszhang/Documents/Learning Data Analytics/TMU␣
 ↪Certificate copy/CIND 820/Yelp Dataset/balancedDataBoW.csv"

data = pd.read_csv(filepath)
data.head()
```

[1]:
```
                review_id                 user_id             business_id  \
0  CoCim4CRm-WCoU-CFfWpLw  McdCFYocB1hFIiDQBRQ7YA  P_nqb7lULOtx3pAJbKfFXA
1  8s6Eejmy24XUhgNkR2uIUA  X67DbQdqHeZ-F2UVUOhn1g  WNjrsnJVPPnv_FtHHdjklA
2  2GdPCXF_5fR4_od5DJTD8Q  -VPeYf78MNJAB0iR7d9-zg  QboMIy08NLnBbLXEsmnDHg
3  vYSCzz-jM7ibdoIUCRLysw  I0Vt1g8iKOD_cxXkJyXb0A  INz7vujcHs0AggsV__pXYQ
4  mLokfOcquwIP57pcOkHBZQ  TV3p-bv5yh8RgdJ3WxM7Ug  eh8WfQqPa2ZWtbXe9_wHgQ

   stars  num_Tokens    neg    neu    pos  compound  aaron  …  yr  yuck  \
0      1          48  0.119  0.779  0.102   -0.0209      0  …   0     0
1      1          84  0.110  0.819  0.071   -0.6697      0  …   0     0
2      1          27  0.276  0.690  0.034   -0.9612      0  …   0     0
```

```
3      1           6  0.222  0.778  0.000   -0.6449      0  …   0     0
4      1          44  0.055  0.919  0.026   -0.3201      0  …   0     0

   yum  yummy  yup  zero  zone  zoo  zucchini  étouffée
0    0      0    0     0     0    0         0         0
1    0      0    0     0     0    0         0         0
2    0      0    0     0     0    0         0         0
3    0      0    0     0     0    0         0         0
4    0      0    0     0     0    0         0         0

[5 rows x 5008 columns]
```

1.3 Descriptive Statistics

Assessing the general structure and distribution of the overall dataset.

[3]:
```
data.describe() # Get summary statistics of the dataset
```

[3]:
```
              stars    num_Tokens           neg           neu           pos  \
count  25000.000000  25000.000000  25000.000000  25000.000000  25000.000000
mean       3.000000     49.302600      0.060373      0.772924      0.166702
std        1.414242     43.485489      0.063002      0.099700      0.111440
min        1.000000      1.000000      0.000000      0.250000      0.000000
25%        2.000000     21.000000      0.008000      0.719000      0.086000
50%        3.000000     36.000000      0.047000      0.786000      0.148000
75%        4.000000     63.000000      0.089000      0.841000      0.228000
max        5.000000    442.000000      0.714000      1.000000      0.750000

           compound         aaron         aback       abandon       ability  \
count  25000.000000  25000.000000  25000.000000  25000.000000  25000.000000
mean       0.478585      0.000840      0.000800      0.001040      0.002280
std        0.653563      0.041465      0.028274      0.039987      0.049345
min       -0.998300      0.000000      0.000000      0.000000      0.000000
25%        0.098450      0.000000      0.000000      0.000000      0.000000
50%        0.843900      0.000000      0.000000      0.000000      0.000000
75%        0.959200      0.000000      0.000000      0.000000      0.000000
max        0.999700      4.000000      1.000000      4.000000      2.000000

       …            yr          yuck           yum         yummy  \
count  …  25000.000000  25000.000000  25000.000000  25000.000000
mean   …      0.001800      0.003000      0.008560      0.015320
std    …      0.046871      0.058233      0.110305      0.131932
min    …      0.000000      0.000000      0.000000      0.000000
25%    …      0.000000      0.000000      0.000000      0.000000
50%    …      0.000000      0.000000      0.000000      0.000000
75%    …      0.000000      0.000000      0.000000      0.000000
max    …      2.000000      3.000000      3.000000      4.000000
```

```
               yup         zero         zone          zoo     zucchini  \
count  25000.000000  25000.000000  25000.000000  25000.000000  25000.000000
mean       0.000880      0.011560      0.001880      0.002360      0.002200
std        0.032238      0.116563      0.047713      0.085057      0.055455
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000      0.000000
50%        0.000000      0.000000      0.000000      0.000000      0.000000
75%        0.000000      0.000000      0.000000      0.000000      0.000000
max        2.000000      4.000000      3.000000      5.000000      3.000000

           étouffée
count  25000.000000
mean       0.000920
std        0.040487
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        4.000000

[8 rows x 5005 columns]
```

Notes Based on the high-level descriptive statistics we were able to generate using *.describe()* we quickly observe the following:

1. That the Bag-of-Words (BoW) vectors is extremely information sparse. This is primarily due to the nature of the BoW matrix, which splits words in a corpus into separate columns before counting the frequency of how often those words appear in document. An understanding of the occurrence distribution for each individual tokenized words provide little value on its own. The value of this BoW matrix arises when combining it with other features. The matrix will be valuable during our model training and development section as it helps to add more dimensionality to text-data during model training. As such, we will disregard the BoW matrix information for the rest of the *Data Analysis Section* as the sparse-information is going to be more pivotal in our *Model Evaluation Section.*

2. We also notice that the *stars* column is equally distributed based on the quartile distribution metrics. This is entirely sensible as we balanced our star-rating through stratified sampling. This simply confirms we sampled and balanced correctly.

3. The *num_Tokens* column shows that on average there were 49 tokens per review while the median was 36. At a glance, it suggests that our num_Tokens is right-skewed where most of our data is concentrated on the left. In other words, long bodies of text in reviews are less common than shorter reviews. Additionally, a standard deviation of 43 tokens would suggest that the majority of reviews were between 6 and 92 tokens. The least number of tokens in a review was 1 word and the most was 442 words. Recall, this column only counts lemmatized words and disregards numbers or other non-alphabet characters.

4. Based on the VADER-scores generated, we observe 4 columns: negative, neutral, positive,

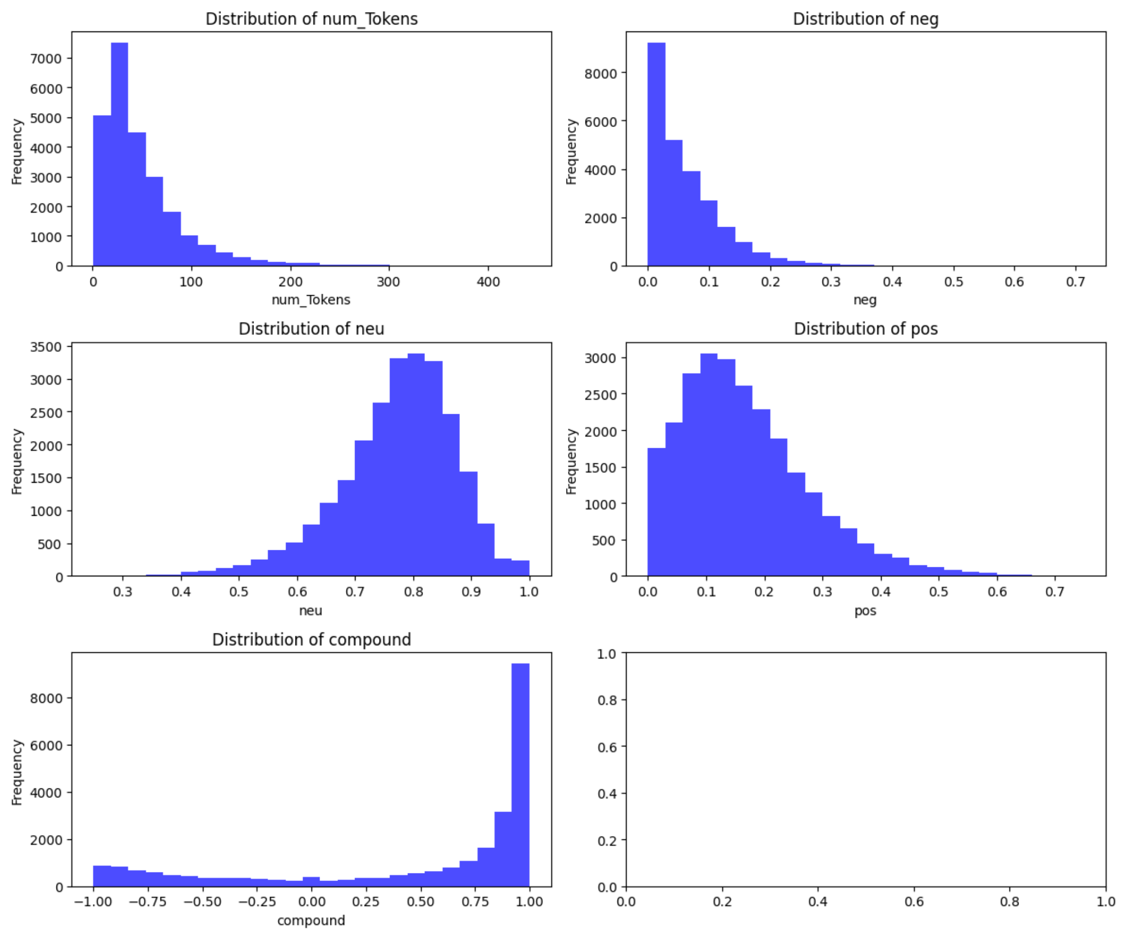
and compound. The compound score is an aggregated view of the previous 3. The compound-score is calculated by using a normalized sum of all the 3 previous sentiment scores. It ranges between -1 and +1.

1. The negative-score measures how much of the document (per review level), on a percentage basis, contained negative text. It therefore ranges from 0 to 1. Based on our output, it seems on average our reviews contained about 6.03% negative sentiment while the medium was 4.7%. This would again suggest a right-skew for the negative-sentiment score suggesting that there was likely a higher concentration of reviews below the 6.03% score. However, the standard devation was 6.3% suggesting a high-degree of variation on how the negative-sentiment score is distributed. A histogram would be valuable here.
2. In a similar fashion, the neutral- and positive-score measures show, on a percentage basis, how much of the document was neutral and how much was positive. They also range from 0 - 1. We observe an average of 77% of documents were neutral but medium as 79% suggesting a left-skew; indicating that more documents had higher neutral proportionalities than the average 77% would have you believe. A standard deviation of ~10% would suggest an ample amount of variation and that most of our neutral-sentiment scores sat between 67% - 87%. Another histogram would be valuable here.
3. The positive-score showed an average of 17% across all documents with a medium of 15%, again indicating a right-skew. It's standard deviation is around 11% suggesting that the majority of the positive-sentiment score resides between 6% - 28%; a rather high amount of variation.
4. Lastly, our compound-score averages at ~0.48 and has a median of 0.84 suggesting a significant left-skew. This may suggest outliers but certainly suggests that we have a small amount of extremely negative sentiment reviews that is weighing down the average. Furthermore, a standard deviation of 0.65 tells us that this metric also contains a high-degree of variation. Another histogram would be useful here.

1.3.1 Creating a Histogram to Visualize the Distributions

Creating Histogram for num_Tokens, negative-, neutral-, positive-sentiment scores, and the compound sentiment score.

[11]:
```python
import matplotlib.pyplot as plt

# Creating a Histogram to Visualize the Distributions
# Creating Histogram for num_Tokens, negative-, neutral-, positive-sentiment␣
 ↪scores, and the compound sentiment score.
fig, axs = plt.subplots(3, 2, figsize=(12, 10)) # 3 rows, 2 columns
axs = axs.ravel() # Flatten the 2D array of axes to 1D for easy iteration
columns = ['num_Tokens', 'neg', 'neu', 'pos', 'compound'] # List of columns to␣
 ↪plot
for i, col in enumerate(columns): # Iterate over the columns
    axs[i].hist(data[col], bins=25, color='blue', alpha=0.7) # Create histogram
    axs[i].set_title(f'Distribution of {col}') # Set title
    axs[i].set_xlabel(col) # Set x-axis label
    axs[i].set_ylabel('Frequency') # Set y-axis label
plt.tight_layout() # Adjust layout
plt.show() # Show the plots
```



1.3.2 Assessing the Correlation between Star Rating, Number of Tokens, and Sentiment Scores

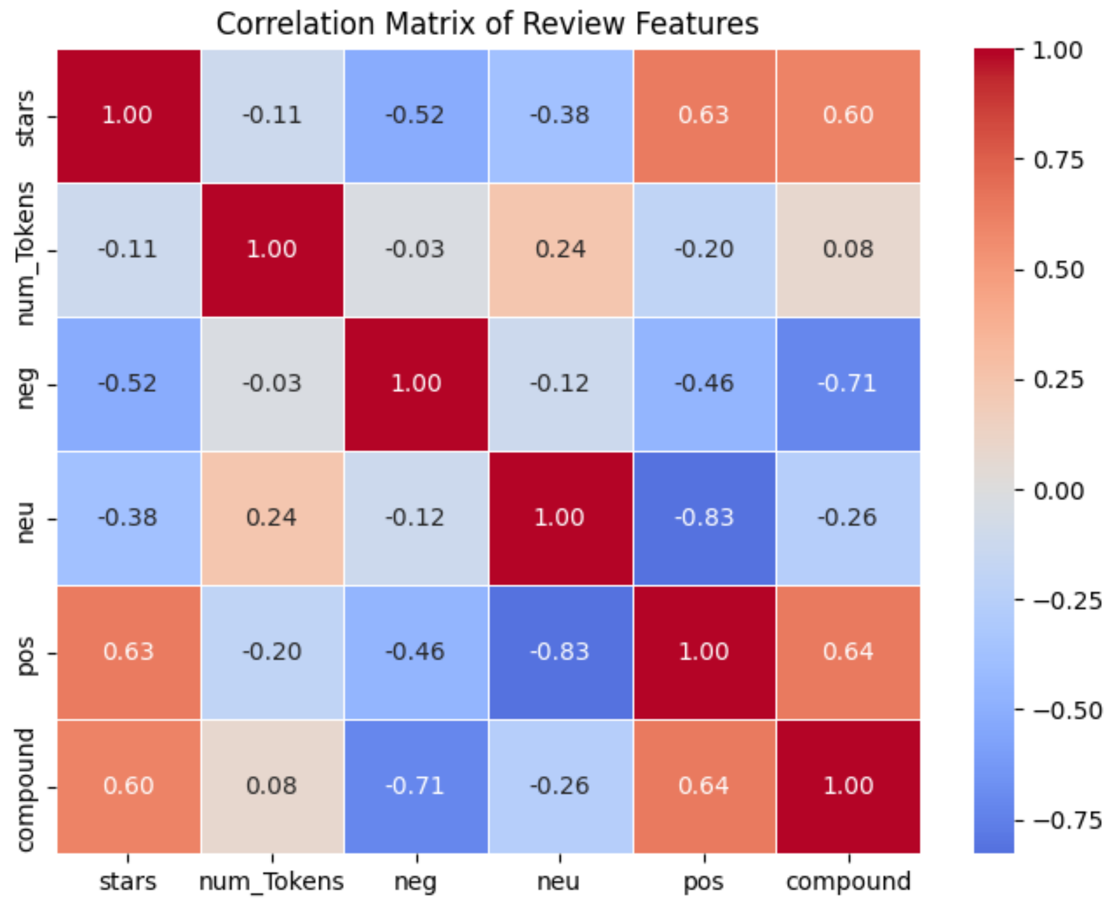
Creating a heat-map matrix and visualization of the matrix.

[12]:
```
data[['stars', 'num_Tokens', 'neg', 'neu', 'pos', 'compound']].corr() #␣
↪Creating a correlation matrix
```

[12]:
```
               stars  num_Tokens       neg       neu       pos  compound
stars       1.000000   -0.114251 -0.516471 -0.377190  0.629407  0.601029
num_Tokens -0.114251    1.000000 -0.026312  0.238532 -0.198498  0.080581
neg        -0.516471   -0.026312  1.000000 -0.118613 -0.459236 -0.714234
neu        -0.377190    0.238532 -0.118613  1.000000 -0.827564 -0.261220
pos         0.629407   -0.198498 -0.459236 -0.827564  1.000000  0.637490
compound    0.601029    0.080581 -0.714234 -0.261220  0.637490  1.000000
```

[13]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Select relevant columns and compute correlation matrix
corr_matrix = data[['stars', 'num_Tokens', 'neg', 'neu', 'pos', 'compound']].
 ↪corr()

# Create the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0, fmt=".2f",␣
 ↪linewidths=0.5)
plt.title("Correlation Matrix of Review Features")
plt.show()
```



	stars	num_Tokens	neg	neu	pos	compound
stars	1.00	-0.11	-0.52	-0.38	0.63	0.60
num_Tokens	-0.11	1.00	-0.03	0.24	-0.20	0.08
neg	-0.52	-0.03	1.00	-0.12	-0.46	-0.71
neu	-0.38	0.24	-0.12	1.00	-0.83	-0.26
pos	0.63	-0.20	-0.46	-0.83	1.00	0.64
compound	0.60	0.08	-0.71	-0.26	0.64	1.00

Notes:

- The most positively correlated features appear to be:
  1. **Positive sentiment scores** and **compound sentiment scores** at **0.64**
  2. **Positive sentiment scores** and **star ratings** at **0.63**
  3. **Compound sentiment scores** and **star ratings** at **0.60**
- The most negatively correlated featurs appear to be:
  1. **Neutral sentiment scores** and **positive sentiment scores** at **-0.83**

2. **Negative sentiment scores** and **compound sentiment scores** at **-0.71**
3. **Negative sentiment scores** and **star ratings** at **-0.52**

1.4 Exploratory Data Analysis

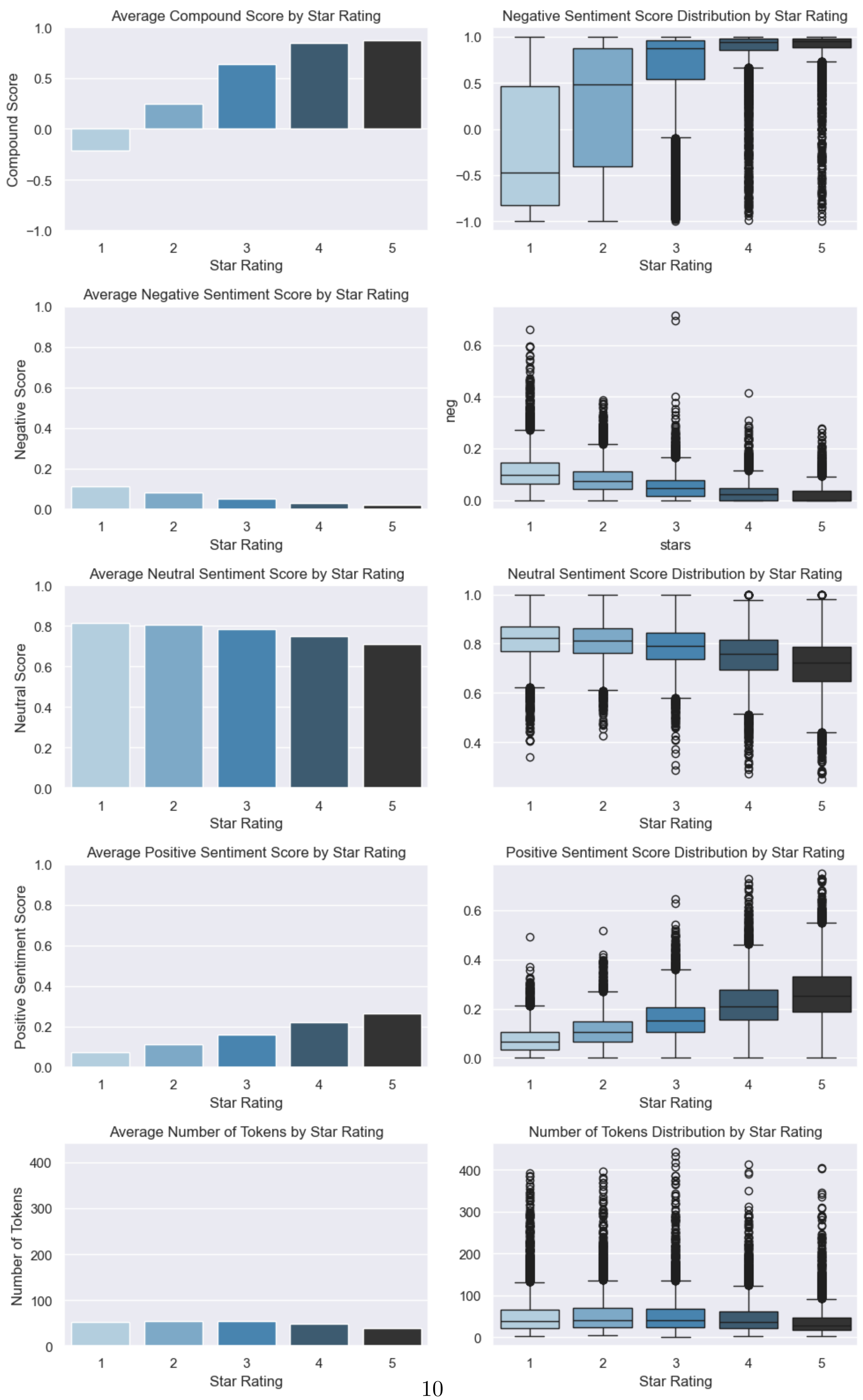
In this section, we will further explore how the data is distributed at star-rating level.

Bar-plot and Boxplot of Average Compound-, Negative-, Neutral-, Positive-Sentiment Scores and Number of Tokens by Star-Rating Group

```
[60]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Set plot style
sns.set(style="darkgrid")

# Create figure and subplots
fig, axes = plt.subplots(5, 2, figsize=(10, 16), sharey=False)

# Bar plot: Average compound score by star rating
avg_compound = data.groupby('stars')['compound'].mean().reset_index()
sns.barplot(x='stars', y='compound', data=avg_compound, ax=axes[0, 0],␣
 ↪palette='Blues_d', hue='stars', legend=False)
axes[0, 0].set_title("Average Compound Score by Star Rating")
axes[0, 0].set_xlabel("Star Rating")
axes[0, 0].set_ylabel("Compound Score")
axes[0, 0].set_ylim(-1, 1)

# Boxplot: Distribution of compound scores by star rating
sns.boxplot(x='stars', y='compound', data=data, ax=axes[0,1],␣
 ↪palette='Blues_d', hue='stars', legend=False)
axes[0, 1].set_title("Compound Score Distribution by Star Rating")
axes[0, 1].set_xlabel("Star Rating")
axes[0, 1].set_ylabel("")  # Avoid repeating y-axis label

# Bar plot: Average negative score by star rating
avg_neg = data.groupby('stars')['neg'].mean().reset_index()
sns.barplot(x='stars', y='neg', data=avg_neg, ax=axes[1, 0], palette='Blues_d',␣
 ↪hue='stars', legend=False)
axes[1, 0].set_title("Average Negative Sentiment Score by Star Rating")
axes[1, 0].set_xlabel("Star Rating")
axes[1, 0].set_ylabel("Negative Score")
axes[1, 0].set_ylim(0, 1)

# Boxplot: Distribution of negative scores by star rating
sns.boxplot(x='stars', y='neg', data=data, ax=axes[1,1], palette='Blues_d',␣
 ↪hue='stars', legend=False)
```

```python
axes[0, 1].set_title("Negative Sentiment Score Distribution by Star Rating")
axes[0, 1].set_xlabel("Star Rating")
axes[0, 1].set_ylabel("")  # Avoid repeating y-axis label

# Bar plot: Average neutral score by star rating
avg_neu = data.groupby('stars')['neu'].mean().reset_index()
sns.barplot(x='stars', y='neu', data=avg_neu, ax=axes[2, 0], palette='Blues_d',␣
 ↪hue='stars', legend=False)
axes[2, 0].set_title("Average Neutral Sentiment Score by Star Rating")
axes[2, 0].set_xlabel("Star Rating")
axes[2, 0].set_ylabel("Neutral Score")
axes[2, 0].set_ylim(0, 1)

# Boxplot: Distribution of neutral scores by star rating
sns.boxplot(x='stars', y='neu', data=data, ax=axes[2,1], palette='Blues_d',␣
 ↪hue='stars', legend=False)
axes[2, 1].set_title("Neutral Sentiment Score Distribution by Star Rating")
axes[2, 1].set_xlabel("Star Rating")
axes[2, 1].set_ylabel("")  # Avoid repeating y-axis label

# Bar plot: Average positive score by star rating
avg_pos = data.groupby('stars')['pos'].mean().reset_index()
sns.barplot(x='stars', y='pos', data=avg_pos, ax=axes[3, 0], palette='Blues_d',␣
 ↪hue='stars', legend=False)
axes[3, 0].set_title("Average Positive Sentiment Score by Star Rating")
axes[3, 0].set_xlabel("Star Rating")
axes[3, 0].set_ylabel("Positive Sentiment Score")
axes[3, 0].set_ylim(0, 1)

# Boxplot: Distribution of positive scores by star rating
sns.boxplot(x='stars', y='pos', data=data, ax=axes[3,1], palette='Blues_d',␣
 ↪hue='stars', legend=False)
axes[3, 1].set_title("Positive Sentiment Score Distribution by Star Rating")
axes[3, 1].set_xlabel("Star Rating")
axes[3, 1].set_ylabel("")  # Avoid repeating y-axis label

# Bar plot: Average num_Tokens by star rating
avg_tokens = data.groupby('stars')['num_Tokens'].mean().reset_index()
sns.barplot(x='stars', y='num_Tokens', data=avg_tokens, ax=axes[4, 0],␣
 ↪palette='Blues_d', hue='stars', legend=False)
axes[4, 0].set_title("Average Number of Tokens by Star Rating")
axes[4, 0].set_xlabel("Star Rating")
axes[4, 0].set_ylabel("Number of Tokens")
axes[4, 0].set_ylim(0, data['num_Tokens'].max())

# Boxplot: Distribution of token counts by star rating
```

```
sns.boxplot(x='stars', y='num_Tokens', data=data, ax=axes[4,1],␣
↪palette='Blues_d', hue='stars', legend=False)
axes[4, 1].set_title("Number of Tokens Distribution by Star Rating")
axes[4, 1].set_xlabel("Star Rating")
axes[4, 1].set_ylabel("")  # Avoid repeating y-axis label

# Final layout
plt.tight_layout()
plt.show()
```

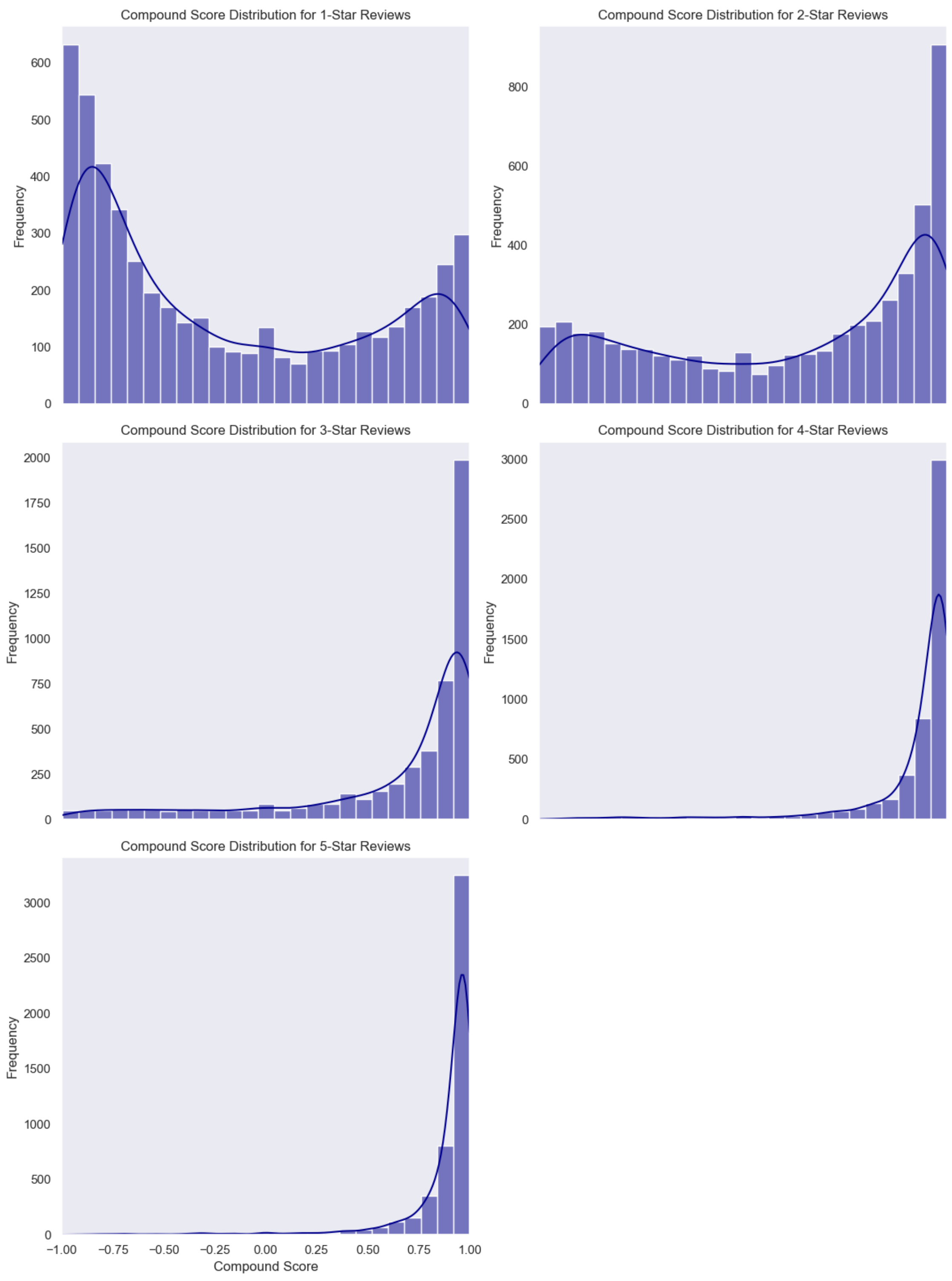

Average Compound Score by Star Rating
Compound Score
Star Rating
Negative Sentiment Score Distribution by Star Rating
Star Rating
Average Negative Sentiment Score by Star Rating
Negative Score
Star Rating
neg
stars
Average Neutral Sentiment Score by Star Rating
Neutral Score
Star Rating
Neutral Sentiment Score Distribution by Star Rating
Star Rating
Average Positive Sentiment Score by Star Rating
Positive Sentiment Score
Star Rating
Positive Sentiment Score Distribution by Star Rating
Star Rating
Average Number of Tokens by Star Rating
Number of Tokens
Star Rating
Number of Tokens Distribution by Star Rating
Star Rating

Notes

- The **compound sentiment-score barplot** above visualizes what the average compound score would be at a star-rating level.
  - It shows notable visible differences in the average compound scores between 1-3 star-ratings but 4-5 star-ratings become much less distinguishable when assessing solely by the average compound score
- The **compound sentiment-score boxplot** visualizes the distribution and variation of the compound score per star-rating
  - It shows that there is a high degree of variation in the compound sentiment scores amonst 1-3 star ratings but much less variation in 4-5 star-ratings. This suggests that in fact, based solely on the compound sentiment scores alone, 4 or 5 star ratings are much more discernable from 1-3 star ratings but less discernable amongst each other.
- The **negative sentiment-score barplot** visualizes the average negative score at a star-rating level.
  - It shows a incremental differences in average negative-scores per star-rating where the proportion of negative ratings decrease as star-ratings increase
  - This measure is congruent that higher star ratings should typically carry stronger positive sentiments
- The **negative sentiment-score boxplot** visualizes the distribution of the negative score at a star-rating level.
  - It shows that the majority of the negative-scores are tightly distributed for each various star-rating category and become incrementally tighter as star-ratings increase
- The **neutral sentiment-score barplot** visualizes the average neutral score at a star-rating level.
  - It too shows incremental differences neutral-scores in average per star-rating.
  - This is likely explained by the way negative, positive, and neutral scores are calculated - the all sum together to make up 100% - thus suggesting that as star-ratings increase the sentiment buckets of positive and neutral becomg more pronounced.
- The **neutral sentiment-score boxplot** visualizes the distribution of the negative score at a star-rating level.
  - It shows that the majority of the neutral-scores are tightly distributed for each various star-rating category but becomes incrementally sparser as star-ratings increase
- The **positive sentiment-score barplot** visualizes the average neutral score at a star-rating level.
  - It too shows incremental differences in the averages of positive-scores per star-rating.
  - This measure is also congruent with the theory that higher star ratings should typically carry stronger positive sentiments
- The **positive sentiment-score boxplot** visualizes the distribution of the negative score at a star-rating level.
  - It shows that the majority of the neutral-scores are tightly distributed for each various star-rating category but becomes incrementally sparser as star-ratings increase

1.4.1 Histrogram of num_Tokens, negative-, neutral-, positive-sentiment scores, and the compound sentiment score per star-rating group.

[71]:
```python
# Histograms of Compound Scores by Star Rating

import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="dark")

# Get unique star ratings
unique_stars = sorted(data['stars'].unique())

# Create subplots: 3 rows, 2 columns
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(12, 16), sharex=True)

# Flatten the 2D axes array to 1D
axes = axes.flatten()

# Loop through each star rating and plot its histogram
for i, star in enumerate(unique_stars):
    subset = data[data['stars'] == star]
    sns.histplot(subset['compound'], bins=25, kde=True, ax=axes[i],␣
 ↪color='darkblue')
    axes[i].set_title(f"Compound Score Distribution for {star}-Star Reviews")
    axes[i].set_xlim(-1, 1)
    axes[i].set_ylabel("Frequency")
    if i == len(unique_stars) - 1:
        axes[i].set_xlabel("Compound Score")
    else:
        axes[i].set_xlabel("")

# Hide any unused subplot (if fewer than 6)
for j in range(len(unique_stars), len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```

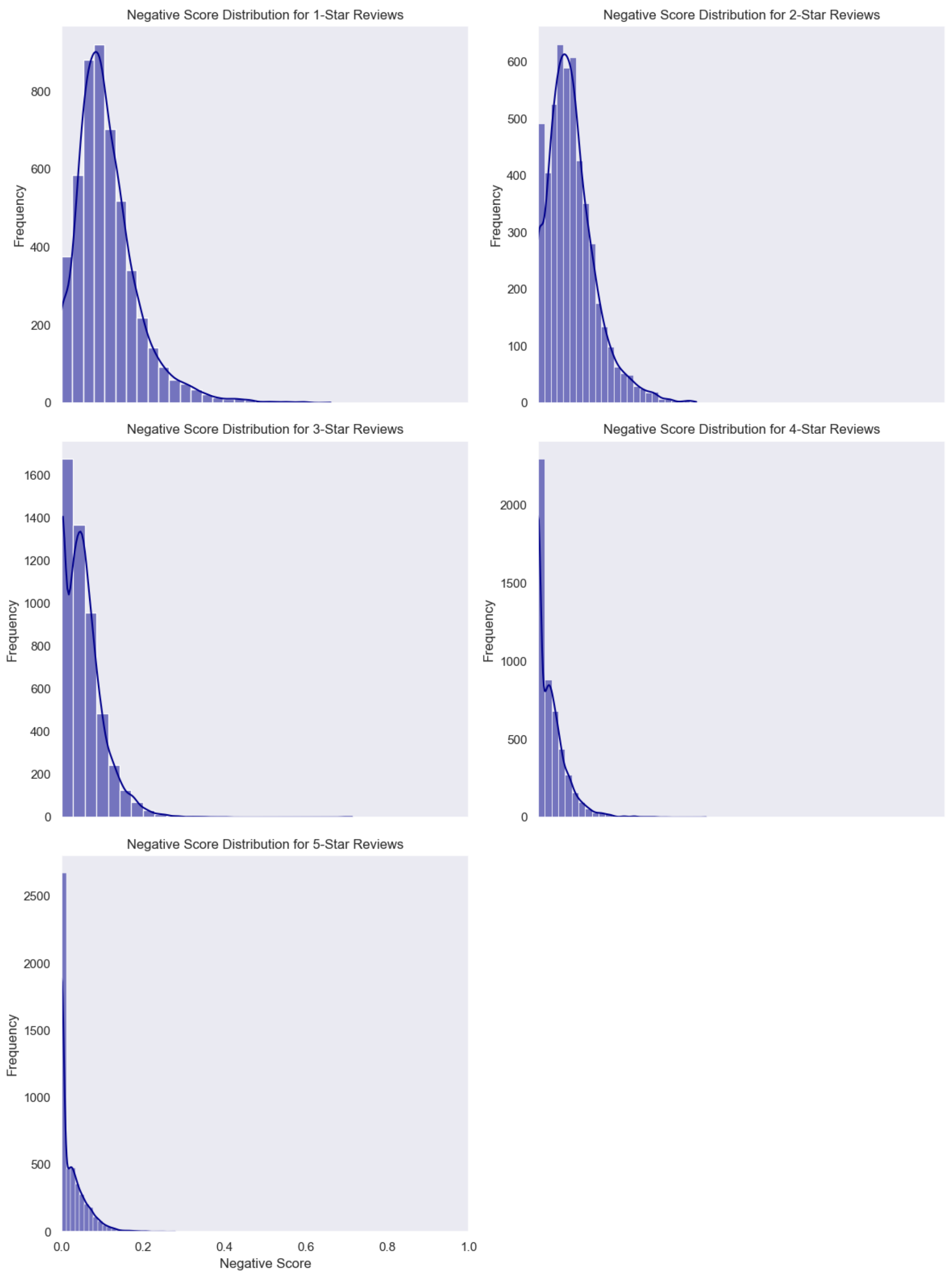


Notes:

- We quickly observe a bimodal distribution of compound scores for 1 and 2 star rating reviews

- We also note that as the star-ratings increase, we begin to see a left-skew form. This left-skew becomes increasingly more pronounced between 3-5 star ratings.
- This distribution suggests that as star-ratings increase, the number of occurence for higher compound scores also increases.

[70]:
```python
# Histograms of Negative Sentiment Scores by Star Rating

import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="dark")

# Get unique star ratings
unique_stars = sorted(data['stars'].unique())

# Create subplots: 3 rows, 2 columns
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(12, 16), sharex=True)

# Flatten the 2D axes array to 1D
axes = axes.flatten()

# Loop through each star rating and plot its histogram
for i, star in enumerate(unique_stars):
    subset = data[data['stars'] == star]
    sns.histplot(subset['neg'], bins=25, kde=True, ax=axes[i], color='darkblue')
    axes[i].set_title(f"Negative Score Distribution for {star}-Star Reviews")
    axes[i].set_xlim(0, 1)
    axes[i].set_ylabel("Frequency")
    if i == len(unique_stars) - 1:
        axes[i].set_xlabel("Negative Score")
    else:
        axes[i].set_xlabel("")

# Hide any unused subplot (if fewer than 6)
for j in range(len(unique_stars), len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```

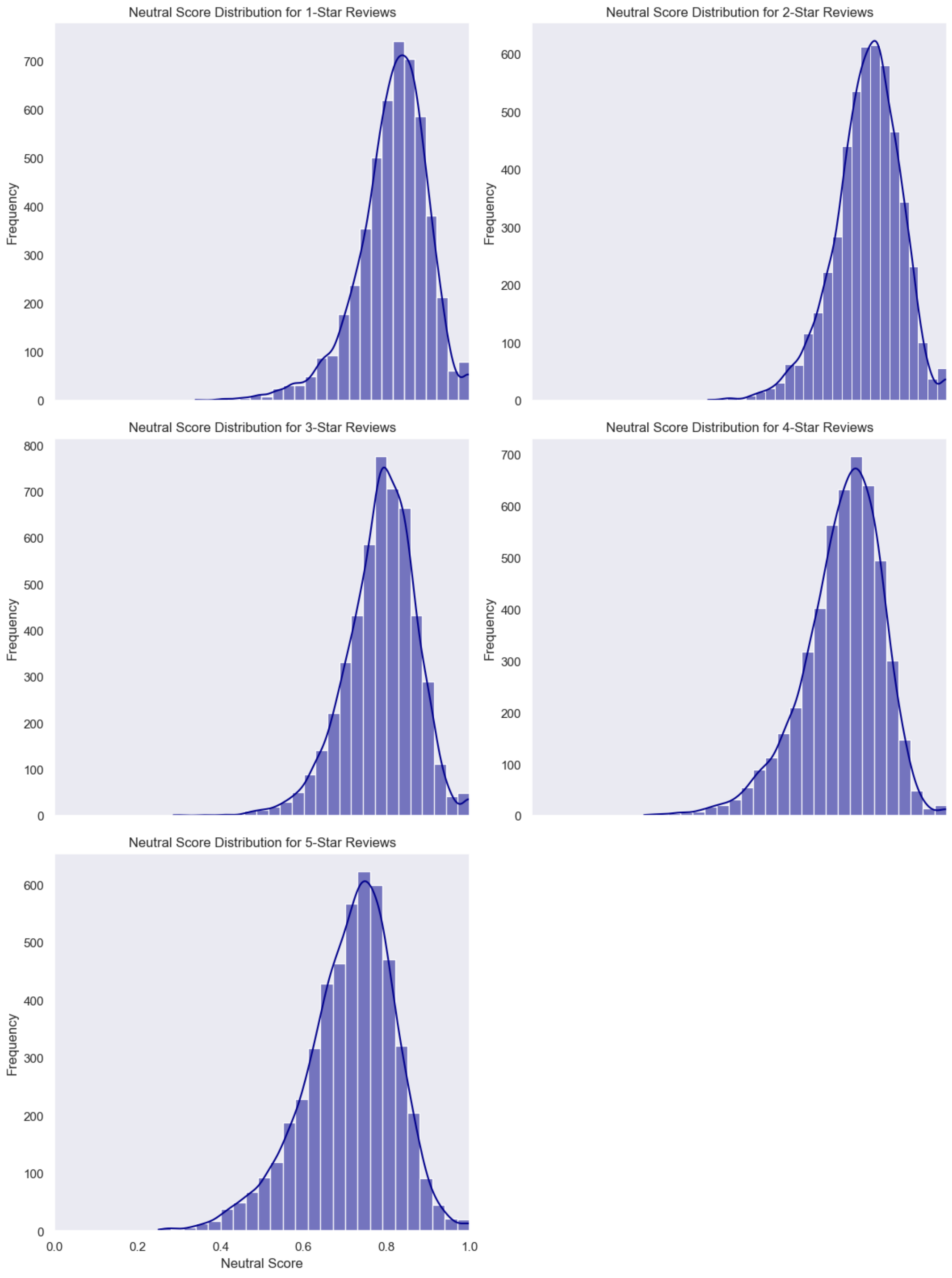


Notes:

- We quickly observe a right-skew shared across all 5-plots suggesting that consistently across

all star-ratings the majority of negative-scores are concentrated on the lower-end; meaning the proportion of negative sentiment within documents are low across star-ratings
- We also observe that as star ratings increase, the pattern remains but on a more minimized scale. This aligns with behavior observed in the compound score histogram.

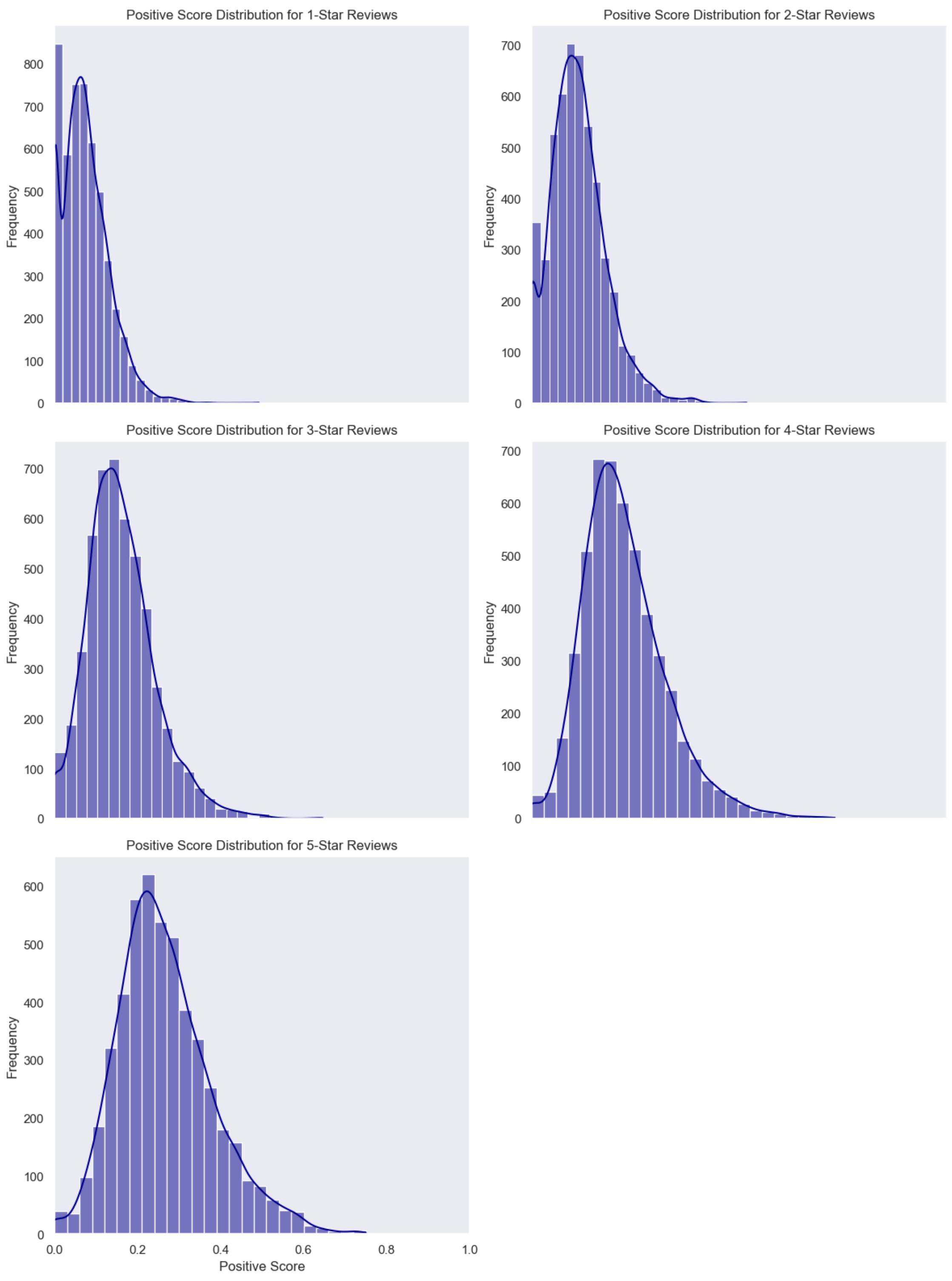
[65]:
```python
# Histograms of Neutral Sentiment Scores by Star Rating

import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="dark")

# Get unique star ratings
unique_stars = sorted(data['stars'].unique())

# Create subplots: 3 rows, 2 columns
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(12, 16), sharex=True)

# Flatten the 2D axes array to 1D
axes = axes.flatten()

# Loop through each star rating and plot its histogram
for i, star in enumerate(unique_stars):
    subset = data[data['stars'] == star]
    sns.histplot(subset['neu'], bins=25, kde=True, ax=axes[i], color='darkblue')
    axes[i].set_title(f"Neutral Score Distribution for {star}-Star Reviews")
    axes[i].set_xlim(0, 1)
    axes[i].set_ylabel("Frequency")
    if i == len(unique_stars) - 1:
        axes[i].set_xlabel("Neutral Score")
    else:
        axes[i].set_xlabel("")

# Hide any unused subplot (if fewer than 6)
for j in range(len(unique_stars), len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```



Notes:

- We observe that the Neutral compound score is more-or-less a normal curve across all 5-star rating plots. Based on this, the information carried by the *neutral score* it doesn't seem like it would be a reliable indicator for identifying star-ratings.

[2]:
```python
# Histograms of Positive Sentiment Scores by Star Rating

import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="dark")

# Get unique star ratings
unique_stars = sorted(data['stars'].unique())

# Create subplots: 3 rows, 2 columns
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(12, 16), sharex=True)

# Flatten the 2D axes array to 1D
axes = axes.flatten()

# Loop through each star rating and plot its histogram
for i, star in enumerate(unique_stars):
    subset = data[data['stars'] == star]
    sns.histplot(subset['pos'], bins=25, kde=True, ax=axes[i], color='darkblue')
    axes[i].set_title(f"Positive Score Distribution for {star}-Star Reviews")
    axes[i].set_xlim(0, 1)
    axes[i].set_ylabel("Frequency")
    if i == len(unique_stars) - 1:
        axes[i].set_xlabel("Positive Score")
    else:
        axes[i].set_xlabel("")

# Hide any unused subplot (if fewer than 6)
for j in range(len(unique_stars), len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```

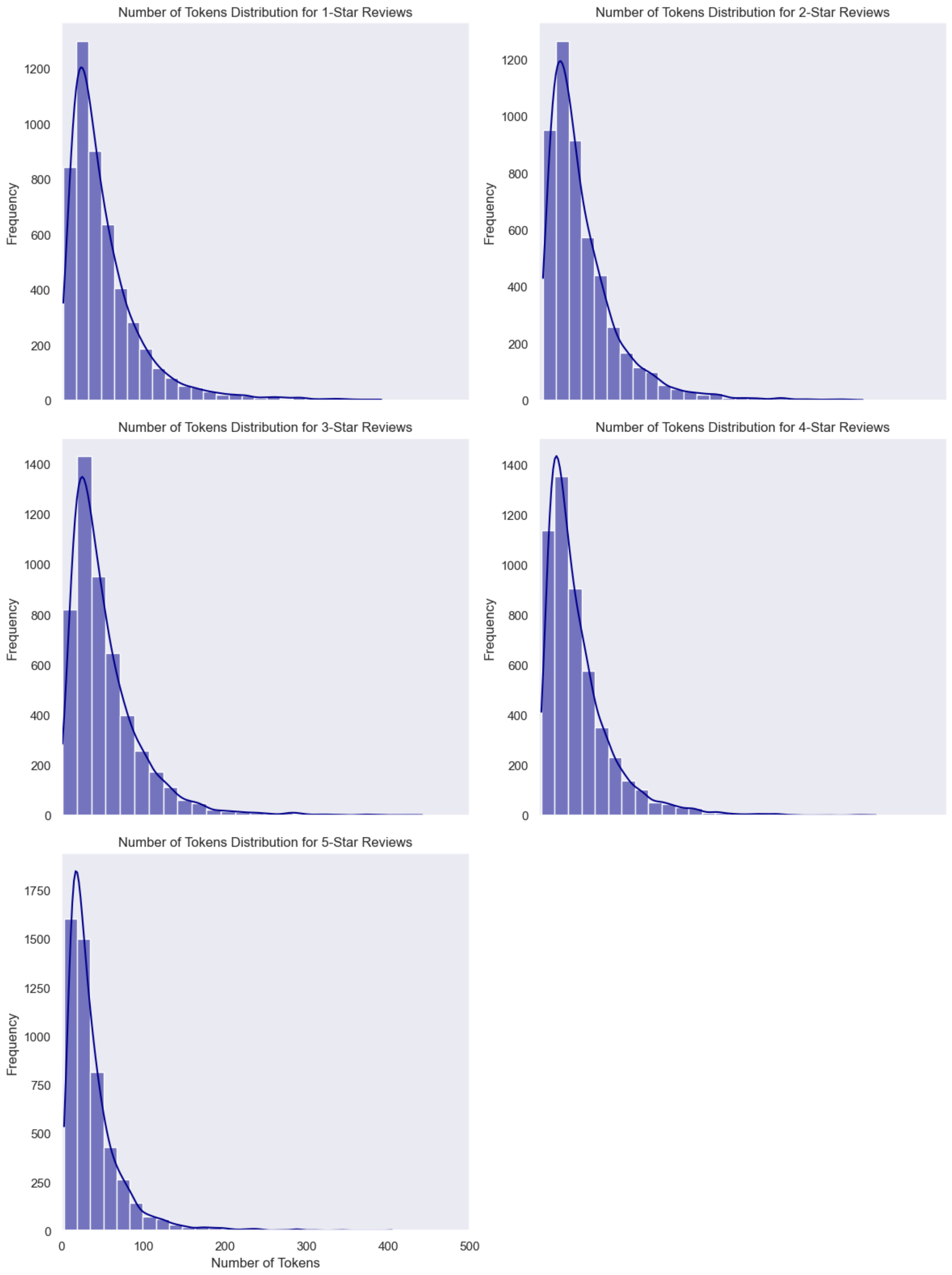


Notes:

- Here we quickly observe a shift from a right-skew distribution of positive sentiment score for lower star-ratings to a more normalized (minimal right-skew curve) as star-ratings increase

- This suggests that when star-ratings are low, the proportion of positive sentiment scores are frequently low as well. However, as star-ratings increase, the distribution of positive-sentiment scores increases where the skew becomes less pronounced suggesting an increase in the proportion of higher positive-sentiment scores.

[ ]:
```
# Histograms of Number of Tokens by Star Rating

import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="dark")

# Get unique star ratings
unique_stars = sorted(data['stars'].unique())

# Create subplots: 3 rows, 2 columns
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(12, 16), sharex=True)

# Flatten the 2D axes array to 1D
axes = axes.flatten()

# Loop through each star rating and plot its histogram
for i, star in enumerate(unique_stars):
    subset = data[data['stars'] == star]
    sns.histplot(subset['num_Tokens'], bins=25, kde=True, ax=axes[i],␣
 ↪color='darkblue')
    axes[i].set_title(f"Number of Tokens Distribution for {star}-Star Reviews")
    axes[i].set_xlim(0, 500)
    axes[i].set_ylabel("Frequency")
    if i == len(unique_stars) - 1:
        axes[i].set_xlabel("Number of Tokens")
    else:
        axes[i].set_xlabel("")

# Hide any unused subplot (if fewer than 6)
for j in range(len(unique_stars), len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```



Notes:

- Here we observe a shared right-skew across all 5-plots suggesting that the number of tokens is consistently concentrated around the lower token counts for all 5-star ratings.

- We also observe that the 5-star plot has the sharpest skew suggesting that 5-star reviews more frequently had the least number of tokens within the review.

CIND 820 - Project Evaluation Section

November 10, 2025

1 CIND 820 - Deliverable 3 - Model Evaluation

1.1 Data Analysis Section

This section of the notebook aims to train and test 3 different supervised machine learning models on its performance with regard to predicting star-ratings based on user feedback. The notebook will be broken down into 3-main sections:

1. Data preparation
2. Model Training and Implementation
    1. Model Training for Logistic Regression Algorithm
    2. Model Training for a Naive Bayes Algorithm
    3. Model Training for a Support Vector Regression Algorithm
3. Model Performance Evaluation

1.2 Data Preparation

[1]:
```
# Importing pandas library
import pandas as pd
import numpy as np

#MacOS Version of Filepath
filepathMac = r"/Users/sszhang/Documents/Learning Data Analytics/TMU␣
 ↪Certificate copy/CIND 820/Yelp Dataset/balancedDataBoW.csv"
# filepathWindows = r"C:\Users\Sunora\iCloudDrive\Documents\Learning Data␣
 ↪Analytics\TMU Certificate copy\CIND 820\Yelp Dataset\balancedDataBoW.csv"
data = pd.read_csv(filepathMac)
data.head()
```

[1]:
```
               review_id                 user_id             business_id  \
0  CoCim4CRm-WCoU-CFfWpLw  McdCFYocB1hFIiDQBRQ7YA  P_nqb7lULOtx3pAJbKfFXA
1  8s6Eejmy24XUhgNkR2uIUA  X67DbQdqHeZ-F2UVUOhn1g  WNjrsnJVPPnv_FtHHdjklA
2  2GdPCXF_5fR4_od5DJTD8Q  -VPeYf78MNJAB0iR7d9-zg  QboMIy08NLnBbLXEsmnDHg
3  vYSCzz-jM7ibdoIUCRLysw  I0Vt1g8iKOD_cxXkJyXb0A  INz7vujcHs0AggsV__pXYQ
4  mLokfOcquwIP57pcOkHBZQ  TV3p-bv5yh8RgdJ3WxM7Ug  eh8WfQqPa2ZWtbXe9_wHgQ
```

```
   stars  num_Tokens    neg    neu    pos  compound  aaron  …  yr  yuck  \
0      1          48  0.119  0.779  0.102   -0.0209      0  …   0     0
1      1          84  0.110  0.819  0.071   -0.6697      0  …   0     0
2      1          27  0.276  0.690  0.034   -0.9612      0  …   0     0
3      1           6  0.222  0.778  0.000   -0.6449      0  …   0     0
4      1          44  0.055  0.919  0.026   -0.3201      0  …   0     0

   yum  yummy  yup  zero  zone  zoo  zucchini  étouffée
0    0      0    0     0     0    0         0         0
1    0      0    0     0     0    0         0         0
2    0      0    0     0     0    0         0         0
3    0      0    0     0     0    0         0         0
4    0      0    0     0     0    0         0         0

[5 rows x 5008 columns]
```

```
[2]: # Dropping 'review_id', 'user_id', and 'business_id' fields
data.drop(columns=['review_id', 'user_id', 'business_id'], inplace = True)
data.head()
```

```
[2]:    stars  num_Tokens    neg    neu    pos  compound  aaron  aback  abandon  \
0      1          48  0.119  0.779  0.102   -0.0209      0      0        0
1      1          84  0.110  0.819  0.071   -0.6697      0      0        0
2      1          27  0.276  0.690  0.034   -0.9612      0      0        0
3      1           6  0.222  0.778  0.000   -0.6449      0      0        0
4      1          44  0.055  0.919  0.026   -0.3201      0      0        0

   ability  …  yr  yuck  yum  yummy  yup  zero  zone  zoo  zucchini  \
0        0  …   0     0    0      0    0     0     0    0         0
1        0  …   0     0    0      0    0     0     0    0         0
2        0  …   0     0    0      0    0     0     0    0         0
3        0  …   0     0    0      0    0     0     0    0         0
4        0  …   0     0    0      0    0     0     0    0         0

   étouffée
0         0
1         0
2         0
3         0
4         0

[5 rows x 5005 columns]
```

1.2.1 Feature Selection

All three ML models will use the following engineered features: 1) The 4 different sentiment scores generated by VADER, and 3) the Bag of Words matrix. The star rating will be the target variable.

Please note that for this Initial Results and Code section we purposely ignored conducting a comprehensive *feature selection* process. The reason behind this is two-fold:

1. All the features being utilized were engineered and not intrinsic to the original dataset. As such, maintaining all the engineered features in the model training portion could help provide a solid baseline to compare each model's performance.
2. The feature selection process is often algorithm or model specific and as such could introduce variation in which features get preserved in the model training step. This too would skew the results when conducting a performance cross-comparison between all three models.

1.2.2 Training and Testing Setup

For the sake of consistency we will be training and testing all 3 models using the same methods:

- Utilize an 80% training size (equivalent to 20,000 rows of data)
- Utilize a 20% test size (equivalent to 5,000 rows of data)

1.3 Model Training and Implementation

1.3.1 Preloading all dependent libraries

```
[ ]: from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     from sklearn.naive_bayes import MultinomialNB
     from sklearn.metrics import classification_report, confusion_matrix,␣
      ↪accuracy_score, mean_squared_error, r2_score
     from sklearn.svm import LinearSVR
     from sklearn.preprocessing import StandardScaler
```

1.3.2 1. Model Training for Logistic Regression Algorithm

```
[ ]: from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import classification_report, confusion_matrix,␣
      ↪accuracy_score

     # Defining the Target Variable (Y = 'stars' column) and Feature Set (X = all␣
      ↪other columns)
     Y = data['stars']
     X = data.drop(columns = 'stars')

     # Splitting the data into training and testing components
     X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
      ↪random_state=2025)

     # Specifying a Logistic Regression algorithm
     model = LogisticRegression(max_iter=5000, solver='lbfgs',␣
      ↪multi_class='multinomial')
```

```
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
c:\Users\Sunora\AppData\Local\Programs\Python\Python313\Lib\site-
packages\sklearn\linear_model\_logistic.py:1272: FutureWarning: 'multi_class'
was deprecated in version 1.5 and will be removed in 1.7. From then on, it will
always use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
```

```
Accuracy: 0.5154
Confusion Matrix:
 [[659 233  63  24  23]
 [234 439 224  53  26]
 [ 69 238 396 238  80]
 [ 23  53 182 428 313]
 [ 15  30  62 240 655]]
Classification Report:
               precision    recall  f1-score   support

           1       0.66      0.66      0.66      1002
           2       0.44      0.45      0.45       976
           3       0.43      0.39      0.41      1021
           4       0.44      0.43      0.43       999
           5       0.60      0.65      0.62      1002

    accuracy                           0.52      5000
   macro avg       0.51      0.52      0.51      5000
weighted avg       0.51      0.52      0.51      5000
```

Notes: Based on the Confusion Matrix output for our **Logistic Regression model**, we quickly observe the following:

- **Precision**, a metric that measures how accurate a model is at categorizing star-ratings, was highest for 1-star and 5-star reviews but suffered between 2-4 star ratings. For instance, the model scored a 66% on precision for 1-star ratings which means that 66% of the model's predictions for 1-star ratings were truly 1-stars. In other words, it can be thought of as the probability of a model's prediction being correct.

- **Recall**, a metric that measures how complete a model is at capturing the right star-ratings, was also highest for 1-star and 5-star reviews and also suffered between 2-4 star ratings. For instance, the model scored a 66% on recall for 1-star ratings which means that 66% of all 1-star ratings were identified.

- **F1-Score**, a harmonized metric that takes the average of the **precision** and **recall** score, is therefore also aligned in that both 1-star and 5-star categories had the highest score.

- **Accuracy**, a metric that measures the overall correctness of a model across all categories, was 52%, indicating that the model correctly predicted just over half of all reviews. While this is better than random guessing (which would yield ~20% in a 5-class setup), it still reflects substantial room for improvement.

1.3.3 2. Model Training for Naive Bayes Algorithm

[ ]:
```
# Importing MultinomialNB from sklearn.naive_bayes
from sklearn.naive_bayes import MultinomialNB

# Revoving negatives from compound score to use Naive Bayes
data_nb = data.copy()
data_nb['compound'] = data_nb['compound'] + 1  # Shift to non-negative

# Defining the Target Variable (Y = 'stars' column) and Feature Set (X = all␣
↪other columns)
Y = data_nb['stars']
X = data_nb.drop(columns = 'stars')

# Splitting the data into training and testing components
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
↪random_state=2025)

# Train the Naive Bayes model
nb_model = MultinomialNB()
nb_model.fit(X_train, Y_train)

# Make predictions with the Naive Bayes model
nb_y_pred = nb_model.predict(X_test)

print("Accuracy:", accuracy_score(Y_test, nb_y_pred))
print("Confusion Matrix:\n", confusion_matrix(Y_test, nb_y_pred))
print("Classification Report:\n", classification_report(Y_test, nb_y_pred))
```

```
Accuracy: 0.5226
Confusion Matrix:
 [[670 239  73   6  14]
 [215 387 287  61  26]
 [ 86 164 427 280  64]
 [ 45  28 156 515 255]
 [ 53  14  43 278 614]]
Classification Report:
               precision    recall  f1-score   support

           1       0.63      0.67      0.65      1002
           2       0.47      0.40      0.43       976
           3       0.43      0.42      0.43      1021
           4       0.45      0.52      0.48       999
           5       0.63      0.61      0.62      1002

    accuracy                           0.52      5000
   macro avg       0.52      0.52      0.52      5000
weighted avg       0.52      0.52      0.52      5000
```

Notes: Based on the Confusion Matrix output for our **Naive Bayes model**, we quickly observe the following:

- **Precision**, a metric that measures how accurate a model is at categorizing star-ratings, was highest for 1-star and 5-star reviews but suffered between 2-4 star ratings. For instance, the model scored a 63% on precision for 1-star ratings which means that 63% of the model's predictions for 1-star ratings were truly 1-stars. In other words, it can be thought of as the probability of a model's prediction being correct.
- **Recall**, a metric that measures how complete a model is at capturing the right star-ratings, was also highest for 1-star and 5-star reviews and also suffered between 2-4 star ratings. For instance, the model scored a 67% on recall for 1-star ratings which means that 67% of all 1-star ratings were identified.
- **F1-Score**, a harmonized metric that takes the average of the **precision** and **recall** score, is therefore also aligned in that both 1-star and 5-star categories had the highest score.
- **Accuracy**, a metric that measures the overall correctness of a model across all categories, was 52%, indicating that the model correctly predicted just over half of all reviews. While this is better than random guessing (which would yield ~20% in a 5-class setup), it still reflects substantial room for improvement.

1.3.4 3. Model Training for Regression-Support Vector Machine Algorithm

[4]:
```
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVR
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
import numpy as np

# Defining the Target Variable (Y = 'stars' column) and Feature Set (X = all␣
 ↪other columns)
Y = data['stars']
X = data.drop(columns = 'stars')

# Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Splitting the data into training and testing components
```

```python
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y, test_size=0.2,␣
 ↪random_state=2025)

# Specifying a Support Vector Regression algorithm
svr_model = LinearSVR(max_iter = 10000)
svr_model.fit(X_train, Y_train)

# Make predictions with the SVR model
svr_y_pred = svr_model.predict(X_test)

# Evaluating the SVR model
print("Mean Squared Error:", mean_squared_error(Y_test, svr_y_pred))
print("R^2 Score:", r2_score(Y_test, svr_y_pred))

# Rounding predictions to nearest integer for classification evaluation
svr_y_pred_rounded = np.clip(np.round(svr_y_pred), 1, 5).astype(int)
print("Rounded Accuracy:", accuracy_score(Y_test, svr_y_pred_rounded))
print("Confusion Matrix:\n", confusion_matrix(Y_test, svr_y_pred_rounded))
print("Classification Report:\n", classification_report(Y_test,␣
 ↪svr_y_pred_rounded))
```

```
Mean Squared Error: 1.364657920850164
R^2 Score: 0.3170491581973893
Rounded Accuracy: 0.4018
Confusion Matrix:
 [[399 387 178  29   9]
 [212 368 301  77  18]
 [ 61 212 426 259  63]
 [ 15  53 260 449 222]
 [  7  22 162 444 367]]
Classification Report:
               precision    recall  f1-score   support

           1       0.57      0.40      0.47      1002
           2       0.35      0.38      0.36       976
           3       0.32      0.42      0.36      1021
           4       0.36      0.45      0.40       999
           5       0.54      0.37      0.44      1002

    accuracy                           0.40      5000
   macro avg       0.43      0.40      0.41      5000
weighted avg       0.43      0.40      0.41      5000


/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-
packages/sklearn/svm/_base.py:1250: ConvergenceWarning: Liblinear failed to
converge, increase the number of iterations.
  warnings.warn(
```

Notes: Based on the Confusion Matrix output for our **Support Vector Regression model**, we quickly observe the following:

- **Precision**, a metric that measures how accurate a model is at categorizing star-ratings, was highest for 1-star and 5-star reviews but suffered between 2-4 star ratings. For instance, the model scored a 57% on precision for 1-star ratings which means that 57% of the model's predictions for 1-star ratings were truly 1-stars. In other words, it can be thought of as the probability of a model's prediction being correct.
- **Recall**, a metric that measures how complete a model is at capturing the right star-ratings, was highest for 4-star reviews. For instance, the model scored a 45% on recall for 4-star ratings which means that 45% of all 4-star ratings were identified.
- **F1-Score**, a harmonized metric that takes the average of the **precision** and **recall** score, showed that both 1-star and 5-star categories had the highest harmonized score.
- **Accuracy**, a metric that measures the overall correctness of a model across all categories, was 40%. This suggests that the SVR model underperformed both the Logistic Regression and Naive Bayes models.

1.4 Key Results and Takeaways

- The Naive Bayes and Multinomial Logistic Regression models performed equivalently whilst the Support Vector Regression model lagged behind
- The Naive Bayes model had the quickest run-time suggesting computational efficiency, while the two regression models were more time-consuming
- Even with 10,000 iterations, the Support Vector Regression failed to converge whilst the Multinomial Logistic Regression model was able to converge with a 5,000 iteration limit
- Overall, all models performed poorly and would require performance tuning.

1.5 Next Steps

- Implement feature selection to reduce the dimensionality of the dataset and to remove highly correlated features in efforts to reduce noise. This may help with performance tuning of the models
- One viable tool for dimensionality reduction would be to conduct a Chi-squared test on the BoW Matrix setting the 'Stars' column as the target variable. This could help in determine which tokenized words are most influential in determining the star-rating. This would be a next-step and could be as a universal feature selection method applicable for all 3 models.

References

Liu, B. (2020). Sentiment Analysis: Mining Opinion, Sentiments, and Emotions. Cambridge University Press. https://doi-org.ezproxy.lib.torontomu.ca/10.1017/9781108639286

Feldman, R. (2013). Techniques and applications for sentiment analysis. *Communications of the ACM, 56(4),* 82-89. https://doi.org/10.1145/2436256.2436274

Ding, X., Liu, B., & Yu, P. (2008). A holistic lexicon-based approach to opinion mining. *Proceedings of the 2008 International Conference on Web Search and Data Mining*, 231-240. https://doi-org.ezproxy.lib.torontomu.ca/10.1145/1341531.1341561

Hutto, C., & Gilbert, E. (2014). VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. Proceedings of the International AAAI Conference on Web and Social Media, 8(1), 216-225. https://doi.org/10.1609/icwsm.v8i1.14550

Kumar, V., & Subba, B. (2020). A TfidfVectorizer and SVM based sentiment analysis framework for text data corpus. *2020 National Conference on Communications (NCC)*, 1-6. 10.1109/NCC48643.2020.9056085

Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs Up? Sentiment Classification Using Machine Learning Techniques. *In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP-2002).* https://doi.org/10.48550/arXiv.cs/0205070

Pang, B., & Lee, L. (2005). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. *[Pre-print] Proceedings of ACL 2005*. https://doi.org/10.48550/arXiv.cs/0506075